IBM

Введение в современные мэйнфреймы: основы z/OS

- Основные понятия в сфере мэйнфреймов, включая использование и архитектуру
- Основы z/OS для студентов и начинающих
- Оборудование и периферийные устройства мэйнфреймов

Майк Эбберс
Уэйн О'Брайен
Билл Огден

ibm.com/redbooks

Redbooks

IBM
International Technical Support Organization

Introduction to the New Mainframe: z/OS Basics

Mike Ebbers
Wayne O'Brien
Bill Ogden

July 2006

Note: Before using this information and the product it supports, read the information in “Notices”.

First Edition (July 2006)

IBM

Международная организация технической поддержки

Введение в современные мэйнфреймы: основы z/OS

Майк Эбберс
Уэйн О'Брайен
Билл Огден

Москва 2007

Перед использованием данного руководства и продукта ознакомьтесь с информацией в разделе «Примечания».

Перевод
М. Аницкий

Научный редактор
Инструктор авторизованного учебного центра IBM
Д. Данилов

Первое издание на русском языке (2007 г.)

Содержание

Предисловие

Эта книга из серии IBM® Redbooks содержит основные сведения и практическую информацию для студентов, специализирующихся в сфере информационных технологий, позволяющие начать использование основных средств мэйнфрейм-компьютеров. Она является первым из запланированной серии учебников, предназначенных для ознакомления студентов с основными понятиями мэйнфреймов и для их подготовки к деятельности в сфере вычислений с использованием больших систем.

Оптимальный процесс обучения предполагает, что студенты успешно прошли вводный курс по основам компьютерных систем, в частности по устройству и архитектуре компьютеров, операционным системам, управлению данными или системам передачи данных. Кроме того, они должны были успешно пройти курсы по одному или нескольким языкам программирования и уметь работать с компьютером на уровне пользователя.

Этот учебник также можно использовать для подготовки к курсам по расширенным задачам, а также для стажировки и специальных исследований. Он не претендует на полноту изложения всех аспектов функционирования мэйнфреймов, равно как и не является справочником, описывающим все функции и опции мэйнфреймов.

Кроме того, этот курс может быть полезен для опытных специалистов в обработке данных, работавших с платформами, отличными от мэйнфреймов, или знакомых с некоторыми аспектами мэйнфреймов, но желающих ознакомиться с другими функциями и преимуществами среды мэйнфреймов.

В процессе изучения этого курса преподавателю рекомендуется чередовать работу с учебником, чтение лекций, обсуждения и практические упражнения. Многие из упражнений являются сводными и имеют цель показать студенту, каким образом осуществляется разработка и реализация рассматриваемой задачи. Обсуждения с преподавателем и практические упражнения являются неотъемлемой частью курса и могут затрагивать темы, не рассматриваемые в этом учебнике.

В этом курсе используются упрощенные примеры и рассматриваются главным образом базовые системные функции. Курс содержит практические примеры, предназначенные для того, чтобы помочь студентам в изучении вычислений на мэйнфреймах.

К концу этого курса вы получите следующие знания:

- основные понятия о мэйнфрейме, включая его использование и архитектуру;
- основные сведения о z/OS® – широко используемой операционной системе для мэйнфреймов;
- представление о рабочей нагрузке на мэйнфреймы и основных приложениях промежуточного программного обеспечения, используемых на мэйнфреймах в настоящее время;

- основу для последующей курсовой работы по более сложным специализированным аспектам z/OS, в частности по системному администрированию или программированию приложений

Каким образом организован материал книги

Материал книги разделен на четыре части следующим образом:

- **Часть 1, «Введение в z/OS и среду мэйнфрейма»**, содержит обзор типов рабочей нагрузки, обычно обрабатываемой на мэйнфрейме, в частности описание пакетных заданий и оперативных транзакций. Эта часть материала помогает студентам в изучении пользовательских интерфейсов z/OS – широко используемой операционной системы для мэйнфреймов. Рассматриваются такие вопросы, как TSO/E и ISPF, интерфейсы UNIX®, язык управления заданиями, файловые структуры и подсистемы ввода заданий. Особое внимание обращается на пользователей мэйнфреймов и на изменение роли мэйнфреймов в современном деловом мире.
- **Часть 2, «Программирование приложений в z/OS»**, описывает инструменты и утилиты, предназначенные для разработки простой программы для выполнения в z/OS. Эта часть книги иллюстрирует процесс проектирования приложения, выбора языка программирования и использования среды выполнения.
- **Часть 3, «Оперативная рабочая нагрузка на z/OS»**, рассматривает основные категории интерактивной рабочей нагрузки, обрабатываемой в z/OS, в частности обработку транзакций, управление базами данных и веб-обработку. Эта часть включает описание нескольких популярных продуктов промежуточного программного обеспечения, включая DB2®, CICS® и WebSphere® Application Server.
- **Часть 4, «Системное программирование в z/OS»**, рассматривает вопросы, позволяющие студенту получить представление о роли системного программиста z/OS. Эта часть материала включает описания системных библиотек, запуска и остановки системы, безопасности, сетевых коммуникаций и кластеризации на основе нескольких систем. Кроме того, она содержит обзор аппаратных систем мэйнфреймов, включая процессоры и устройства ввода-вывода.

В книге используются упрощенные примеры и рассматриваются главным образом базовые системные функции. Курс содержит практические примеры, предназначенные для того, чтобы помочь студентам в изучении вычислений на мэйнфреймах. Упражнения включают ввод заданий в систему, проверку их состояния и изучение выходных данных запущенных заданий.

Каким образом организована каждая глава

Каждая глава имеет одинаковый формат:

- цели для студента;
- вопросы, освещающие основную тему, связанную с вычислениями на мэйнфреймах;

- обзор основных идей, рассматриваемых в главе;
- список основных терминов, представленных в главе;
- контрольные вопросы, позволяющие студентам проверить понимание материала;
- темы для дальнейшего обсуждения, стимулирующие изучение студентами вопросов, выходящих за рамки целей главы;
- практические упражнения, позволяющие студентам закрепить понимание материала.

Об авторах

Этот материал был разработан техническими специалистами, работающими в International Technical Support Organization, Покипси (Poughkeepsie).

Майк Эбберс (Mike Ebbers) работает с мэйнфрейм-системами в IBM на протяжении 32 лет. В это время он в том числе вел курсы практического использования мэйнфреймов для новых сотрудников, поступивших на работу сразу после колледжа. В настоящее время Майк занимается созданием серии IBM Redbooks™ – популярного набора документации, расположенного по адресу

`http://www.ibm.com/redbooks`

Уэйн О'Брайен (Wayne O'Brien) – инженер-консультант по программному обеспечению в IBM, Покипси (Poughkeepsie). После перехода в IBM в 1988 году он занимался разработкой руководств пользователя и электронной справочной документации для многочисленных программных продуктов. Уэйн имеет степень магистра наук в области технических коммуникаций, полученную в Политехническом институте Ренселера (Rensselaer Polytechnic Institute, RPI) в г. Трой, штат Нью-Йорк.

Билл Огден (Bill Ogden) – бывший представитель старшего технического персонала компании IBM, теперь на пенсии. Имеет степени бакалавра наук в области электротехники и магистра наук в области компьютерных наук, работает с мэйнфреймами с 1962 года, а с системой z/OS с тех времен, когда она имела название OS/360 Release 1/2. После перехода в ITSO в 1978 году Билл специализировался в работе с пользователями, впервые работающими с операционной системой и соответствующим аппаратным обеспечением.

Благодарности

Хотим выразить благодарность следующим людям за их вклад в этот проект.

Дэн Андрасик (Dan Andrascik) – студент последнего курса Университета штата Пенсильвания (Pennsylvania State University), специализирующийся в информационных науках и технологиях. Дэн является специалистом в компьютерных языках (C++, Visual Basic®, HTML, XML, SQL), теории организации, теории и проектировании баз данных, а также в планировании и управлении проектами. Во время стажировки в ITSO в г. Покипси (Poughkeepsie) Дэн интенсивно работал с элементами платформы zSeries®.

Рама Айар (Rama Ayyar) – старший специалист по информационным технологиям в центре поддержки IBM в Сиднее, Австралия. Он имеет 20-летний опыт работы с операционной системой MVS™ и работает в сфере информационных технологий на протяжении 30 лет. Его область знаний включает TCP/IP, безопасность, управление хранением, управление конфигурациями и определение проблем. Рама имеет степень магистра в области компьютерных наук, полученную в Индийском институте технологий, г. Канпур.

Эмиль Сиполла (Emil T. Cipolla) – консультант по информационным системам в Соединенных Штатах Америки с 40-летним опытом работы в области информационных систем. Он имеет степени магистра машиностроения и бизнес-администрирования, полученные в Корнелльском университете. В настоящее время Эмиль является преподавателем-стажером колледжа.

Марк Даубман (Mark Daubman) – студент последнего курса Университета Св. Бонавентуры, специализирующийся в деловых информационных системах и изучающий компьютерные науки в качестве предмета второй специализации. Во время стажировки в IBM Марк интенсивно работал со множеством интерфейсов z/OS, описанных в этом учебнике. После дипломирования Марк планирует работать в области мэйнфреймов.

Мириам Дюамель (Myriam Duhamel) – специалист по информационным технологиям в Бельгии. Она имеет 20-летний опыт работы в сфере разработки приложений и 12 лет работает в IBM. Ее область знаний включает разработку приложений в различных средах z/OS (в частности, COBOL, PL/I, CICS, DB2 и WebSphere MQ). В настоящее время Мириам преподает курсы по DB2 и WebSphere MQ.

Пер Фремстад (Per Fremstad) – специалист по информационным технологиям, сертифицированный по продуктам IBM и работающий в IBM Systems and Technology Group в IBM Norway. Он работает в IBM с 1982 года и имеет обширный опыт в области мэйнфреймов и z/OS. Его область знаний включает веб-технологии, WebSphere для z/OS и обеспечение доступа к среде z/OS через веб. Он часто преподает курсы по темам z/OS, zSeries и WebSphere для z/OS. Пер имеет степень бакалавра наук, полученную в Университете Осло, Норвегия.

Луис Мартинес Фуентес (Luis Martinez Fuentes) – сертифицированный специалист-консультант по информационным технологиям (в области интеграции данных) в IBM Systems and Technology Group в IBM Spain. Он имеет 20-летний опыт работы с мэйнфреймами IBM, главным образом в сфере CICS и DB2. В настоящее время он работает в группе технической поддержки продаж и занимается новыми видами рабочих нагрузок на мэйнфреймы. Луис является членом Пиренейского совета технических экспертов (Iberia Technical Expert Council), аффилированного с Академией технологий IBM (IBM Academy of Technology). Луис преподает курсы по мэйнфреймам в двух университетах в Мадриде.

Мириам Гелински (Miriam Gelinski) – штатный сотрудник Maffei Consulting Group в Бразилии, где она отвечает за поддержку планирования клиентов и установку программного обеспечения мэйнфреймов. Она имеет 5-летний опыт работы с мейнфреймами. Мириам имеет степень бакалавра в области информационных систем, полученную в Университете Св. Марка (Universidade São Marcos) в Сан-Паулу. Ее область знаний включает операционную систему z/OS, ее подсистемы, а также TSO и ISPF.

Майкл Гроссманн (Michael Grossmann) – специалист по образованию в области информационных технологий в Германии, имеющий 9-летний опыт работы в качестве системного программиста z/OS и преподавателя. Его область знаний включает преподавание z/OS для начинающих, управление z/OS, автоматизацию, оборудование мэйнфреймов и Parallel Sysplex®.

Олегарио Хернандез (Olegario Hernandez) – бывший инженер-консультант по системам IBM в Чили. Он имеет более 35 лет опыта в проектировании и разработке приложений для мэйнфрейм-систем. Он написал множество публикаций об интерфейсе приложений CICS, управлении системами и grid-вычислениях. Олегарио имеет степень в области химического машиностроения, полученную в Университете Чили.

Роберто Юити Хиратзука (Roberto Yuiti Hiratzuka) – системный программист MVS в Бразилии. Он имеет 15-летний опыт работы системным программистом мэйнфреймов. Роберто имеет степень в области информационных систем, полученную в Faculdade de Tecnologia Sao Paulo (FATEC-SP).

Джон Кеттнер (John Kettner) – консультант-разработчик архитектуры программного обеспечения в zSeries Advanced Architecture Group. Он имеет 30-летний опыт работы с мэйнфреймами и обладает степенью бакалавра компьютерных наук, полученной в Университете Лонг-Айленда. Его специализация включает внутреннее устройство zSeries, интеграцию с WebSphere и планирование мощностей. Джон написал несколько книг из серии Redbooks и участвовал в различных образовательных программах в IBM.

Георг Мюллер (Georg Müller) – студент Университета Лейпцига в Германии. Он имеет 3-летний опыт работы в области z/OS и оборудования мэйнфреймов. Он планирует окончить обучение в следующем году со степенью магистра в компьютерных науках. Для этого учебника Георг написал разделы, посвященные WebSphere MQ и HTTP Server, составил примеры программ и помог проверить итоговую последовательность учебных модулей.

Род Нойфельд (Rod Neufeld) – старший специалист по техническому обслуживанию в Канаде. Он имеет 25-летний опыт работы в системном программировании для MVS и z/OS. Его область знаний включает системное программное обеспечение и поддержку z/OS, Parallel Sysplex, а также обеспечение непрерывности работы и восстановление. Род имеет степень бакалавра наук с отличием, полученную в Университете Манитобы.

Пол Ньютон (Paul Newton) – старший разработчик программного обеспечения в IBM Developer Relations Technical Support Center (Даллас, Техас). Он имеет 25-летний опыт работы с операционными системами мэйнфреймов IBM, подсистемами и сетями данных. Пол имеет степень в бизнес-администрировании, полученную в Университете Аризоны.

Билл Сьюберт (Bill Seubert) – разработчик архитектуры программного обеспечения для zSeries в Соединенных Штатах Америки. Он имеет более 20 лет опыта работы в области мэйнфреймов и распределенных вычислений. Он имеет степень бакалавра в области компьютерных наук, полученную в Университете Миссури, Колумбия. Его область знаний включает z/OS, интеграционное программное обеспечение WebSphere и архитектуру программного обеспечения. Билл часто общается с клиентами IBM по поводу архитектуры интеграции и модернизации предприятий.

Хенрик Торсен (Henrik Thorsen) – старший специалист-консультант по информационным технологиям в IBM Denmark. Он имеет 25-летний опыт работы с мэйнфреймами, а также имеет степень магистра наук в машиностроении, полученную в Техническом университете в Копенгагене, и степень бакалавра наук в экономике, полученную в Копенгагенской школе бизнеса. Он специализируется в z/OS, Parallel Sysplex, высокой доступности, планировании производительности и мощностей. Хенрик является автором нескольких книг серии IBM Redbooks и прочих документов, а также участником различных образовательных программ в IBM и участником технического сообщества zSeries.

Энди Уилкинсон (Andy R. Wilkinson) – специалист по информационным технологиям в Великобритании. Он имеет 25-летний опыт работы с системами резервирования и системным программированием для z/OS и 6 лет работает в IBM. Его область знаний включает конфигурирование аппаратного обеспечения и SMP/E. Энди имеет степень в области материаловедения и техники, полученную в Университете Шеффилда, а также степень в области вычислительной техники, полученную в Открытом университете.

И наконец, выражаем особую благодарность редакторам центра ITSO в Покипси, Нью-Йорк:

- **Терри Бартел (Terry Barthel);**
- **Элла Буслович (Ella Buslovich, графика);**
- **Альфред Шваб (Alfred Schwab).**

Мы будем рады вашим отзывам

Ваши отзывы очень важны для нас!

Мы стремимся сделать наши книги максимально полезными. Направляйте нам ваши отзывы об этой книге или других книгах из серии IBM Redbooks, используя один из следующих способов:

– используйте веб-форму **Contact us** по адресу

ibm.com/redbooks

– направляйте ваши отзывы по электронной почте

redbook@us.ibm.com

– отправляйте ваши отзывы по адресу:

IBM Corporation, International Technical Support Organization

Dept. HYJ Mail Station P099

2455 South Road

Poughkeepsie, NY 12601-5400

Часть 1

Введение в z/OS и среду мэйнфрейма

Добро пожаловать в мир вычислений на мэйнфреймах! Мы начнем с общих сведений о мэйнфрейм-компьютере и его месте в современной организации сферы информационных технологий. Мы рассмотрим, почему государственные и частные предприятия по всему миру используют мэйнфреймы в качестве основы для крупномасштабных вычислений. Мы обсудим типы рабочей нагрузки, обычно связанные с мэйнфреймами, в частности пакетные задания и оперативные или интерактивные транзакции, а также уникальный способ обработки этой нагрузки с помощью широко используемой операционной системы для мэйнфреймов – z/OS.

В ходе описания мы будем обращать особое внимание на людей, использующих мэйнфреймы, и на роль современных мэйнфреймов в деловом мире.

1

Введение в современные мэйнфреймы

Цель. Как техническому специалисту в мире мэйнфрейм-вычислений, вам необходимо понять, какую роль мэйнфрейм-компьютеры играют в поддержке информационной инфраструктуры и бизнес-целей вашей компании. Вам также нужно знать названия должностей различных участников группы поддержки мэйнфреймов в вашей компании.

После завершения работы над этой главой вы сможете:

- описать, каким образом современные мэйнфреймы оспаривают традиционное представление о сопоставлении централизованных вычислений и распределенных вычислений;
- объяснить, каким образом предприятия используют вычислительную мощность мэйнфреймов, перечислить области применения мэйнфреймов и отличия мэйнфрейм-вычислений от других типов вычислений;
- описать основные типы рабочей нагрузки, для которых использование мэйнфреймов подходит лучше всего;
- назвать пять должностей или описать должностные обязанности, связанные с мэйнфрейм-вычислениями;
- назвать четыре операционных системы для мэйнфреймов.

1.1 Современные мэйнфреймы

В настоящее время мэйнфрейм-компьютеры занимают центральное место в повседневной работе большинства крупнейших корпораций мира, включая многие компании из списка Fortune 1000. Несмотря на то что другие виды вычислительной техники широко используются в различных сферах деятельности, мэйнфреймы занимают завидное место в современной среде электронного бизнеса. В банковских, финансовых, здравоохранительных, страховых, коммунальных, правительственных и многих других государственных и частных учреждениях мэйнфрейм-компьютеры представляют основу современного бизнеса.

Электронный бизнес – ведение бизнеса посредством использования электронных средств связи, в частности Интернета

Длительный успех мэйнфрейм-компьютеров является беспрецедентным в отрасли информационных технологий. Периодические потрясения сотрясают мировую экономику, и постоянные (часто надуманные) изменения в сфере информационных технологий неоднократно приводили к заявлениям о том, что некогда незаменимые инновации становятся жертвами безжалостного прогресса. Как только новые технологии привлекают к себе внимание общественности, многие технологии сразу же становятся устаревшими по причине появления еще более новых усовершенствований. Но даже в наше время, как и в каждом десятилетии, начиная с 1960-х годов, мэйнфрейм-компьютеры и *стиль* мэйнфрейм-вычислений доминируют в области крупномасштабных деловых вычислений.

Почему именно этот тип вычислений занял настолько прочное место во множестве корпораций мира? В этой главе мы рассмотрим причины, по которым мэйнфрейм-компьютеры сохраняют популярность в сфере крупномасштабных деловых вычислений.

1.2 S/360: поворотный пункт в истории мэйнфреймов

Когда именно появились мэйнфрейм-компьютеры? Появление мэйнфрейм-компьютеров произошло в 1950-х, если не раньше. В те времена мэйнфрейм-компьютеры были не просто самыми большими компьютерами; они были *единственными существующими* компьютерами, и лишь некоторые предприятия могли позволить себе их приобретение.

В своем развитии мэйнфреймы прошли несколько *поколений*, начиная с 1950-х годов. Системы первого поколения, такие, как IBM 705 в 1954 и IBM 1401 в 1959, были далеко не такими мощными, как последующие системы, но они однозначно имели свойства мэйнфрейм-компьютеров. Эти компьютеры позиционировались как счетные системы для бизнеса и использовались, как впрочем и сейчас, в качестве центрального хранилища данных в корпоративном центре обработки данных[1].

В 1960-х ход истории вычислительной техники значительно изменился, когда производители мэйнфреймов начали стандартизацию аппаратного и программного обеспечения, предлагаемого клиентам. Появление IBM System/360™ (или S/360™)

[1] В соответствии с описанием продукта, стандартными областями применения 1401 были «ведение ведомостей, учет железнодорожных перевозок, учет клиентов коммунальных служб, управление запасами и учет дебиторской задолженности».

в 1964-м сигнализировало о начале третьего поколения – первых компьютеров общего назначения. Предыдущие системы, такие, как 1401, были предназначены для использования либо в коммерческой, либо в научной сфере. Революционный S/360 мог выполнять оба вида вычислений, если только клиент, компания-производитель программного обеспечения или консультант предоставляли соответствующие программы. В действительности название S/360 указывает на широкую сферу применения архитектуры – 360 градусов, охватывающие полный круг возможных областей использования.

System/360 – первый компьютер общего назначения, выпущенный в 1964 году

S/360 также был первым компьютером, использовавшим *микрокод* для реализации многих машинных инструкций, в отличии от систем, в которых все машинные инструкции были реализованы на аппаратном уровне. Микрокод (или *микропрограммное обеспечение*, как его иногда называют) состоит из хранимых микроинструкций, недоступных для пользователей, которые представляют функциональный уровень между аппаратным и программным обеспечением. Преимущество использования микрокода состоит в гибкости, при которой любое изменение или новую функцию можно реализовать путем простого изменения существующего микрокода, вместо того чтобы заменять компьютер.

Используя стандартизированные мэйнфрейм-компьютеры для обработки рабочей нагрузки, клиенты могли, в свою очередь, создавать бизнес-приложения, не требовавшие специального аппаратного или программного обеспечения. Более того, клиенты свободно могли переходить на новые и более мощные процессоры, не боясь возникновения проблем совместимости с существующими приложениями. Первые бизнес-приложения создавались главным образом на ассемблере, COBOL, FORTRAN или PL/1, и значительное количество этих старых программ до сих пор используется.

В последующие десятилетия мощность мэйнфрейм-компьютеров неизменно росла, вследствие чего они достигли невероятных вычислительных возможностей. Современные мэйнфреймы имеют беспрецедентные возможности обслуживания десятков тысяч конечных пользователей, управления петабайтами данных и реконфигурирования аппаратных и программных ресурсов для приспособления к изменениям в рабочей нагрузке, и все это осуществляется с единого пункта управления.

1.3 Развивающаяся архитектура

Архитектура представляет собой набор определенных терминов и правил, используемых в качестве инструкций при разработке продуктов. В компьютерных науках архитектура описывает организационную структуру системы. Архитектуру можно рекурсивно разделить на части, взаимодействующие через интерфейсы, отношения, связывающие части, и ограничения для компоновки частей. К частям, взаимодействующим через интерфейсы, относятся классы, компоненты и подсистемы.

Архитектура – описывает организационную структуру системы

Начиная с первых больших компьютеров, появившихся в 1960-х и известных под названием «Большое железо» («Big Iron», в отличие от небольших систем уровня отде-

ла компании), каждое новое поколение мэйнфрейм-компьютеров содержало усовершенствования в одной или нескольких частях архитектуры[1]:

- увеличение количества и скорости процессоров;
- увеличение объема физической памяти и возможностей адресации памяти;
- возможности динамического обновления аппаратного и программного обеспечения;
- повышение степени автоматизации проверки аппаратных ошибок и восстановления;
- усовершенствованные устройства ввода-вывода и увеличение количества и скорости путей (*каналов*) между устройствами ввода-вывода и процессорами;
- увеличение функциональности устройств ввода-вывода, в частности сетевых адаптеров с широкими возможностями встроенной обработки;
- повышение возможностей разделения ресурсов одного компьютера между несколькими логически независимыми и изолированными системами, на каждой из которых запущена собственная операционная система;
- усовершенствованные технологии кластеризации, такие, как Parallel Sysplex, а также возможность совместного использования данных несколькими системами.

Несмотря на постоянные изменения, мэйнфрейм-компьютеры остаются самыми стабильными, защищенными и совместимыми среди всех вычислительных платформ. Последние модели могут обрабатывать наиболее сложную и требовательную рабочую нагрузку, продолжая при этом выполнять приложения, написанные в 1970-х и ранее.

Как может технология после таких изменений оставаться настолько стабильной? Это возможно путем развития с целью достижения новых задач. В начале 1990-х появилась клиент-серверная модель вычислений с распределенными узлами, представленными менее мощными компьютерами, которая бросила вызов доминированию мэйнфрейм-компьютеров. Отраслевые эксперты предсказывали скорый конец мэйнфрейм-компьютерам, называя их «динозаврами». В ответ на это разработчики мэйнфреймов сделали то, что они всегда делали при наступлении сложных времен и росте требований пользователей, – они разработали новые мэйнфрейм-компьютеры, соответствующие требованиям. В шутку над «динозавроненавистниками» компания IBM, как ведущий производитель мэйнфрейм-компьютеров, дала своей новой на тот момент мэйнфрейм-системе кодовое имя T-Rex (Tyrannosaurus Rex – тиранозавр).

> Я предсказываю, что отключение последнего мейнфрейма произойдет 15 марта 1996 года.
> *Стюард Олсоп (Stewart Alsop), Infoworld, март 1991 г.*

Расширенные функции и дополнительные уровни возможностей обработки данных, в частности обеспечение веб-обработки, возможности самоуправления, аварийное восстановление и grid-вычисления, обеспечивают готовность мэйнфрейм-компьютера к следующей волне развития

[1] Со времен выпуска S/360 в 1964 году компания IBM приблизительно раз в 10 лет выполняла значительное расширение платформы: System/370™ в 1970, System/370 Extended Architecture (370-XA) в 1983 году, Enterprise Systems Architecture/390® (ESA/390) в 1990 и z/Architecture™ в 2000 году. Дополнительные сведения о более ранних аппаратных мейнфрейм-системах см. в приложении А, «Краткий обзор истории мейнфреймов IBM».

отрасли информационных технологий. Производители мэйнфреймов, такие, как IBM, в очередной раз сообщили о росте годовых продаж в десятках процентов.

И развитие продолжается. Хотя мэйнфрейм-компьютеры сохранили свою традиционную центральную роль в ИТ-организации, эта роль теперь включает использование в качестве основного концентратора в крупнейших распределенных сетях. В действительности Интернет в значительной степени основывается на многочисленных взаимосвязанных мэйнфрейм-компьютерах, используемых в качестве основных концентраторов и маршрутизаторов.

Наблюдая за эволюцией образа мэйнфрейм-компьютера, можно задать вопрос: представляет ли мэйнфрейм-компьютер самостоятельную вычислительную среду, или же он является лишь фрагментом в картине распределенных вычислений? Дело в том, что современный мэйнфрейм представляет собой и то и другое – и самостоятельный вычислительный центр, достаточно мощный для обработки самой большой и разнообразной рабочей нагрузки в одном защищенном месте, и настолько же эффективную систему при использовании в качестве главного сервера в корпоративной распределенной серверной ферме. В сущности, мэйнфрейм-компьютер является определяющим сервером в клиент-серверной модели вычислений.

1.4 Мэйнфреймы среди нас

Несмотря на доминирование мэйнфреймов в деловом мире, эти системы в значительной степени незаметны для общественности, научного сообщества и даже для многих опытных специалистов в информационных технологиях. Другие виды вычислительной техники привлекают больше внимания, по крайней мере с точки зрения общественности. Это и неудивительно. В конце концов, кому из нас нужен непосредственный доступ к мэйнфрейму? А если бы и был нужен, где бы мы его нашли? Правда, однако, состоит в том, что мы *все* являемся пользователями мэйнфреймов, понимаем мы это или нет (об этом ниже).

Большинство из нас, имея некоторые навыки работы с персональным компьютером и достаточно средств, может купить себе ноутбук и сразу же приступить к работе – запускать программы, просматривать веб-страницы и возможно даже писать лабораторные работы для университетских профессоров. При большей усидчивости и технических навыках можно заняться более глубоким изучением средств рабочей станции на основе Intel® и исследованием ее возможностей на практике, используя или не используя многочисленные доступные печатные издания или источники информации в Интернете.

Мэйнфреймы, в свою очередь, склонны оставаться незаметными для общественности. Они надежно выполняют свою работу (обеспечивая почти стопроцентную надежность) и обладают высокой устойчивостью к большинству видов злоупотреблений, затрагивающих персональные компьютеры, в частности к вирусам, передаваемым по электронной почте, и троянским коням. Работая стабильно, спокойно и с незначительными простоями, мэйнфреймы являются эталоном, по которому оценивают все остальные компьютеры. Но в то же время недостаточное внимание приводит к тому, что мэйнфреймы уходят в тень.

Кроме того, при стандартной инсталляции мэйнфрейм работает рядом со многими другими аппаратными устройствами, например внешними системами хранения, аппаратными сетевыми маршрутизаторами, контроллерами каналов и «роботами» автоматизированной библиотеки на магнитных лентах. Современный мэйнфрейм по размерам не больше многих из этих устройств и физически не выделяется из массы периферийных устройств.

Итак, как же можно исследовать практические возможности мэйнфреймов? Как можно научиться работе с мэйнфреймом, изучить его возможности и понять его важность в деловом мире? Крупные корпорации стремятся нанять новых специалистов в мэйнфреймах, но это не так просто, так что приходится рассчитывать на имеющийся опыт.

Узнали бы вы мэйнфрейм, если бы вы его увидели, притом что его эволюция обеспечила ему процветание в ИТ-организации XXI века? Нам необходим опытный проводник в этой *охоте на динозавров*, и здесь нам поможет эта книга.

1.5 Что такое мэйнфрейм?

Во-первых, давайте разберемся с терминологией. Современные производители компьютеров не всегда используют термин *мэйнфрейм* для обозначения мэйнфрейм-компьютеров. Вместо этого большинство из них решило называть любой компьютер для коммерческого использования (независимо от его размера) *сервером*, где мэйнфрейм просто является самым большим типом серверов, используемых в настоящее время. Последний мэйнфрейм компании IBM, например, имеет название «сервер IBM System z9™». В этой книге термин «мэйнфрейм» используется для обозначения компьютеров, способных поддерживать тысячи приложений и устройств ввода-вывода для одновременного обслуживания тысяч пользователей.

Серверная ферма – очень большая группа серверов

Серверы склонны разрастаться. Предприятие может иметь большую группу серверов, включающую серверы транзакций, серверы баз данных, почтовые серверы и веб-серверы. Очень большие группы серверов иногда называют *серверными фермами* (в действительности некоторые информационные центры занимают площади в несколько *акров*). Оборудование сервера может варьироваться от кластера, представленного стойкой персональных компьютеров, до самых мощных мэйнфреймов, производимых в настоящее время.

В корпоративном центре обработки данных мэйнфрейм используется в качестве центрального хранилища данных или *концентратора*, связанного с пользователями посредством менее мощных устройств, таких? как рабочие станции или терминалы. Наличие мэйнфрейма часто предполагает использование централизованной, а не распределенной модели вычислений. Централизованное хранение данных в едином мэйнфрейм-хранилище освобождает клиентов от необходимости осуществлять управление обновлениями для нескольких копий своих коммерческих данных, что повышает вероятность актуальности данных.

Однако различия между централизованной и распределенной моделями вычислений быстро стираются по мере увеличения вычислительной мощности небольших систем и повышения гибкости и универсальности мэйнфреймов. Рыночные условия

требуют от современных предприятий постоянной переоценки своей стратегии в области информационных технологий и поиска более эффективных способов поддержки постоянно изменяющегося рынка. В результате в настоящее время мэйнфреймы часто используются в сочетании с сетями менее крупных серверов в различных конфигурациях. Возможность динамической реконфигурации аппаратных и программных ресурсов мэйнфрейма (в частности, процессоров, памяти и соединений между устройствами) без прерывания работы приложений еще больше подчеркивает гибкую, эволюционирующую природу современного мэйнфрейма.

Сложность классификации относится не только к оборудованию мэйнфреймов, но и к операционным системам, работающим на мэйнфреймах. В былые времена эти термины были взаимно определяющими – мэйнфреймом называлась любая аппаратная система, на которой выполнялась основная операционная система компании IBM[1]. В последние годы такое значение термина утратило актуальность, так как эти операционные системы могут работать на очень небольших системах.

Платформа – компьютерная архитектура (аппаратное и программное обеспечение)

Производители компьютеров и специалисты в информационных технологиях часто используют термин *платформа* для обозначения аппаратного и программного обеспечения, связанного с определенной компьютерной архитектурой. Например, мэйнфрейм-компьютер и его операционная система (и их предшественники[2]) считаются платформой; UNIX на системе RISC (Reduced Instruction Set Computer) считается отдельной платформой, вне зависимости от того, какая именно система RISC используется; персональные компьютеры можно рассматривать как несколько разных платформ, в зависимости от используемой операционной системы.

Мейнфрейм – большая компьютерная система, используемая для размещения баз данных, серверов транзакций и приложений, требующих высокого уровня безопасности и доступности

Теперь давайте вернемся к нашему вопросу: «Что такое мэйнфрейм?» В настоящее время термин «мэйнфрейм» лучше всего использовать для описания *стиля* операций, приложений и средств операционной системы. Для начала можно использовать следующее определение: «мэйнфрейм – это система, используемая предприятиями для размещения коммерческих баз данных, серверов транзакций и приложений, требующих более высокого уровня безопасности и доступности, чем обычно обеспечивают системы меньшего размера».

Первые мэйнфрейм-системы размещались в огромных, размером с комнату, металлических ящиках или рамах (*англ.* frame), откуда, возможно, и произошел термин. Первые мэйнфреймы требовали больших затрат электроэнергии и вентиляции, и комната была заполнена главным образом устройствами ввода-вывода. Кроме того, обычно в вычислительном центре устанавливалось несколько мэйнфреймов, и большинство устройств ввода-вывода подключалось ко всем мэйнфрей-

[1] Это название традиционно применялось и для больших компьютерных систем других изготовителей.

[2] IBM System/390® (S/390®) обозначает серию систем, на место которых пришли системы IBM zSeries. Тем не менее многие системы S/390 все еще используются. Поэтому следует помнить о том, что несмотря на то что в этой книге мы рассматриваем системы zSeries, почти все, что здесь обсуждается, относится и к системам S/390. Единственным важным исключением является 64-разрядная адресация, используемая только в zSeries.

мам. Во времена своих максимальных размеров стандартный мэйнфрейм занимал от 2 000 до 10 000 квадратных футов (от 600 до 3 000 квадратных метров). Некоторые инсталляции были даже еще больше.

Начиная где-то с 1990-х процессоры мэйнфреймов и большая часть устройств ввода-вывода стали меньше по размерам, тогда как их функциональность и мощность продолжали расти. Современные мэйнфрейм-системы гораздо меньше, чем более ранние системы, и имеют размер большого холодильника.

В некоторых случаях можно запустить операционную систему мэйнфрейма на персональном компьютере, эмулирующем мэйнфрейм. Такие эмуляторы полезны для разработки и тестирования бизнес-приложений перед их переносом в рабочую среду мэйнфрейма.

Очевидно, что в настоящее время термин «мэйнфрейм» имеет более широкое значение, описывающее больше, чем просто физические характеристики системы. Теперь это слово означает некоторое сочетание следующих свойств:

- Совместимость с операционными системами мэйнфреймов, мэйнфрейм-приложениями и данными.
- Централизованное управление ресурсами.
- Оборудование и операционные системы, поддерживающие разделение доступа к дисковым приводам с другими системами, обеспечивающие автоматическую блокировку и защиту от повреждений вследствие одновременного использования дисковых данных.
- *Стиль* операций, часто предполагающий наличие специального операторского персонала, использующего подробные *руководства по процедурам операций* и в высшей степени организованные процедуры резервного копирования, восстановления, тренинга и аварийного восстановления на альтернативной системе.
- Оборудование и операционные системы, постоянно работающие с сотнями или тысячами одновременных операций ввода-вывода.
- Технологии кластеризации, позволяющие клиенту работать с несколькими копиями операционной системы как с единой системой. Такая конфигурация, имеющая название Parallel Sysplex, аналогична UNIX-кластеру, однако позволяет добавлять или удалять системы по мере необходимости без прерывания работы приложений. Такая гибкость позволяет владельцам мэйнфреймов добавлять новые приложения или прерывать использование существующих приложений в зависимости от изменений деловой активности.
- Дополнительные возможности совместного использования данных и ресурсов. В Parallel Sysplex, например, пользователи различных систем могут одновременно осуществлять доступ к одним и тем же базам данных, при этом управление доступом к базам данных выполняется на *уровне записей*.

Улучшение показателей производительности и стоимости таких аппаратных ресурсов, как центральный процессор и внешние устройства хранения, и увеличение количества и разнообразия устройств, подключаемых к центральному процессору, позволяет обеспечить более эффективное использование возможностей усовершенствованного оборудования программным обеспечением операционной системы. Кроме того, постоянное усовершенствование функциональных возможностей программного обеспечения позволяет направить разработку каждого нового поколения аппаратных систем.

1.6 Кто использует мэйнфреймы?

Итак, кто же использует мэйнфреймы? Практически *каждый* в той или иной степени использовал мэйнфрейм-компьютеры. Если вы когда-либо пользовались банкоматом для выполнения операций со своим банковским счетом, вы использовали мэйнфрейм.

В настоящее время мэйнфрейм-компьютеры занимают центральное место в повседневной работе большинства крупнейших корпораций мира. Несмотря на то что другие виды вычислительной техники широко используются в различных сферах деятельности, мэйнфреймы занимают завидное место в современной среде электронного бизнеса. В банковских, финансовых, здравоохранительных, страховых, коммунальных, правительственных и многих других государственных и частных предприятиях мэйнфрейм-компьютеры представляют основу современного бизнеса.

До середины 1990-х мэйнфреймы представляли собой *единственное* приемлемое средство обеспечения требований обработки данных крупного предприятия. В основе этих требований в те времена (а часто и сейчас) лежали большие и сложные пакетные задачи, например обработка платежных ведомостей и главных бухгалтерских книг.

Популярность и долговечность мэйнфреймов в значительной степени обусловлена свойственной им надежностью и стабильностью, которые являются результатом продуманных технологических усовершенствований, выполненных со времени выпуска System/360 в 1964 году. Ни одна другая компьютерная архитектура не может заявить о таком постоянном эволюционном развитии при обеспечении совместимости с предыдущими моделями.

По причине вышеперечисленных преимуществ архитектуры мэйнфреймы часто используются в ИТ-организациях для размещения *критически важных* приложений. К таким приложениям обычно относят обработку заказов клиентов, финансовые транзакции, управление производством и инвентаризацией, ведение платежных ведомостей, а также многие другие виды рабочей нагрузки.

Когда речь идет о пользовательском интерфейсе мэйнфрейма, часто представляют терминал с 80x24-символьным «зеленым экраном», названным так за зеленый цвет символов, присущий старым ЭЛТ-мониторам. В действительности пользовательский интерфейс мэйнфрейма во многом похож на интерфейс персональных компьютеров или UNIX-систем. В то время как пользователь осуществляет доступ к бизнес-приложению через веб-браузер, для выполнения критических функций часто неявным образом используется мэйнфрейм-компьютер.

На многих наиболее популярных веб-сайтах рабочие базы данных хранятся на мэйнфреймах. Новые аппаратные и программные продукты для мэйнфреймов иде-

ально подходят для веб-транзакций, так как они разработаны таким образом, чтобы обеспечивать быстрый и одновременный доступ огромного количества пользователей и приложений к одним и тем же данным без помехи друг для друга. Такой уровень безопасности, масштабируемости и надежности критически важен для эффективного и защищенного выполнения обработки информации в настоящее время.

Корпорации используют мэйнфреймы для выполнения приложений, требующих масштабируемости и надежности. Например, банковское учреждение может использовать мэйнфрейм для размещения базы данных счетов своих клиентов, для которых можно осуществлять транзакции с любого из тысяч банкоматов по всему миру.

Современные предприятия используют мэйнфреймы в следующих целях:

- выполнение обработки большого объема транзакций (тысячи транзакций в секунду)[1];
- поддержка тысяч пользователей и приложений, осуществляющих одновременный доступ ко множеству ресурсов;
- управление терабайтами информации в базах данных;
- поддержка связи с высокой пропускной способностью.

Информационные магистрали часто ведут к мэйнфрейму.

1.7 Факторы, способствующие использованию мэйнфреймов

Существует множество причин для использования мэйнфреймов, но большинство из них относятся к одной или нескольким из следующих категорий:

- надежность, доступность и удобство обслуживания;
- безопасность;
- масштабируемость;
- последовательная совместимость;
- эволюционирующая архитектура;

Рассмотрим эти категории более подробно.

1.7.1 Надежность, доступность и удобство обслуживания

Надежность, *доступность* и *удобство обслуживания* (*reliability, availability, serviceability* – RAS) компьютерной системы всегда были важными факторами в обработке данных. Когда говорится, что та или иная компьютерная система «имеет характеристики RAS», имеется в виду, что она разработана таким образом, чтобы обеспечивать непрерывную работу в любое время. В идеале RAS является основным требованием при разработке всех аспектов компьютерной системы, включая приложения.

В настоящее время термин RAS используется в качестве собирательного понятия, определяющего множество характеристик аппаратного и программного обеспече-

[1] Мейнфрейм-компьютеры IBM последней серии. IBM System z9 109 (также известные под названием z9-109) могут обрабатывать один миллиард транзакций в день.

ния, важных для пользователей мэйнфреймов. Этим терминам можно дать следующие определения:

Надежность	Аппаратные компоненты системы имеют широкие возможности самопроверки и самовосстановления. Надежность программного обеспечения системы является результатом всестороннего тестирования и возможности быстрой установки обновлений для устранения выявленных проблем.
Доступность	Система может выполнить восстановление после отказа компонента без воздействия на остальную часть работающей системы. Это относится как к восстановлению оборудования (автоматическая замена отказавших элементов запасными), так и к восстановлению программного обеспечения (операционная система обеспечивает несколько уровней восстановления после ошибок).
Удобство обслуживания	Система может определить, почему возник отказ. Это позволяет выполнить замену аппаратных или программных элементов при минимальном воздействии на операционную систему. Этот термин также предполагает использование четко определенных единиц замены оборудования или программного обеспечения.

Доступность – способность восстановления после отказа компонента без воздействия на остальную часть работающей системы

Компьютерная система считается доступной тогда, когда доступны ее приложения. Доступной является надежная система; другими словами, она редко требует отключения для обновления или ремонта. И если система была остановлена вследствие возникновения ошибки, она должна быть удобной в обслуживании, т. е. легкой в ремонте за сравнительно короткий промежуток времени.

Показатель средней наработки на отказ (mean time between failure, MTBF) характеризует доступность компьютерной системы. Современные мэйнфреймы и используемое на них программное обеспечение к настоящему времени дошли до такого уровня, что клиенты часто могут работать месяцами и даже *годами* между простоями системы. Кроме того, когда система становится недоступной вследствие неожиданного отказа или запланированного обновления, период простоя обычно является очень коротким. Высокая доступность системы при обработке критически важных приложений организации необходима при современной глобальной экономике, работающей 24 часа в сутки. Не только оборудование, но и операционные системы для мэйнфреймов демонстрируют характеристики RAS в таких сферах, как защита систем хранения и процесс управляемого обслуживания.

Помимо соответствия характеристикам RAS, о современных мэйнфрейм-системах можно сказать, что они обеспечивают *высокую доступность* и *отказоустойчивость*. Использование резервных аппаратных компонентов на критически важных узлах, усовершенствованная защита систем хранения, процесс управляемого обслуживания и системное программное обеспечение, разработанное для обеспечения неограниченной доступности, позволяют сохранить согласованность и высокую доступность среды для бизнес-приложений в случае отказа системных компонентов. Такой подход позволяет проектировщику системы предельно сократить риск появления *единой точкой отказа*, снижающей показатели RAS компьютерной системы.

1.7.2 Безопасность

Одним из наиболее ценных ресурсов компании являются ее данные: списки клиентов, бухгалтерские данные, информация о сотрудниках и т. д. Эти критически важные данные подлежат безопасному управлению и контролю и вместе с тем должны быть доступны пользователям, авторизованным для их просмотра. Мэйнфрейм-компьютер имеет широкие возможности одновременного совместного использования данных компании множеством пользователей с обеспечением их защиты.

В отрасли информационных технологий безопасность данных определяется как их защита от несанкционированного доступа, передачи, изменения или нарушения, как случайного, так и преднамеренного. Для защиты данных и обеспечения ресурсов, необходимых для достижения целей безопасности, клиенты обычно устанавливают полнофункциональную программу управления безопасностью в операционную систему мэйнфрейма. Общая ответственность за использование доступных технологий для преобразования политики безопасности компании в практический план часто возлагается на администратора безопасности.

Защищенная компьютерная система не позволяет пользователям осуществлять доступ или изменять какие-либо объекты в системе, включая пользовательские данные, за исключением случаев употребления интерфейсов системы, применяющих правила полномочий. Современные мэйнфреймы обеспечивают очень надежную систему для выполнения большого числа разнообразных приложений, осуществляющих доступ к критическим данным. Пример системы безопасности мэйнфрейма рассматривается в главе 18, «Безопасность в z/OS».

1.7.3 Масштабируемость

Как говорится, единственное, что остается постоянным, – это *перемены*. Это тем более истинно в отрасли информационных технологий. В бизнесе положительные результаты могут вызвать рост информационной инфраструктуры в целях обеспечения соответствия растущему спросу. Уровень, до которого ИТ-организация может нарастить мощности без нарушения нормальных бизнес-процессов или без возникновения чрезмерных накладных расходов (непроизводительной обработки), в значительной степени определяется масштабируемостью заданной вычислительной платформы.

Масштабируемость – способность системы сохранить уровень производительности при добавлении процессоров, памяти и устройств хранения

Под масштабируемостью понимается способность оборудования, программного обеспечения или распределенной системы продолжать эффективное функционирование при изменении размера или объема; например, способность сохранить уровень производительности при добавлении процессоров, памяти и устройств хранения. Масштабируемая система может эффективно адаптироваться к работе, где сети большего или меньшего размера выполняют задачи различной сложности.

По мере увеличения количества сотрудников, клиентов и бизнес-партнеров, компании обычно необходимо добавлять вычислительные ресурсы для поддержки роста предприятия. Один подход состоит в том, чтобы добавить больше процессоров оди-

наковой мощности, что увеличивает нагрузку по управлению более сложной системой. С другой стороны, компания может консолидировать множество менее мощных процессоров в небольшое количество более крупных систем. Используя мэйнфрейм-системы, многие компании значительно сократили совокупную стоимость владения (total cost of ownership, TCO), которая учитывает не только стоимость системы (ее аппаратного и программного обеспечения), но и стоимость ее эксплуатации.

Мэйнфреймы демонстрируют высокую масштабируемость аппаратного и программного обеспечения и способны выполнять несколько копий программного обеспечения операционной системы как единую систему, что называется системным комплексом – *сисплексом (sysplex)*. Более подробно технология кластеризации мэйнфреймов и ее использование рассматривается в разделе 2.8, «Что такое Parallel Sysplex?».

1.7.4 Последовательная совместимость

Покупатели мэйнфреймов склонны производить очень большие финансовые вложения в свои приложения и данные. Некоторые приложения разрабатывались и дорабатывались десятилетиями. Одни приложения были написаны много лет назад, тогда как другие могли быть написаны буквально на днях. Способность приложения работать в системе или его способность работать с другими устройствами или программами называется *совместимостью (compatibility)*.

Совместимость – способность системы выполнять как программное обеспечение, использующее новые аппаратные инструкции, так и более старое программное обеспечение, использующее первоначальные аппаратные инструкции

Необходимость поддерживать приложения разного времени выпуска создает строгое требование совместимости для аппаратного и программного обеспечения мэйнфрейма, которое многократно обновлялось с момента появления первого мэйнфрейм-компьютера System/360 в 1964 году. Приложения *должны* продолжать работать должным образом. Итак, основная работа над проектированием нового оборудования и системного программного обеспечения вращается вокруг требований совместимости.

Исключительная необходимость обеспечения совместимости является также основной причиной, определяющей работу многих аспектов системы, например ограничения синтаксиса языка управления заданиями (job control language, JCL), используемого для управления пакетными заданиями. Все новые усовершенствования JCL должны обеспечивать совместимость с более старыми заданиями, чтобы их можно было продолжать выполнять без внесения изменений. Желание и необходимость последовательной совместимости является одним из определяющих свойств мэйнфрейм-вычислений.

Абсолютной совместимости за десятилетия изменений и усовершенствований достичь, конечно же, невозможно, но для проектировщиков оборудования и программного обеспечения она является главным приоритетом. Если обеспечение совместимости невозможно, проектировщики *как минимум за год* предупреждают пользователей о необходимости внесения изменений в программное обеспечение.

1.8 Стандартные задачи для мэйнфреймов

Большинство задач для мэйнфреймов относятся к одной из двух категорий: пакетная обработка или обработка оперативных транзакций, куда относятся и веб-приложения (рис. 1.1).

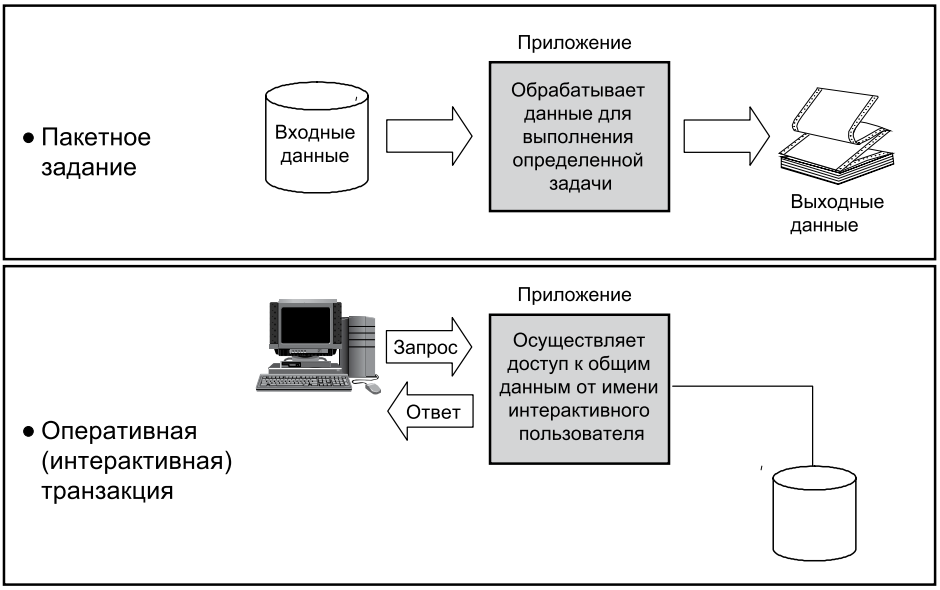


Рис. 1.1. Стандартные задачи для мэйнфреймов

Эти задачи рассматриваются в нескольких главах этой книги; в следующих разделах представлены общие сведения.

1.8.1 Пакетная обработка

Одним из основных преимуществ мэйнфрейм-систем является их способность обрабатывать терабайты данных, размещенных на высокоскоростных устройствах хранения, и производить ценные выходные данные. Например, использование мэйнфрейм-систем позволяет банкам и прочим финансовым учреждениям выполнять квартальную обработку и создавать отчеты, необходимые для клиентов (например, квартальные отчеты о запасах или пенсионные отчеты) или правительства (например, финансовые результаты). Используя мэйнфрейм-системы, магазины могут генерировать и консолидировать суточные отчеты о продажах для региональных менеджеров по продажам.

Пакетная обработка – выполнение заданий на мейнфрейме без вмешательства пользователя

Приложения, генерирующие эти отчеты, называются *пакетными* приложениями; другими словами, они выполняются на мэйнфрейме без вмешательства пользователя. *Пакетное задание* запускается на компьюте-

ре, считывает и обрабатывает данные в больших количествах (возможно, терабайты данных) и производит выходные данные, в частности клиентские счета. Это подобно выполнению файла скрипта в UNIX или командного файла в Windows®, однако пакетное задание z/OS может обрабатывать миллионы записей.

Несмотря на то что пакетная обработка возможна и на распределенных системах, она не настолько распространена, как на мэйнфреймах, так как распределенным системам часто не хватает:

- достаточного дискового пространства;
- доступной мощности процессора или *циклов;*
- управления системными ресурсами и планированием заданий на уровне сисплекса.

Операционные системы мэйнфреймов обычно содержат полнофункциональное программное обеспечение для планирования заданий, которое позволяет персоналу информационного центра осуществлять передачу, управление и слежение за выполнением и выводом пакетных заданий[2].

Пакетные процессы обычно имеют следующие свойства:

- Выполняется обработка и хранение больших объемов вводимых данных (возможно, терабайтов или еще больше), осуществляется доступ к большому количеству записей и производится большой объем выходных данных.
- Мгновенное реагирование обычно не является обязательным требованием. Однако часто выделяется определенный «период выполнения пакетных заданий», т. е. период менее интенсивной активности, заданный в *соглашении об уровне обслуживания (service level agreement, SLA).*
- Генерируется информация о большом количестве пользователей или информационных объектов (например, заказах клиентов или товаре на складе).
- Запланированный пакетный процесс может представлять выполнение сотен или тысяч заданий в предварительно заданной последовательности.

Во время пакетной обработки возможно генерирование различных типов задач. Часто используется консолидированная информация, например о доходности инвестиционных фондов, запланированном резервном копировании баз данных, обработке ежедневных заказов и пополнении склада. На рис. 1.2 показано несколько пакетных заданий, выполняемых в стандартной среде мэйнфрейма.

На рис. 1.2 представлены следующие элементы, выполняемые в запланированном пакетном процессе:

1. Ночью происходит обработка множества пакетных заданий, осуществляющих запуск программ и утилит. Эти задания консолидируют результаты оперативных транзакций, произошедших в течение дня.
2. Пакетные задания генерируют отчеты с коммерческой статистикой.

---

[1] На заре эпохи мэйнфреймов для ввода заданий, выполняемых системой, часто использовались перфокарты. Операторы вводили данные с использованием клавишных перфораторов и производили стопки (или пакеты) перфокарт. Их вставляли в устройства чтения перфокарт, которые считывали задания и данные в систему. Как и следовало ожидать, этот процесс был очень обременительным и ненадежным. В настоящее время вместо перфокарт можно использовать текстовый файл. Различные способы ввода заданий в мэйнфрейм рассматриваются в главе 7, «Пакетная обработка и JES».

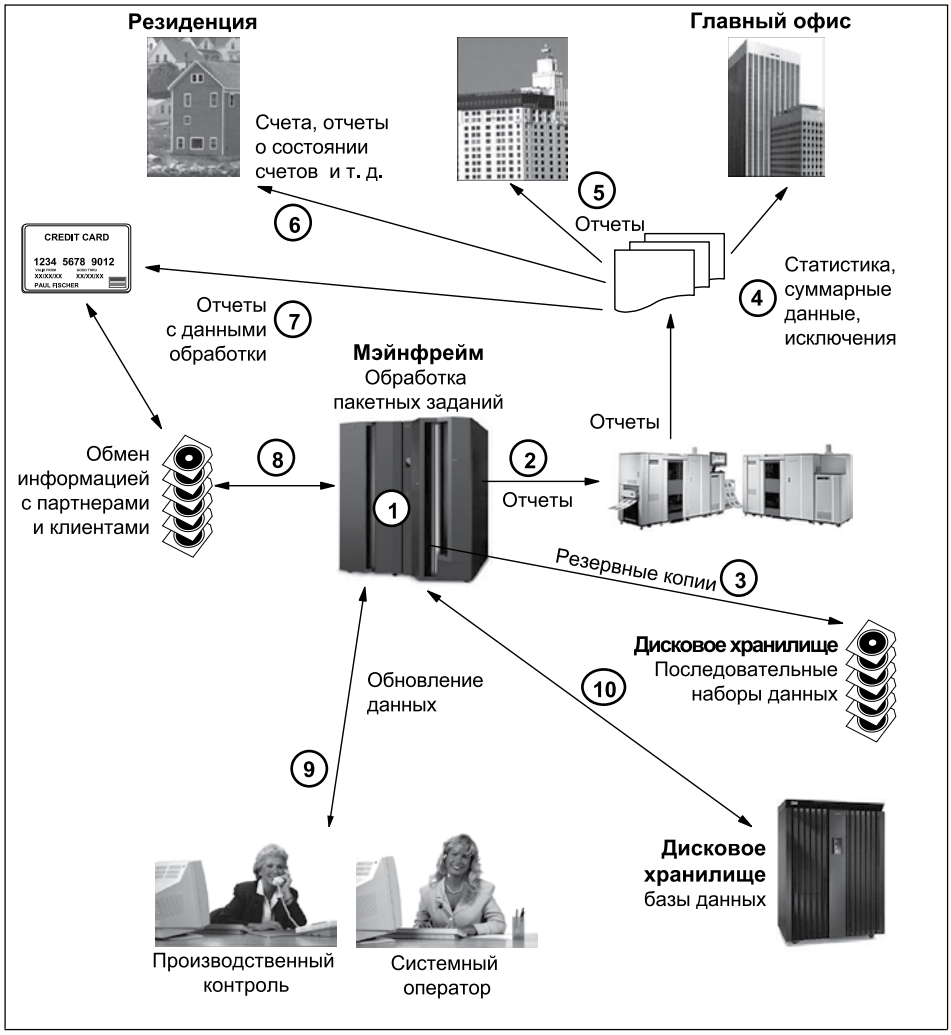


Рис. 1.2. Стандартные примеры использования пакетных заданий

3. До и после периода выполнения пакетных заданий создаются резервные копии критических файлов и баз данных.
4. Отчеты с коммерческой статистикой пересылаются в специальный отдел для анализа на следующий день.
5. Отчеты с исключениями пересылаются в филиалы.
6. Ежемесячные отчеты о состоянии счетов генерируются и рассылаются всем клиентам банка.
7. Отчеты с суммарными данными обработки отправляются партнерской компании-владельцу торговой марки кредитных карт.
8. Выполняется получение отчета о транзакциях по кредитным картам от партнерской компании.

9. В отделе производственного контроля осуществляется мониторинг сообщений на системной консоли и выполнение заданий.
10. Задания и транзакции выполняют чтение или обновление базы данных (той же, которая используется оперативными транзакциями), и множество файлов записывается на магнитную ленту.

1.8.2 Обработка оперативных транзакций

Обработка транзакций, выполняемая интерактивно с участием конечного пользователя, называется *обработкой оперативных транзакций (online transaction processing, OLTP)*. Обычно мэйнфрейм обслуживает огромное количество *систем выполнения транзакций (transaction systems)*. Эти системы часто представляют собой критически важные приложения, от которых зависит выполнение основных функций предприятия. Системы выполнения транзакций должны быть способны поддерживать непредсказуемое количество одновременно работающих пользователей и типов транзакций. Большинство транзакций выполняется за кратчайшие промежутки времени, в некоторых случаях за доли секунды.

Обработка оперативных транзакций (OLTP) – обработка транзакций, выполняемая интерактивно с участием конечного пользователя

Одно из основных свойств системы выполнения транзакций состоит в том, что взаимодействие между пользователем и системой происходит очень быстро. Бизнес-транзакция целиком состоит из серии коротких взаимодействий, при которых для каждого взаимодействия требуется немедленное реагирование. В настоящее время такие системы поддерживают критически важные приложения; поэтому непрерывная доступность, высокая производительность, а также защита и обеспечение целостности данных являются обязательным требованием.

Оперативные транзакции знакомы большинству людей. Приведем следующие примеры:

- транзакции при работе с банкоматом, в частности депонирование, снятие денег, проверка состояния счета и перевод денег;
- оплата в супермаркете дебетовой или кредитной картой;
- покупка товаров через Интернет.

Например, в банковском филиале или в Интернете клиенты могут использовать оперативные службы при проверке состояния счета или при работе с денежными средствами.

В действительности оперативная система во многом выполняет те же функции, что и операционная система:

- управление и распределение задач;
- управление полномочиями доступа пользователей к системным ресурсам;
- управление использованием памяти;
- управление и контроль над одновременным доступом к файлам данных;
- обеспечение независимости устройств.

В некоторых отраслях оперативные системы на основе мэйнфреймов используются следующим образом:

- банки – банкоматы, банковские системы для обслуживания клиентов;
- страхование – агентские системы для управления политиками и обработки страховых требований;
- туризм и транспорт – системы бронирования авиабилетов;
- промышленность – управление запасами, производственное планирование;
- правительство – налоговая обработка, выдача и управление лицензиями.

Каким образом конечные пользователи в этих отраслях могут взаимодействовать с мэйнфрейм-системами? На дизайн системы обработки транзакций компании может влиять множество факторов, в том числе:

- Количество пользователей, взаимодействующее с системой в любой момент времени.
- Количество транзакций в секунду (TPS).
- Требования доступности приложения. Например, должно ли приложение быть доступно 24 часа в сутки, семь дней в неделю, или же его раз в неделю можно на короткое время отключать?

До того как стали популярными персональные компьютеры и интеллектуальные рабочие станции, самым распространенным способом связи с оперативными мэйнфрейм-приложениями были терминалы 3270. Эти устройства иногда называли «немыми» терминалами, хотя они могли осуществлять сбор и отображение полного экрана данных, вместо того чтобы взаимодействовать с компьютером при каждом нажатии клавиши, что позволяло сократить использование процессора. Символы на черном экране имели зеленый цвет, поэтому мэйнфрейм-приложения часто называли «зеленым экраном».

Эти факторы вызывают различия при взаимодействии с пользователями в различных инсталляциях. В настоящее время во многих инсталляциях выполняется переработка существующих мэйнфрейм-приложений с целью реализовать пользовательский интерфейс на основе веб-браузера. Иногда это требует разработки нового приложения, однако часто бывает достаточно приобрести программное обеспечения стороннего разработчика для добавления новых интерфейсов с приложением. В этом случае конечный пользователь часто не осознает, что он имеет дело с мэйнфреймом.

В этой книге нет необходимости описывать процесс взаимодействия с мэйнфреймом через веб-браузер, так как он ничем не отличается от всех остальных взаимодействий, осуществляемых пользователем через Интернет. Единственное различие состоит в том, какой компьютер на другом конце.

Оперативные транзакции обычно имеют следующие свойства:

- небольшое количество входных данных, небольшое количество хранимых записей, к которым осуществляется доступ и обработка, и небольшое количество выходных данных;
- мгновенное реагирование, обычно время реагирования составляет меньше секунды;

- большое количество пользователей, задействованных в большом количестве транзакций;
- круглосуточная доступность транзакционного интерфейса для пользователя;
- обеспечение безопасности транзакций и пользовательских данных.

В банковском филиале, например, клиенты используют оперативные службы при проверке состояния счета или внесении денег.

На рис. 1.3 представлен набор стандартных оперативных транзакций, использующих мэйнфрейм.

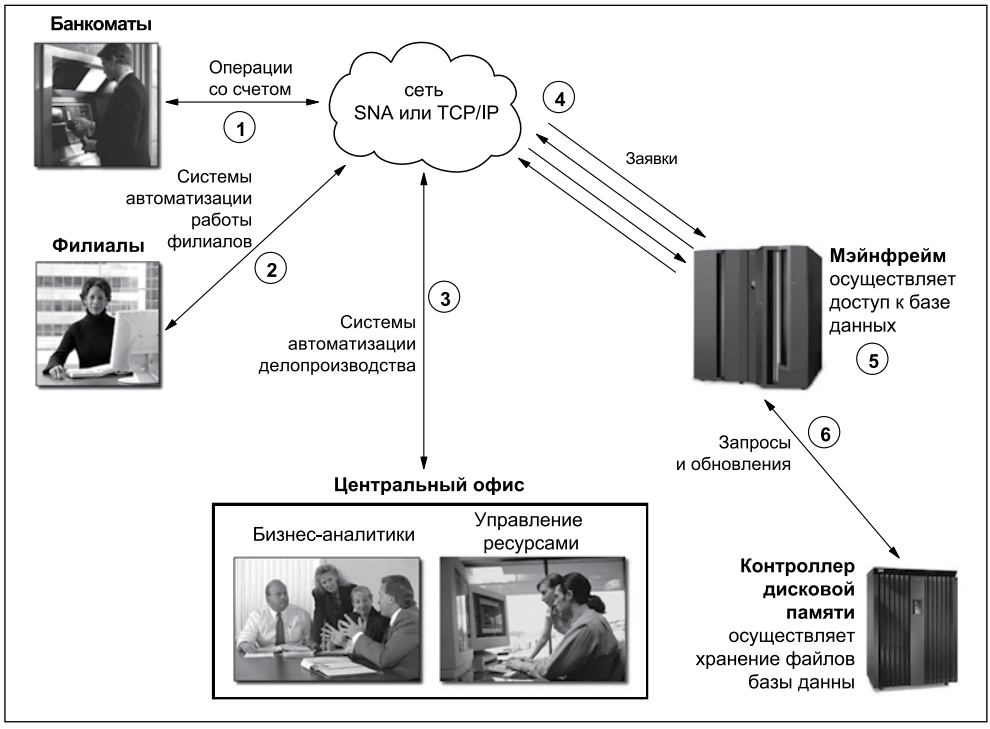


Рис. 1.3. Стандартные оперативные транзакции

1. Клиент использует банкомат, который имеет понятный пользовательский интерфейс для выполнения различных функций: снятия денег, запроса состояния счета, депонирования, перевода или снятия средств с кредитной карты.
2. В другом месте той же частной сети банковский сотрудник в филиале выполняет такие операции, как консультирование, распределение денежных средств и работа с денежными переводами.
3. В центральном офисе банка бизнес-аналитики настраивают транзакции для улучшения их производительности. Другие сотрудники используют специализированные оперативные системы автоматизации делопроизводства для осуществления управления взаимодействием между клиентами, бюджетного планирования и контроля ресурсов.

4. Все запросы направляются для обработки на мэйнфрейм-компьютер.
5. Программы, запущенные на мэйнфрейм-компьютере, выполняют обновления и запросы к системе управления базой данных (например, DB2).
6. Специализированные системы дисковой памяти осуществляют хранение файлов базы данных.

1.9 Роли в мире мэйнфреймов

Мэйнфрейм-системы предназначены для использования большим количеством людей. Большинство людей, взаимодействующих с мэйнфреймами, являются конечными пользователями – людьми, применяющими приложения, размещенные на системе. Однако по причине большого количества конечных пользователей, приложений, выполняющихся на системе, а также многофункциональности и сложности системного программного обеспечения, поддерживающего пользователей и приложения, для функционирования и поддержки системы необходимо множество ролей.

В сфере информационных технологий этим ролям соответствует множество различных названий должностей; в данной книге мы используем следующие:

- системные программисты;
- системные администраторы;
- проектировщики и программисты приложений;
- системные операторы;
- аналитики производственного контроля.

В среде распределенных систем используются многие из тех же ролей, которые используются в среде мэйнфреймов. Однако должностные обязанности часто не настолько четко определены. С 1960-х годов роли в области мэйнфреймов развивались и расширялись, обеспечивая среду, в которой системное программное обеспечение и приложения могут беспрепятственно и эффективно функционировать, обслуживая тысячи пользователей. Может показаться, что количество сотрудников, занимающихся поддержкой мэйнфрейма, слишком велико, однако эти числа становятся сравнительно небольшими, если учитывать количество поддерживаемых пользователей, количество выполняемых транзакций и высокую ценность работы, выполняемой на мэйнфрейме.

Эта книга сосредоточена главным образом на ролях системного программиста и программиста приложений в среде мэйнфреймов. Однако существует и несколько других важных должностей, занимающихся «уходом» за мэйнфреймом, и мы затронем некоторые из этих ролей, чтобы дать вам понимание о том, как все работает.

При работе с мэйнфреймами часто необходимо взаимодействие между различными ролями, в частности при выполнении следующих операций:

- установка и конфигурирование системного программного обеспечения;
- разработка и написание новых приложений для выполнения на мэйнфрейме;
- запуск новых задач в системе, в частности пакетных заданий и обработки оперативных транзакций;

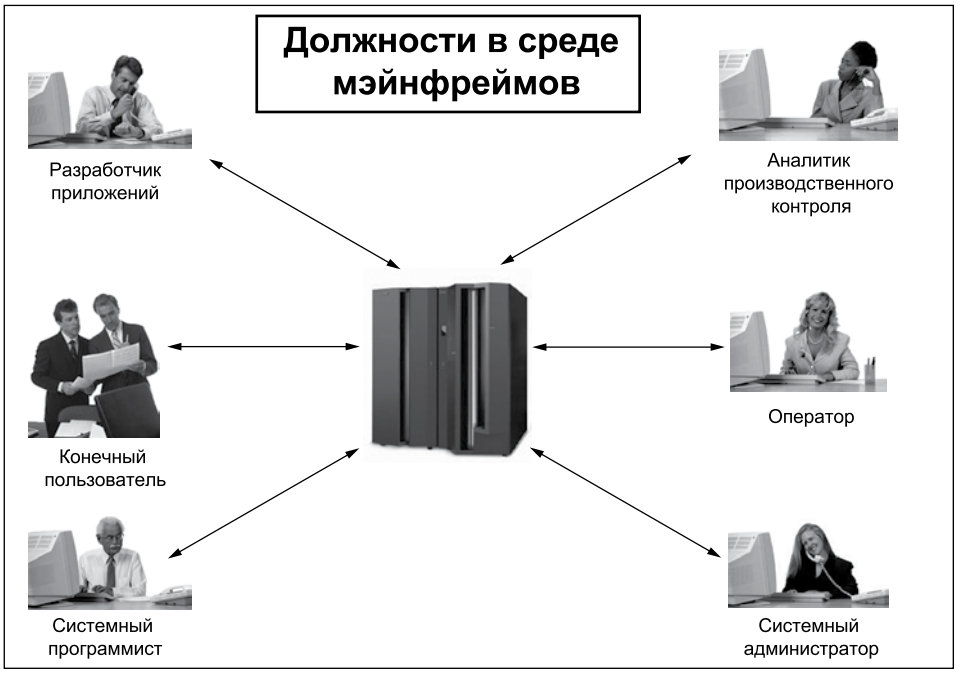


Рис. 1.4. Кто есть кто в мире мейнфреймов

- использование и обслуживание программного и аппаратного обеспечения мэйнфрейма.

В последующих разделах мы опишем каждую роль более подробно.

1.9.1 Кто такой системный программист?

В ИТ-организации, использующей мэйнфрейм, системный программист играет ключевую роль. Системный программист устанавливает, настраивает и обслуживает операционную систему, а также устанавливает и обновляет продукты, выполняемые в системе. Системный программист может заниматься обновлением существующих систем до последней версии операционной системы либо выполнять такие простые задачи, как обновление одной программы, например приложения сортировки.

Системный программист – сотрудник, занимающийся установкой, настройкой и обслуживанием операционной системы

Системный программист, в частности, выполняет следующие задачи:

- планирование обновлений аппаратного и программного обеспечения и изменений в конфигурации;
- подготовка системных операторов и программистов приложений;
- автоматизация операций;
- планирование вычислительной мощности;
- запуск заданий и скриптов установки;

- выполнение заданий настройки, относящихся к инсталляции;
- тестирование взаимодействия новых продуктов с существующими приложениями и пользовательскими процедурами;
- настройка производительности в масштабах системы в целях обеспечения требуемого уровня обслуживания.

Системный программист должен иметь навыки в исправлении проблем в системном программном обеспечении. Эти проблемы часто фиксируются в копии содержимого памяти компьютера, называемой *дампом*, которую создает система при отказе программного продукта, пользовательского задания или транзакции. Используя дамп и специализированные инструменты отладки, системный программист может определить отказавшие компоненты. При возникновении ошибки в программном продукте системный программист работает непосредственно с представителями службы поддержки изготовителя программного обеспечения, чтобы определить, известна ли причина проблемы и доступно ли исправление.

Системные программисты также занимаются установкой и обслуживанием *промежуточного программного обеспечения (middleware)* на мэйнфрейме, в частности систем управления базами данных, систем оперативной обработки транзакций и вебсерверов. Промежуточное программное обеспечение представляет собой программный «слой» между операционной системой и конечным пользователем или приложением конечного пользователя. Оно содержит важные функции, не реализованные в операционной системе. Основные промежуточные программные продукты, такие, как DB2, CICS и IMS™, могут быть настолько же сложными, как и сама операционная система, если даже не сложнее.

1.9.2 Кто такой системный администратор?

Различия между системным программистом и системным администраторам колеблются в разных организациях, использующие мэйнфреймы. В небольших ИТ-организациях, где один человек может выполнять несколько ролей, эти термины могут использоваться как взаимозаменяемые.

Системный администратор – сотрудник, осуществляющий обслуживание критически важных данных предприятия, расположенных на мейнфрейме

Более крупные ИТ-организации со множеством отделов склонны более четко разделять должностные обязанности. Системные администраторы выполняют большую часть повседневных задач, связанных с обслуживанием критически важных данных предприятия, расположенных на мэйнфрейме, тогда как системный программист сосредоточен на обслуживании самой системы. Одной из причин разделения обязанностей является необходимость соответствия процедурам аудита, которые часто требуют, чтобы ни один человек в ИТ-организации не имел неограниченный доступ к конфиденциальным данным или ресурсам. В частности, к системным администраторам относятся администратор базы данных и администратор безопасности.

Если к компетенции системного программиста относится главным образом аппаратное и программное обеспечение, то системные администраторы больше занима-

ются работой с приложениями. Они часто взаимодействуют непосредственно с программистами приложений и конечными пользователями, обеспечивая соответствие аспектам администрирования приложений. Эти роли необязательно должны быть уникальными в среде мэйнфрейма, но они тем не менее крайне важны для его бесперебойной работы.

В крупных ИТ-организациях системный администратор осуществляет поддержку среды системного программного обеспечения для достижения целей предприятия, включая повседневное обслуживание систем для обеспечения их бесперебойной работы. Например, администратор базы данных должен обеспечивать целостность и эффективный доступ к данным, хранящимся в системах управления базами данных.

К другим примерам основных задач системного администратора относятся:

- установка программного обеспечения;
- добавление и удаление пользователей и управление профилями пользователей;
- управление списками безопасного доступа к ресурсам;
- управление устройствами хранения и принтерами;
- управление сетями и связью;
- мониторинг системной производительности.

В вопросах определения проблем системный администратор обычно взаимодействует с персоналом центра поддержки изготовителя программного обеспечения для диагностики проблем, чтения дампов и выявления исправлений в тех случаях, когда эти задачи не выполняются системным программистом.

1.9.3 Кто такие проектировщики и программисты приложений?

Проектировщик приложений и программист приложений (или разработчик приложений) осуществляют проектирование, компоновку, тестирование и доставку мэйнфрейм-приложений конечным пользователям и клиентам компании. На основании требований, полученных от бизнес-аналитиков и конечных пользователей, проектировщик создает техническое задание, на основании которого программист создает приложение. Этот процесс включает несколько итераций из изменений кода и компиляций, компоновки приложения и модульного тестирования.

В процессе разработки приложения проектировщик и программист должны взаимодействовать с другими ролями на предприятии. Например, программист часто работает в команде вместе с другими программистами, разрабатывающими код для связанных модулей приложения. После разработки каждый модуль проходит процесс тестирования, который может включать функциональные, интеграционные и системные тесты. После выполнения тестов приложения должны пройти приемочное тестирование сообществом пользователей, чтобы определить, соответствует ли код поставленным требованиям.

Помимо создания кода новых приложений, программист отвечает за обслуживание и доработку существующих мэйнфрейм-приложений компании. В действительности это часто является основной задачей для многих современных программистов мэйнфрейм-приложений. Несмотря на то что для создания новых программ для

мэйнфреймов все еще используется COBOL (Common Business Oriented Language) и PL/I, такие языки, как Java™, набирают популярность точно так же, как и на распределенных платформах.

Повсеместная разработка мэйнфрейм-программ на высокоуровневых языках, таких, как COBOL и PL/I, идет быстрым темпом, несмотря на слухи о противоположном. Тысячи программ работают на мэйнфрейм-системах по всему миру, и эти программы являются критически важными для повседневной работы корпораций, которые их используют. Программисты, знающие COBOL и другие высокоуровневые языки, нужны для поддержки существующего кода и создания обновлений и изменений для существующих программ. Кроме того, многие корпорации продолжают создавать новые приложения на языке COBOL и других традиционных языках, и компания IBM продолжает совершенствовать свои компиляторы высокоуровневых языков, включая в них новые функции и возможности, позволяющие этим языкам использовать новые технологии и форматы данных.

Более подробно роли проектировщика приложений и программиста приложений мы рассмотрим во второй части этой книги.

1.9.4 Кто такой системный оператор?

Системный оператор осуществляет мониторинг и контроль операций, выполняемых на аппаратном и программном обеспечении мэйнфрейма. Оператор запускает и останавливает системные задачи, проверяет системные консоли на возникновение необычных состояний и работает с системными программистами и специалистами по производственному контролю, обеспечивая исправность и нормальную работу систем.

Системный оператор – сотрудник, осуществляющий мониторинг и контроль операций, выполняемых на аппаратном и программном обеспечении мейнфрейма

При добавлении приложений в мэйнфрейм системный оператор также отвечает за обеспечение их бесперебойной работы. Новые приложения из отдела программирования приложений обычно доставляются операционному персоналу вместе с *документацией,* содержащей инструкции. Документация описывает операционные требования приложения, о которых операторы должны знать при выполнении задачи. Инструкции в документации могут включать, например: консольные сообщения приложения, требующие вмешательства оператора; рекомендуемые действия оператора при определенных системных событиях и указания по изменению потоков заданий для приспособления к изменениям требований предприятия[1].

Оператор также отвечает за запуск и остановку основных подсистем, таких, как системы обработки транзакций, системы управления базами данных и самой операционной системы. В настоящее время *операции перезапуска* выполняются намного реже, чем в прошлом, так как доступность мэйнфрейма за последние годы значительно возросла. Однако оператор все же должен уметь должным образом выполнять завершение работы и запуск системы и ее задач при необходимости.

[1] Когда-то консольные сообщения были настолько объемными, что операторам часто было сложно определить, является ли возникшая ситуация проблемой. В последние годы появились инструменты, сократившие размер сообщений и автоматизирующие ответы на стандартные ситуации, которые значительно упростили работу операторов, позволяя им сосредоточиваться только на нестандартных ситуациях, требующих вмешательства человека.

В случае отказа или возникновения нештатной ситуации оператор связывается с системными программистами, которые помогают ему выбрать правильный образ действий, а также с аналитиком производственного контроля, который работает совместно с оператором, чтобы убедиться в правильности выполнения рабочих задач.

1.9.5 Кто такой аналитик производственного контроля?

Аналитик производственного контроля отвечает за проверку безошибочного и бесперебойного выполнения пакетных заданий. На некоторых мэйнфреймах сначала выполняются интерактивные задачи для оперативных пользователей, а после основной смены, когда оперативные системы не работают, запускаются пакетные обновления. Хотя эта модель работы все еще распространена, для многих компаний, осуществляющих операции во всемирном масштабе, требующие постоянного доступа к рабочим данным через Интернет, модель «оперативный режим днем/пакетный режим ночью» является устаревшей. Тем не менее пакетные задачи продолжают быть частью информационной обработки, и опытные аналитики производственного контроля играют важную роль.

Аналитик производственного контроля – сотрудник, обеспечивающий безошибочное и бесперебойное выполнение пакетных заданий

Мэйнфрейм-системы часто обвиняют в том, что они негибки и сложны в работе, особенно если требуется внести изменения. Аналитики производственного контроля часто слышат такие жалобы, однако понимают, что использование хорошо структурированных правил и процедур для контроля изменений, что является достоинством мэйнфрейм-среды, позволяет избежать простоев. В действительности одна из причин приобретения мэйнфреймами устойчивой репутации высокодоступных и высокопроизводительных систем заключается в том, что предусмотрен контроль изменений и сложно внести изменение в обход требуемых процедур.

1.9.6 Какую роль играют изготовители?

Некоторые роли компаний-изготовителей используются и в организациях, использующих мэйнфреймы. Так как большая часть мэйнфрейм-компьютеров производится компанией IBM, и операционные системы и основные оперативные системы также предоставляются компанией IBM, большинство контактных лиц изготовителей являются сотрудниками компании IBM. Тем не менее продукты независимых изготовителей программного обеспечения также используются в среде мэйнфреймов IBM и, кроме того, клиенты часто используют оборудование OEM-производителей, в частности устройства хранения на дисках и магнитнтой ленте. Ниже перечислены стандартные роли изготовителей.

- *Специалист по поддержке оборудования или наладчик оборудования.* Изготовители оборудования обычно предоставляют поддержку аппаратных устройств «на месте». В IBM специалисты по обслуживанию оборудования часто называются наладчиками оборудования (customer engineer, CE). Наладчик выполняет установку и ремонт оборудования и периферийных устройств мэйнфрейма. При отказе оборудования или установке нового оборудования наладчики обычно работают непосредственно в связке с операционными командами.

- *Специалист по поддержке программного обеспечения.*
  Существует множество ролей компании-изготовителя, предназначенных для поддержки программных продуктов на мэйнфрейме[1]. IBM имеет централизованный центр поддержки (Support Center), предоставляющий гарантийную и дополнительную поддержку по устранению дефектов программного обеспечения, а также справочную поддержку при использовании. Кроме того, для предоставления дополнительной предпродажной и послепродажной поддержки программных продуктов в зависимости от размера предприятия и конкретной ситуации могут быть задействованы специалисты по информационным технологиям и разработчики.
- *Полевой специалист по технической поддержке сбыта, системный инженер или представитель клиента.*
  Для крупных покупателей мэйнфреймов компания IBM и другие изготовители обеспечивают личную поддержку сбыта. Представители изготовителя специализируются в различных типах оборудования и семействах программных продуктов и посещают отдел клиентской организации, отвечающий за приобретение продуктов. В IBM специалист по технической поддержке сбыта называется полевым специалистом по технической поддержке сбыта (field technical sales support, FTSS) или более старым термином «системный инженер» (systems engineer, SE).
  Для крупных покупателей мэйнфреймов компания IBM часто выделяет сотрудника, называемого представителем клиента, знакомого с подробностями бизнеса в определенном секторе рынка и работающего исключительно с небольшим числом клиентов. Представитель клиента выступает в качестве «единой точки контакта» между клиентом и различными организациями в составе IBM.

1.10 z/OS и прочие операционные системы мэйнфреймов

Эта книга главным образом сосредоточена на изучении основ z/OS – передовой операционной системы для мэйнфреймов компании IBM. Рассмотрение основных понятий z/OS начинается в главе 3, «Общие сведения о z/OS». Тем не менее студентам, изучающим мэйнфреймы, полезно иметь практическое знание других операционных систем для мэйнфреймов. Одна из причин заключается в том, что на мэйнфрейм-компьютере может выполняться несколько операционных систем. Например, часто на одном мэйнфрейме одновременно используются системы z/OS, z/VM® и Linux®.

Операционные системы мэйнфреймов являются полнофункциональными продуктами, имеющими в значительной степени различающиеся свойства и цели, и подробное описание каждой из них заняло бы отдельную книгу. Помимо z/OS, еще четыре операционные системы доминируют на мэйнфреймах: z/VM, z/VSE™, Linux для zSeries и z/TPF.

[1] В этой книге не рассматриваются вопросы продажи и стоимости программного обеспечения мейнфрейма. Однако доступность и стоимость промежуточного программного обеспечения и прочих лицензионных программ является критическим фактором, влияющим на развитие и использование мейнфреймов.

1.10.1 z/VM

z/Virtual Machine (z/VM) содержит два основных компонента: *управляющую программу (control program, CP)* и однопользовательскую операционную систему (CMS). Работая в режиме управляющей программы, z/VM выступает в качестве *гипервизора (hypervisor)*, так как она запускает другие операционные системы в создаваемых виртуальных машинах. Любая из операционных систем для мэйнфреймов IBM, например z/OS, Linux для zSeries, z/VSE и z/TPF, может быть запущена как *гостевая система (guest system)* в отдельной виртуальной машине, и z/VM может запустить любое сочетание гостевых систем.

Управляющая программа искусственно создает несколько виртуальных машин на основе реальных аппаратных ресурсов. С точки зрения конечных пользователей все выглядит так, как будто они имеют возможность выделенного использования общих реальных ресурсов. К общим реальным ресурсам относятся принтеры, дисковые устройства хранения и процессор. Управляющая программа обеспечивает безопасность данных и приложений между гостевыми системами. Реальное оборудования может совместно использоваться гостевыми системами или быть выделено для использования одной гостевой системой в целях производительности. Системный программист устанавливает доступ гостевых систем к реальным устройствам. Для большинства клиентов использование гостевых систем позволяет избежать необходимости использования более крупных аппаратных конфигураций.

Еще одним важным компонентом z/VM является диалоговый монитор (Conversational Monitor System, CMS). Этот компонент z/VM запускается в виртуальной машине и обеспечивает как интерактивный интерфейс для конечного пользователя, так и общий интерфейс программирования приложений для z/VM.

1.10.2 z/VSE

z/Virtual Storage Extended (z/VSE) популярна среди пользователей небольших мэйнфрейм-компьютеров. Некоторая часть таких пользователей в конечном итоге переходят на z/OS, как только они перерастают возможности z/VSE.

В сравнении с z/OS операционная система z/VSE предоставляет небольшую и менее сложную основу для пакетной обработки и обработки транзакций. Дизайн и структура управления z/VSE идеально подходят для выполнения стандартных рабочих задач, включающих множество пакетных заданий (выполняющихся параллельно) и широкую традиционную обработку транзакций. На практике большинство пользователей z/VSE также применяют операционную систему z/VM в качестве основного терминального интерфейса для разработки приложений для z/VSE и управления системой.

z/VSE изначально имела название Disk Operating System (DOS) и была первой дисковой операционной системой для мэйнфрейм-компьютеров System/360. DOS планировалось использовать как временное решение до выхода OS/360. Однако некоторым клиентам мэйнфреймов так понравилась простота (и небольшой размер) этой системы, что они решили оставить ее после выпуска OS/360. Затем DOS сменила название на DOS/VS (когда она начала использовать виртуальную память), затем на VSE/SP, затем на VSE/ESA™ и, наконец, на z/VSE. Название VSE часто используется собирательно для обозначения любой из более поздних версий.

1.10.3 Linux для zSeries

На мэйнфрейме можно использовать несколько дистрибутивов Linux (выпущенные не IBM). Для обозначения этих дистрибутивов обычно используются следующие названия:

- Linux для S/390 (использует 31-разрядную адресацию и 32-разрядные регистры);
- Linux для zSeries (использует 64-разрядную адресацию и регистры).

Название *Linux on zSeries* используется для обозначения операционной системы Linux, выполняемой на системе S/390 или zSeries, когда нет необходимости явно указывать, о какой версии идет речь: 31-разрядной или 64-разрядной. Предполагается, что студенты имеют общее представление об операционной системе Linux, и поэтому мы опишем только те свойства, которые относятся к использованию на мэйнфреймах. К ним относятся следующие свойства:

- Linux использует традиционные дисковые устройства CKD (count-key-data)[1] и SCSI-устройства, подключенные к SAN. Другие операционные системы для мэйнфреймов могут распознавать эти устройства как приводы Linux, но не могут использовать форматы данных на этих приводах. Другими словами, это препятствует совместному использованию данных между Linux и другими операционными системами для мэйнфреймов.
- Linux не использует дисплейные терминалы 3270, тогда как все остальные операционные системы для мэйнфреймов используют терминалы 3270 в базовой терминальной архитектуре[2]. Linux использует терминалы на основе X-Window System или эмуляторы X-Window System на персональных компьютерах; он также поддерживает стандартные ASCII-терминалы, обычно связываемые посредством протокола *telnet*. X-Window System представляет собой стандартный графический интерфейс в Linux. Он является промежуточным уровнем между оборудованием и диспетчером окон.
- При надлежащей настройке систему Linux под управлением z/VM можно быстро «клонировать», создав отдельный образ Linux. Локальную сеть, эмулируемую в z/VM, можно использовать для соединения нескольких образов Linux и создания для них внешнего маршрута локальной сети. Файловые системы с доступом только для чтения, например стандартная файловая система /usr, могут совместно использоваться различными образами Linux.
- Linux на мэйнфрейме работает с набором символов ASCII, а не с EBCDIC[3], обычно используемым на мэйнфреймах. В системе EBCDIC используется только при выводе на такие символьные устройства, как дисплеи и принтеры. Преобразование символов для этих устройств осуществляются драйверами Linux.

---

1 CKD-устройства форматируются таким образом, чтобы считывающая готовка диска могла получить прямой доступ к отдельным фрагментам данных.

2 В Linux существует драйвер для минимальных операций с 3270 в очень ограниченных режимах, однако он редко используется. Терминалы 3270 представляли собой полноэкранные буферизованные неинтеллектуальные терминалы с управляющими модулями и потоками данных для обеспечения максимальной эффективности передачи данных.

3 EBCDIC обозначает Extended Binary Coded Decimal Interchange Code (расширенный двоично-десятичный код обмена информацией) и представляет кодированный набор символов, содержащий 256 8-разрядных символов, разработанный для представления текстовых данных. EBCDIC несовместим с набором символов ASCII. Удобная таблица преобразования представлена в приложении D, «Таблица EBCDIC – ASCII».

1.10.4 z/TPF

Операционная система z/Transaction Processing Facility (z/TPF) является системой особого назначения, используемой в компаниях с очень большими объемами транзакций, например в компаниях-владельцах торговой марки кредитной карты и системах бронирования авиабилетов; z/TPF раньше имела название Airline Control Program (ACP). Она все еще используется авиакомпаниями и была расширена для использования в других очень крупных системах высокоскоростной обработки большого объема транзакций.

z/TPF может использовать несколько мэйнфреймов в слабосвязанной среде для постоянной обработки десятков тысяч транзакций в секунду, обеспечивая при этом многолетнюю непрерывную доступность. Часто используются очень большие сети терминалов, включая сети специальных протоколов, применяемые в сфере бронирования.

1.11 Заключение

В настоящее время мэйнфрейм-компьютеры занимают центральное место в повседневной работе большинства крупнейших корпораций мира, включая многие компании из списка Fortune 1000. Несмотря на то что другие виды вычислительной техники широко используются в различных сферах деятельности, мэйнфреймы занимают завидное место в современной среде электронного бизнеса. В банковских, финансовых, здравоохранительных, страховых, коммунальных, правительственных и многих других государственных и частных учреждениях мэйнфрейм-компьютеры представляют основу современного бизнеса.

Популярность и долговечность мэйнфреймов в значительной степени обусловлена свойственной им надежностью и стабильностью, которые являются результатом продуманных технологических усовершенствований, выполненных со времени выпуска IBM System/360 в 1964 году. Ни одна другая компьютерная архитектура не может заявить о таком постоянном эволюционном развитии при обеспечении совместимости с существующими приложениями.

Термин *мэйнфрейм* постепенно изменил значение с физического описания больших компьютеров компании IBM на описание стиля вычислений. Одним из определяющих свойств мэйнфрейма является последовательная совместимость на протяжении десятилетий.

В ИТ-организациях, использующих мэйнфреймы, существует множество различных ролей и обязанностей. Для поддержки бесперебойной и надежной работы мэйнфрейм-компьютера требуется опытный персонал. Может показаться, что для обеспечения работы мэйнфрейм-среды требуется гораздо больше ресурсов, чем для небольших распределенных систем. Однако если в полной мере определить роли на стороне распределенных систем, они будут содержать множество тех же ролей.

В настоящее время существует несколько операционных систем для мэйнфреймов. Эта книга сосредоточена на одной из них – z/OS. Однако студентам, изучающим мэйнфреймы, необходимо знать о существовании других операционных систем и понимать их положение относительно z/OS.

Основные термины в этой главе				
Архитектура	Доступность	Пакетная обработка	Совместимость	Электронный бизнес
Мэйнфрейм	Обработка оперативных транзакций (OLTP)	Платформа	Аналитик производственного контроля	Документация
Масштабируемость	Масштабируемость	Системный оператор	Системный программист	System/360

1.12 Контрольные вопросы

Чтобы проверить понимание материала этой главы, ответьте на следующие вопросы:

1. Перечислите, каким образом современные мэйнфреймы изменяют традиционное представление о сопоставлении централизованных и распределенных вычислений.
2. Опишите, каким образом предприятия используют вычислительную мощность мэйнфреймов и в чем отличие мэйнфрейм-вычислений от других типов вычислений.
3. Назовите три преимущества мэйнфрейм-вычислений и опишите основные типы задач, для которых мэйнфреймы подходят лучше всего.
4. Назовите пять должностей или обязанностей, связанных с мэйнфрейм-вычислениями.
5. В этой главе было названо по меньшей мере пять операционных систем, используемых на мэйнфрейме. Выберите три из них и опишите их основные свойства.

1.13 Темы для дальнейшего обсуждения

1. Что представляет собой мэйнфрейм в настоящее время? Как возник этот термин? Адекватен ли он в настоящее время?
2. Почему важно обеспечивать совместимость системы со старыми приложениями? Почему бы просто не заменять существующие интерфейсы программирования приложений при выпуске более новых интерфейсов?
3. Опишите, каким образом использование мэйнфрейма позволяет сэкономить средства, учитывая большое количество ролей, необходимых для обслуживания мэйнфрейм-системы.
4. Какие свойства (хорошие или плохие) имеет вычислительная среда мэйнфрейма в связи с ролями, используемыми в мэйнфрейм-центрах? (эффективность? надежность? масштабируемость?).
5. В большинстве мэйнфрейм-центров реализованы строгие процедуры управления системами, безопасности и операций. Реализованы ли такие же процедуры в средах распределенных систем? Почему?

6. Найдите примеры использования мэйнфреймов в своей ежедневной деятельности? Опишите их, а также то, насколько мэйнфрейм-обработка очевидна для конечных пользователей. Можно использовать следующие примеры:
   - Популярные веб-сайты, использующие мэйнфрейм в качестве сервера заднего плана для поддержки оперативных транзакций и баз данных.
   - Мэйнфреймы, используемые в вашей местности. Это может включать банки и финансовые центры, крупные магазины, транспортные узлы, а также здравоохранительные и медицинские учреждения.
7. Найдите примеры распределенных систем в ежедневной деятельности? Можно ли было бы улучшить их работу путем добавления мэйнфрейма? Каким образом?

2

Аппаратные системы мэйнфрейма и высокая доступность

Цель. Как новому системному программисту в z/OS, вам важно иметь полное представление об оборудовании, на котором выполняется операционная система z/OS, которая разработана таким образом, чтобы обеспечить полное использование оборудования мэйнфрейма и его сложных периферийных устройств. Вам также необходимо понимать, каким образом аппаратное и программное обеспечение достигают почти постоянной доступности посредством использования таких концепций, как Parallel Sysplex и «отсутствие единых точек отказа».

После завершения работы над этой главой вы сможете:

- обсуждать устройство оборудования S/360 и zSeries;
- рассказать о процессорах и дисковом оборудовании;
- объяснить, в чем отличие мэйнфреймов от персональных компьютеров с точки зрения кодировки данных;
- перечислить несколько стандартных аппаратных конфигураций;
- объяснить, каким образом Parallel Sysplex позволяет достичь постоянной доступности;
- объяснить понятие динамической балансировки рабочей нагрузки;
- описать понятие единого образа системы.

2.1 Введение в аппаратные системы мэйнфрейма

Эта глава содержит общие сведения об аппаратных системах мэйнфрейма с основным акцентом на «процессорном блоке».

Дополнительные сведения. Подробное описание основных функций z/Architecture см. в руководстве *z/Architecture Principles of Operation*. Эту и другие публикации IBM можно найти на веб-сайте z/OS Internet Library

```
http://www.ibm.com/servers/eserver/zseries/zos/bkserv/
```

Начнем главу с обзора терминологии, связанной с оборудованием мэйнфрейма. Знание различных значений таких терминов, как *системы*, *процессоры*, *центральные процессоры* и т. д., является важным для понимания мэйнфрейм-компьютеров.

CPU – синоним понятия *процессор*

Во времена S/360 система имела единственный процессор, называемый центральным процессором (central processing unit, CPU). Термины *система*, *процессор* и *центральный процессор* использовались как взаимозаменяемые. Однако эти термины утратили ясность с появлением систем, имеющих больше одного процессора. Это показано на рис. 2.1.

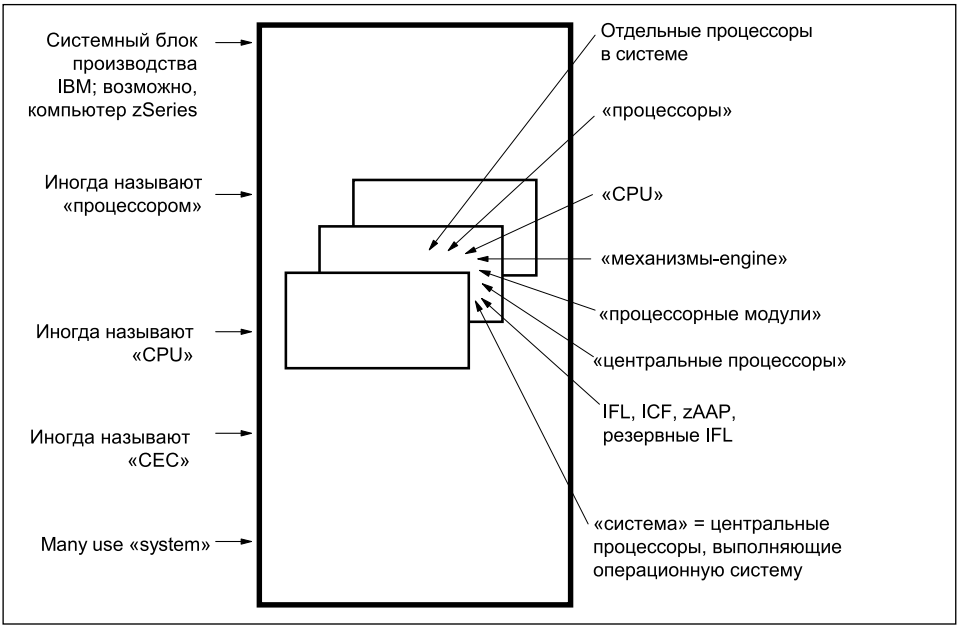


Рис. 2.1. Наложение терминологии

Терминами *процессор* и *CPU* часто называют либо весь системный блок, либо один из процессоров (CPU) в системном блоке. Несмотря на то что значение термина может быть понятным из контекста обсуждения, специалистам по мэйнфреймам важно уточнять, какое именно значение термина *процессор* или *CPU* используется в обсуждении. Системные программисты для обозначения системного блока мэйнф-

рейма используют применяемый в IBM термин *центральный процессорный комплекс (central processor complex, CPC)*. В данной книге термин CPC используется для обозначения физического набора аппаратного обеспечения, включающего основную память, один или несколько центральных процессоров, таймеры и каналы.

Центральный процессорный комплекс (CPC) – физический набор аппаратного обеспечения, включающий основную память, один или несколько центральных процессоров, таймеры и каналы

Управление разделами и некоторые термины из рис. 2.1 обсуждаются позже в этой главе. Если коротко, то все процессоры S/390 или z/Architecture в CPC являются процессорными модулями *(processing units, PU)*. Для процессорных модулей внутри CPC используются такие названия, как центральные процессоры (CP) для обычной работы, Integrated Facility for Linux (IFL), Integrated Coupling Facility (ICF) для конфигураций Parallel Sysplex и т. д.

В этой книге значение терминов *система* и *процессор* должно быть понятным из контекста. Обычно мы используем слово «система» для обозначения аппаратного блока, всей аппаратной среды (с устройствами ввода-вывода) или операционной среды (с программным обеспечением), в зависимости от контекста. Слово «процессор» обычно используется для обозначения одного процессора (CP) в центральном процессорном комплексе (CPC).

2.2 Устройство ранних систем

Центральный процессорный блок содержит процессоры, память[1], цепи управления и интерфейсы для *каналов*. Канал обеспечивает независимый путь передачи данных и управляющей информации между устройствами ввода-вывода и памятью. Ранние системы имели до 16 каналов; самые крупные мэйнфрейм-компьютеры на момент написания этой книги имели более 1 000 каналов.

Каналы подключаются к *устройствам управления (control units)*. Устройство управления содержит логику для работы с определенным типом устройств ввода-вывода. Например, устройство управления принтером имеет внутреннюю схему и логику, значительно отличающиеся от устройства управления приводом накопителей на магнитных лентах. Некоторые устройства управления могут иметь несколько подключений к каналам, обеспечивая несколько путей между устройством управления и его устройствами ввода-вывода.

Устройства управления подключаются к периферийным *устройствам (devices)*, таким, как дисковые приводы, приводы накопителей на магнитных лентах, коммуникационные интерфейсы и т. д. Разделение схемы и логики между устройством управления и его периферийными устройствами формально не определено, но обычно экономически более целесообразно разместить большую часть схемы в устройстве управления.

На рис. 2.2 представлена общая схема системы S/360. Подключение в современных системах отличается от представленного на рис. 2.2. Однако этот рисунок помогает понять основную терминологию, используемую при обсуждении мэйнфреймов.

[1] В некоторых мэйнфреймах S/360 использовались отдельные блоки для памяти. Однако обсуждение таких подробностей не настолько важно, и мы будем их игнорировать.

Каналы, показанные на рис. 2.2, представляют собой *параллельные каналы* (также называемые *каналами шины* и *тега*, названные так из-за двух тяжелых медных кабелей, которые они используют). Параллельный канал может быть подключен не более чем к восьми устройствам управления. Большинство устройств управления можно подключить к нескольким периферийным устройствам; показатель максимального количества устройств зависит от конкретного устройства управления, однако 16 – стандартное значение.

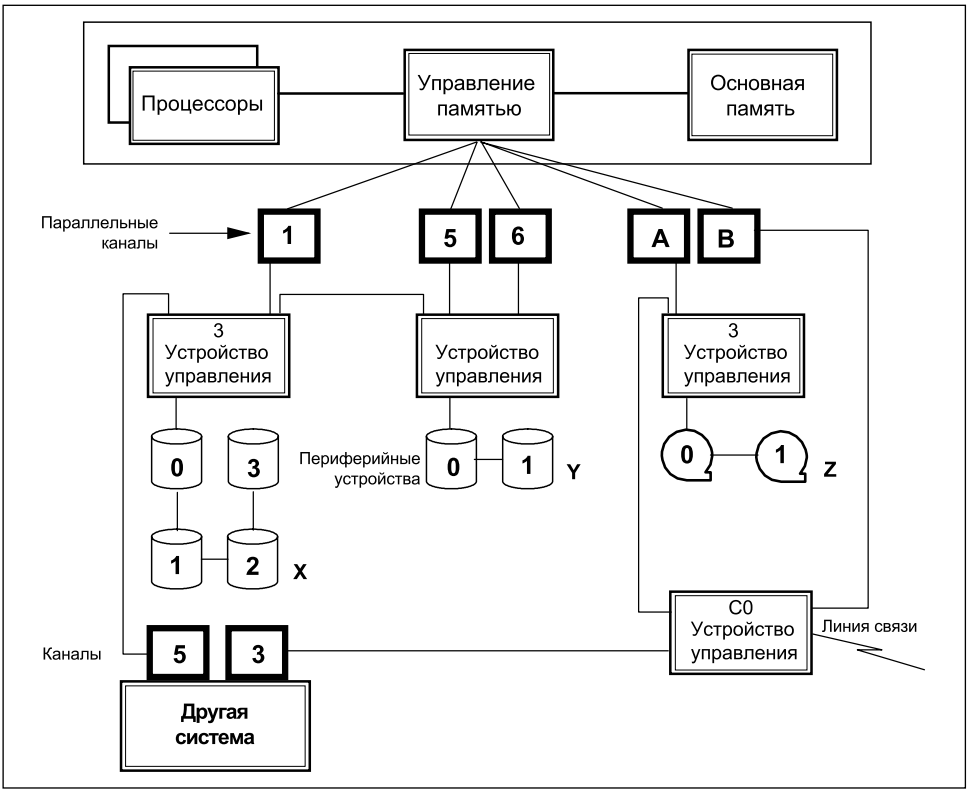


Рис. 2.2. Общая схема S/360

Каждый канал, устройство управления и периферийное устройство имеет адрес, выраженный шестнадцатеричным числом. Дисковый привод, обозначенный на рис. 2.2 буквой X, имеет адрес 132, полученный способом, представленным на рис. 2.3.

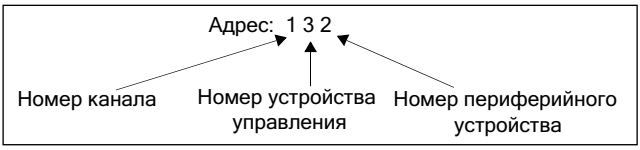


Рис. 2.3. Адрес устройства

Дисковому приводу, обозначенному на рисунке буквой Y, можно назначить адреса 171, 571 или 671, так как он подключен через три канала. По умолчанию используется наименьший адрес устройства (171), однако все три адреса могут использоваться операционной системой для доступа к дисковому приводу. Наличие нескольких путей к устройству полезно в целях производительности и доступности. Когда приложение запрашивает доступ к диску 171, операционная система сначала пытается получить доступ через канал 1. Если он занят (или недоступен), она пытается использовать канал 5 и т. д.

На рис. 2.2 представлена другая система S/360, у которой два канала подключены к устройствам управления, используемым первой системой. Такое совместное использование устройств ввода-вывода является распространенным во всех инсталляциях мэйнфреймов. Привод для носителей на магнитной ленте Z имеет адрес A31 в первой системе, однако во второй системе используется адрес 331. Совместное использование устройств, в особенности дисковых приводов, – непростая тема, и существуют аппаратные и программные методы, используемые операционной системой для предотвращения таких ситуаций, как одновременное изменение одних и тех же данных на диске с двух разных систем.

Как говорилось выше, современные мэйнфреймы не используются таким же образом, как мэйнфреймы, представленные на рис. 2.2. Существуют следующие различия:

- Параллельные каналы недоступны на новых мэйнфреймах и постепенно заменяются на старых системах.
- Параллельные каналы были заменены каналами ESCON® (Enterprise Systems CONnection) и FICON® (FIber CONnection). Эти каналы подключаются только к одному устройству управления или, что более вероятно, к *директору* (коммутатору) и представляют собой оптоволоконные кабели.

ESCON – Enterprise Systems Connection

- Современные мэйнфреймы имеют больше 16 каналов и используют две шестнадцатеричных цифры для обозначения части адреса, соответствующей каналу.
- На современных системах для обозначения каналов обычно используются идентификаторы канальных путей (channel path identifier, CHPID) или идентификаторы физических каналов (physical channel identifier, PCHID), хотя использование термина *канал* также корректно. Все каналы интегрированы в главный процессорный блок.

Адрес устройства, видимый программным обеспечением и более правильно называемый номером устройства (несмотря на то что термин *адрес* все еще широко используется), косвенно связан с устройством управления и адресами устройств.

Дополнительные сведения о разработке мэйнфреймов IBM с 1964 года см. в приложении A, «Краткий обзор истории мэйнфреймов IBM».

2.3 Устройство современных систем

Устройство современных центральных процессорных комплексов намного сложнее, чем устройство ранних систем S/360. Эта сложность проявляется во многих сферах:

- связь и конфигурирование устройств ввода-вывода;
- использование устройств ввода-вывода;
- управление разделами системы.

2.3.1 Связь устройств ввода-вывода

На рис. 2.4 представлена современная конфигурация. Реальные системы имеют больше каналов и устройств ввода-вывода, однако этот рисунок иллюстрирует основные понятия. Разделы, ESCON-каналы и FICON-каналы описаны ниже.

CHPID – идентификатор канального пути (channel path identifier)

Если коротко, то разделы создают отдельные логические машины в CPC. ESCON- и FICON-каналы логически подобны параллельным каналам, однако в них используются оптоволоконные соединения и они работают гораздо быстрее. Современная система может иметь 100 – 200 каналов или CHPID[1]. Основные понятия, частично представленные здесь, включают следующее:

- ESCON- и FICON-каналы подключаются только к одному устройству или одному порту коммутатора.
- В большинстве современных мэйнфреймов используются коммутаторы между каналами и устройствами управления. Коммутаторы могут подключаться к нескольким системам, совместно используя устройства управления и некоторые или все их устройства ввода-вывода во всех системах.
- Адреса CHPID содержат две шестнадцатеричные цифры.
- Несколько разделов иногда могут совместно использовать CHPID. Это зависит от типа устройств управления, используемых через CHPID. В целом CHPID, используемые для дисков, допускают совместное использование.
- Между операционной системой в разделах (или на основной машине, если разделы не используются) и CHPID существует уровень подсистемы ввода-вывода.

ESCON-директор или FICON-коммутатор представляют собой сложные устройства, поддерживающие высокую скорость передачи данных через несколько подключений (крупный директор, например, может иметь 200 подключений, и все они могут передавать данные одновременно). Директор или коммутатор должны регистрировать, какой CHPID (и раздел) какую операцию ввода-вывода инициировал, чтобы данные и информация о состоянии возвращались туда, куда нужно. Множество запросов ввода-вывода от множества CHPID, подключенных ко множеству разделов на множестве систем, могут обрабатываться одним устройством управления.

Уровень управления вводом-выводом использует управляющий файл, называемый IOCDS (I/O Control Data Set), преобразующий физические адреса ввода-вывода (куда относятся номера CHPID, номера портов коммутаторов, адреса устройств управления

[1] Самые последние мейнфреймы могут иметь более 256 каналов, но для этого необходимо выполнить дополнительную настройку. Каналы назначаются таким образом, что для составления адресов CHPID требуется только две шестнадцатеричные цифры.

и адреса периферийных устройств) в *номера устройств (device numbers)*, используемые программным обеспечением операционной системы для доступа к устройствам. Он загружается в область HSA (Hardware Save Area) при включении и допускают динамическое изменение. Номер устройства выглядит как адреса, описанные для ранних машин S/360, с тем исключением, что он может содержать три или четыре шестнадцатеричные цифры.

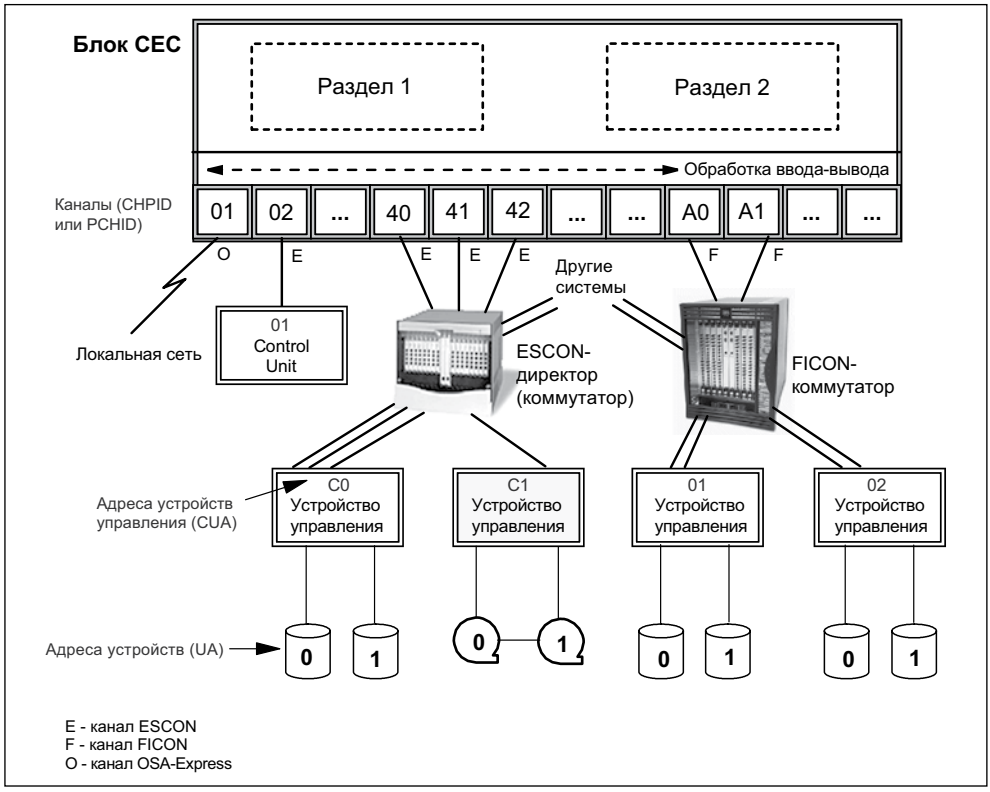


Рис. 2.4. Конфигурация современной системы

Многие пользователи все еще называют их «адресами», хотя номера устройств представляют собой произвольные числа между x'0000' и x'FFFF'. Последние мэйнфреймы (на момент написания этой книги) имеют два уровня преобразований адресов ввода-вывода между реальными элементами ввода-вывода и программным обеспечением операционной системы. Второй уровень был добавлен для упрощения миграции на новые системы.

Современные устройства управления, особенно для дисков, часто используют несколько подключений к каналам (или коммутаторам) и несколько подключений к своим устройствам. Они могут одновременно обрабатывать множество передач данных через несколько каналов. Каждое устройство имеет блок управления устройством (unit control block, UCB) в каждом образе z/OS.

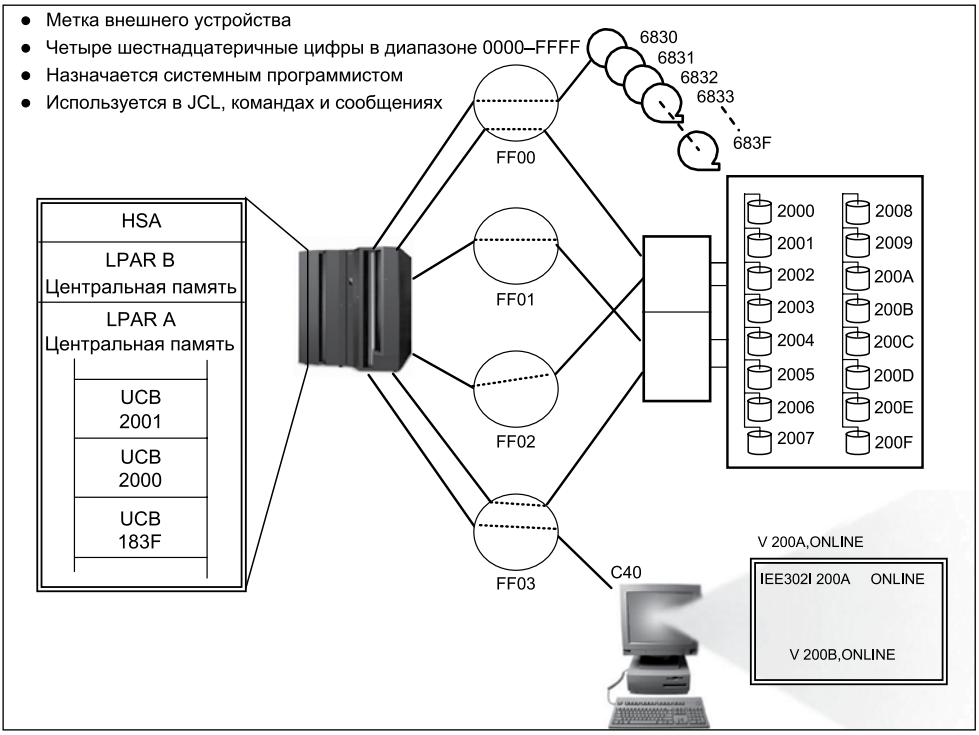


Рис. 2.5. Адресация устройств

2.3.2 Средства управления системой и разделы

Существует много способов проиллюстрировать внутреннюю структуру мэйнфрейма в зависимости от того, что требуется выделить. Рис. 2.6 очень абстрактно иллюстрирует некоторые функции внутренних системных средств управления на современных мэйнфреймах. В качестве внутренних контроллеров используются микропроцессоры с гораздо более простой организацией и набором инструкций, чем процессоры zSeries. Их обычно называют *контроллерами* во избежание путаницы с *процессорами* zSeries.

Среди функций управления системой существует возможность разделения системы на несколько *логических разделов (logical partition, LPAR)*. LPAR представляет собой поднабор процессорного оборудования, выделенный для поддержки операционной системы. LPAR содержит ресурсы (процессоры, память и устройства ввода-вывода) и функционирует как независимая система. Одна аппаратная мэйнфрейм-система может содержать несколько логических разделов.

Логический раздел – поднабор процессорного оборудования, выделенный для поддержки операционной системы

Многие годы существовало ограничение, позволявшее использовать не более 15 LPAR на мэйнфрейме; более современные машины имеют огра-

ничение в 30 логических разделов (возможно, и больше). Практические ограничения объема памяти, устройств ввода-вывода и доступной вычислительной мощности обычно устанавливают ограничения числа логических разделов, меньшие, чем эти максимальные показатели.

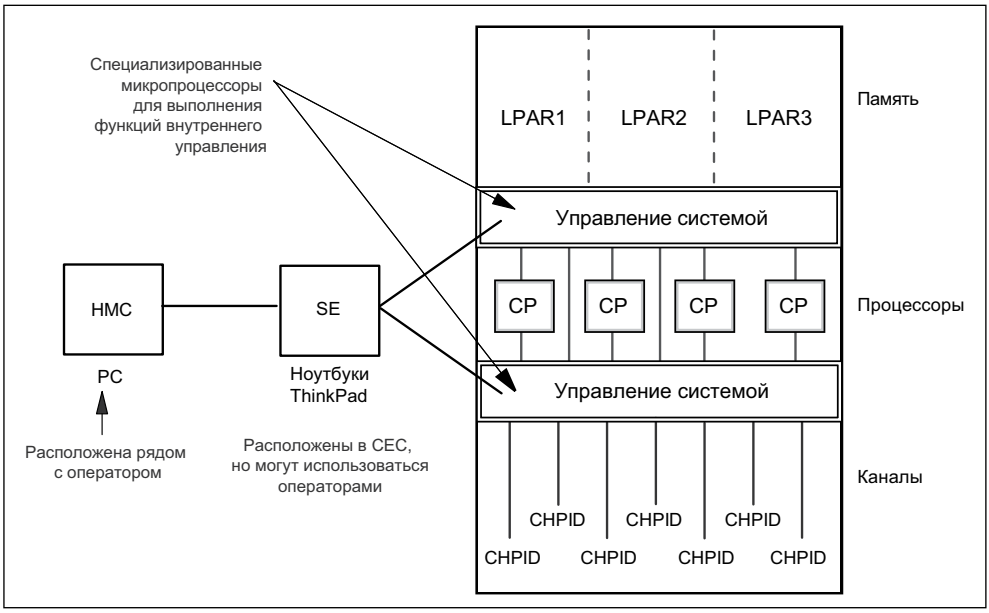


Рис. 2.6. Средства управления системой и разделы

Примечание. Аппаратное и микропрограммное обеспечение, реализующее функции управления разделами, имеет название PR/SM™ (Processor Resource/System Manager). Для создания и функционирования LPAR используются функции PR/SM. Различие между понятиями PR/SM (встроенное средство) и LPAR (результат использования PR/SM) часто игнорируется, и термин LPAR используется в собирательном значении для обозначения как средства, так и результатов его работы.

Системные администраторы выделяют участки памяти для каждого логического раздела; память не может совместно использоваться разными LPAR. Администраторы могут назначать процессоры (обозначенные как CP на рис. 2.6) определенным LPAR или позволять системным контроллерам осуществлять распределение некоторых или всех процессоров среди всех LPAR с использованием встроенного механизма балансировки нагрузки. Каналы (CHPID) могут назначаться определенным LPAR или совместно использоваться несколькими LPAR, в зависимости от типа устройств на каждом канале.

Система с одним процессором (CP) может иметь несколько LPAR. PR/SM содержит встроенный диспетчер, который может выделять часть процессора для каждого LPAR, подобно тому как диспетчер операционной системы выделяет часть процессорного времени каждому процессу, потоку или заданию.

Параметры управления разделами частично описываются в файле IOCDS, а частично описываются в системном *профиле (profile)*. Как IOCDS, так и профиль находятся в элементе поддержки (Support Element, SE), который представляет собой простой ноутбук внутри системы. SE может быть подключен к одному или нескольким консолям управления аппаратными средствами (Hardware Management Console, HMC), которые представляют собой настольные персональные компьютеры, используемые для мониторинга и управления оборудованием, в частности микропроцессорами мэйнфрейма. HMC более удобен в использовании, чем SE, и может управлять несколькими мэйнфреймами.

HMC – консоль, используемая для мониторинга и управления аппаратными средствами, в частности микропроцессорами мейнфрейма

Работая с HMC (или с SE, в нестандартной ситуации), оператор готовит мэйнфрейм к использованию путем выбора и загрузки профиля и IOCDS. Это создает логические разделы и настраивает каналы, задавая им номера устройств, назначения LPAR, различную информацию о путях и т. д. Это называется сбросом по питанию (Power-on Reset, POR). Загружая другой профиль и IOCDS, оператор может полностью изменить количество и тип логических разделов, а также конфигурацию ввода-вывода. Это обычно нарушает работу запущенных операционных систем и приложений, поэтому редко выполняется без заблаговременного планирования.

2.3.3 Свойства логических разделов

Логические разделы на практике равнозначны отдельным мэйнфреймам. На каждом LPAR выполняется отдельная операционная система. Это может быть любая операционная система для мэйнфреймов; другими словами, необязательно запускать z/OS на каждом LPAR. Специалисты по планированию установки могут решить установить совместное использование устройств ввода-вывода несколькими LPAR, однако такие решения принимаются на месте.

Системный администратор может выделить один или несколько процессоров системы для эксклюзивного использования одним LPAR. С другой стороны, администратор может разрешить использование всех процессоров некоторыми или всеми LPAR. В данном случае функции управления системой (часто называемые микрокодом или микропрограммным обеспечением) содержат диспетчер, осуществляющий совместное использование процессоров выбранными логическими разделами. Администратор может задать максимальное количество процессоров, одновременно задействованных в каждом логическом разделе. Администратор может также задать весовые коэффициенты для разных логических разделов, например определить для LPAR1 в два раза больше процессорного времени, чем для LPAR2.

Операционная система в каждом логическом разделе загружается отдельно, при этом используется отдельная копия операционной системы[1], используется отдельная консоль оператора (если требуется) и т. д. Если происходит отказ системы на одном LPAR, это не влияет на другие LPAR.

Например, на рис. 2.6 может использоваться рабочая система z/OS на LPAR1, тестовая версия z/OS на LPAR2 и Linux для S/390 на LPAR3. Если в сумме система имеет

[1] Большинство (но не все) системных библиотек z/OS допускают совместное использование.

8 Гб памяти, можно выделить 4 Гб для LPAR1, 1 Гб для LPAR2, 1 Гб для LPAR3 и оставить 2 Гб в резерве на всякий случай. Консоли операционной системы для двух логических разделов z/OS могут располагаться в совершенно разных местах[1].

На практике в большинстве случаев нет разницы между, например, тремя отдельными мэйнфреймами с запущенной z/OS (и с совместным использованием большей части конфигурации ввода-вывода) и тремя LPAR на одном мэйнфрейме, настроенных таким же образом. За редким исключением z/OS, операторы и приложения не могут обнаружить разницу.

Ко второстепенным отличиям относятся способность z/OS (если разрешено при определении LPAR) получить информацию о производительности и использовании по всей мэйнфрейм-системе и динамически перераспределять использование ресурсов (процессоров и каналов) между разными LPAR для улучшения производительности.

2.3.4 Консолидация мэйнфреймов

В настоящее время используется меньше мэйнфреймов, чем 15 или 20 лет назад. В некоторых случаях все приложения были перенесены на системы других типов. Однако в большинстве случаев меньшее число вызвано консолидацией. Другими словами, несколько небольших мэйнфреймов были заменены на меньшее число более крупных систем.

Существует убедительный аргумент в пользу консолидации. Программное обеспечение мэйнфреймов (от различных изготовителей) может быть дорогостоящим и обычно стоит больше, чем оборудование мэйнфрейма. Часто бывает выгоднее (и иногда *намного* выгоднее) заменить несколько лицензий на программное обеспечение (для небольших машин) на одну-две лицензии (для больших машин). Стоимость лицензии на программное обеспечение часто привязана к мощности системы, однако кривые цен говорят в пользу небольшого количества крупных машин.

Стоимость лицензий на программное обеспечение для мэйнфреймов стала основным фактором в росте и направлении развития отрасли мэйнфреймов. Существует несколько нелинейных факторов, которые значительно затрудняют формирование цен на программное обеспечение. Необходимо помнить о том, что рынок программного обеспечения для мэйнфреймов не является таким массовым, как рынок программ для персональных компьютеров. Рост вычислительной мощности мэйнфреймов в последние годы происходил нелинейно.

Относительная мощность, необходимая для выполнения традиционного приложения для мэйнфреймов (например, пакетного задания на языке COBOL), не имеет линейной связи с мощностью, необходимой для нового приложения (имеющего графический интерфейс и написанного на C или Java). Эффект консолидации привел к появлению сверхмощных мэйнфреймов. Для выполнения приложения может быть достаточно 1% их мощности, однако изготовители приложений часто назначают цену, исходя из общей мощности машины.

В результате возникла необычная ситуация, где клиенты хотят приобрести самый новый мэйнфрейм (чтобы использовать новые функции или снизить стоимость обслуживания, связанную со старыми машинами), но при этом они ищут *самый мед-*

[1] В Linux не используется консоль оператора в том же смысле, что и консоли z/OS.

ленный мэйнфрейм для выполнения своих приложений (чтобы снизить затраты на программное обеспечение, цена на которое привязывается к суммарной мощности системного процессора).

2.4 Процессорные устройства

На рис. 2.1 показано несколько различных типов процессоров в системе. Все они являются процессорами z/Architecture, которые можно использовать для несколько различающихся целей[1]. Некоторые из этих целей связаны с контролем стоимости программного обеспечения, тогда как другие являются более принципиальными.

z/Architecture – архитектура IBM для мейнфрейм-компьютеров и периферии. Семейство серверов zSeries использует архитектуру z/Architecture

Все эти процессоры в начале являются эквивалентными процессорными устройствами[2] (processor unit, PU) или механизмами (engines). Процессорное устройство представляет собой процессор, который изначально не был специализирован для определенной задачи. Каждый из процессоров в начале является процессорным устройством и специализируется компанией IBM во время установки или позднее.

Возможные специализации:

- Центральный процессор (Central Processor, CP).
  Это процессор, доступный для обычной операционной системы и прикладного программного обеспечения.
- Вспомогательный системный процессор (System Assistance Processor, SAP).
  Каждый современный мэйнфрейм имеет по меньшей мере один SAP; более крупные системы могут иметь несколько таких процессоров. SAP выполняют внутренний код[3] для обеспечения работы подсистемы ввода-вывода. Например, SAP преобразует номера устройств и реальные адреса CHPID, адреса устройств управления и периферийных устройств. Он управляет путями к устройствам управления и выполняет восстановление после временных ошибок. Операционные системы и приложения не могут обнаружить SAP, и SAP не используют «обычную» память.
- Интегрированный процессор для Linux (Integrated Facility for Linux, IFL).
  Представляет собой обычный процессор с одной-двумя отключенными инструкциями, используемыми только в z/OS. Linux не использует эти инструкции и может выполняться на IFL. Linux может также выполняться на CP. Различие состоит в том, что IFL не учитывается при определении модельного номера[4] для системы. Это может существенно снизить стоимость программного обеспечения.

[1] Не путайте их с микропроцессорами-контроллерами. Процессоры, рассматриваемые в этом разделе, представляют собой полнофункциональные стандартные процессоры для мейнфреймов.

[2] Это обсуждение относится к современным (на момент написания книги) машинам zSeries. В более ранних системах процессоры имели меньше специализаций, а еще более ранние системы вообще не использовали эти технологии.

[3] В IBM он называется лицензированным внутренним кодом (Licensed Internal Code, LIC). Он также часто называется микрокодом (что некорректно с технической точки зрения) или микропрограммным обеспечением. И конечно же, это не пользовательский код.

[4] В некоторых системах не поддерживается применение нескольких моделей; в этом случае используется *модельный номер по мощности (capacity model number)*.

- zAAP.
  Представляет собой процессор со множеством отключенных функций (обработка прерываний, некоторые инструкции), так что полная операционная система не может выполняться на этом процессоре. Однако z/OS может обнаруживать наличие zAAP-процессоров и использовать их для выполнения Java-кода (и возможно, другого подобного кода в будущем). Этот же Java-код может выполняться и на стандартном CP. Однако zAAP-процессоры опять же не учитываются при определении модельного номера системы. Подобно IFL, они нужны только для снижения стоимости программного обеспечения.
- zIIP.
  zIIP (System z9 Integrated Information Processor) представляет собой специализированный процессор для обработки задач, связанных с базами данных. zIIP служит для снижения стоимости программного обеспечения, предназначенного для определенных задач на мэйнфрейме, например задач бизнес-аналитики (business intelligence, BI), управления ресурсами предприятия (enterprise resource planning, ERP) и управления связями с клиентами (customer relationship management, CRM). zIIP усиливает использование мэйнфрейма в роли концентратора данных предприятия, что помогает сделать прямой доступ к DB2 более экономичным и устранить необходимость в нескольких копиях данных.
- Процессор внутреннего сопряжения (Integrated Coupling Facility, ICF).
  Эти процессоры выполняют только лицензированный внутренний код. Они невидимы для обычных операционных систем и приложений. Устройство сопряжения в действительности представляет собой разделяемую память большого объема, используемую несколькими системами для согласования работы. ICF-процессоры должны назначаться логическим разделам, которые затем используются в качестве устройств сопряжения.
- Резервные процессоры.
  Процессорные устройства без специализации используются в качестве резервных. Когда системные контроллеры обнаруживают отказ CP или SAP, его можно заменить резервным процессорным устройством. В большинстве случаев это можно сделать без прерывания работы системы, даже если приложение выполняется на отказавшем процессоре.
- Существуют различные формы технологии изменения производительности по запросу (*Capacity on Demand)* и подобных технологий, в которых клиент может включать дополнительные CP в определенные периоды (например, при неожиданной пиковой нагрузке).

Помимо этих специализаций процессоров, существуют модели или версии мэйнфреймов, настроенные на меньшую скорость работы, чем потенциальная скорость их центральных процессоров. Это часто называется *kneecapping*, хотя IBM предпочитает термин *ограничение мощности (capacity setting)* или что-то подобное. Это достигается путем использования микрокода, вставляющего нулевые циклы в поток инструкций процессора. Цель, опять же, состоит в том, чтобы снизить стоимость программного обеспечения путем приобретения минимальной модели или версии

мэйнфрейма, соответствующей требованиям приложения. IFL, SAP, zAAP и ICF всегда работают на полной скорости, так как эти процессоры не учитываются в формировании цен на программное обеспечение[1].

2.5 Мультипроцессоры

Во всех приведенных выше обсуждениях и примерах предполагается, что система (и возможно, логический раздел) содержит несколько процессоров (CP). Можно купить современный мэйнфрейм с одним процессором (CP), но такие системы являются редкостью[2]. Термин «мультипроцессор» указывает на наличие нескольких процессоров (CP) и предполагает использование нескольких процессоров копией z/OS.

Мультипроцессор CPC – мультипроцессор, который можно физически разделить для создания двух операционных процессорных комплекса

Все современные операционные системы, от персональных компьютеров до мэйнфреймов, могут работать в мультипроцессорной среде. Однако степень интеграции нескольких процессоров значительно варьируется. Например, отложенные прерывания в системе (или в LPAR) могут быть приняты любым процессором в системе (или в LPAR). Любой процессор может инициировать и управлять операциями ввода-вывода через любой канал или устройство, доступное для системы или LPAR. Каналами, устройствами ввода-вывода, прерываниями и памятью владеет система (или LPAR), а не какой-либо отдельный процессор.

Такая мультипроцессорная интеграция кажется простой на первый взгляд, но является сложной в реализации. Кроме того, она важна для обеспечения максимальной производительности; особенно важна способность любого процессора принимать любое прерывание, посылаемое в систему (или в LPAR).

Каждый процессор в системе (или в LPAR) имеет небольшую личную область памяти (8 Кб, начиная с реального адреса 0, и всегда отображаемую с виртуального адреса 0), уникальную для этого процессора. Эта область называется областью хранения префикса (Prefix Storage Area, PSA) и используется для обработки прерываний и ошибок. Процессор может осуществлять доступ к PSA другого процессора посредством специального программирования, хотя это обычно применяется только в целях восстановления после ошибок. Процессор может прерывать другие процессоры, используя специальную инструкцию (SIGP, Signal Processor). Опять же, это обычно применяется только для восстановления после ошибок.

2.6 Дисковые устройства

На современных мэйнфреймах часто используются дисковые приводы IBM 3390. С абстрактной точки зрения, подключение выполняется просто, как показано на рис. 2.7.

---

[1] Это относится к программному обеспечению IBM, но может не распространяться на программное обеспечение других изготовителей.

[2] Все современные мейнфреймы IBM требуют наличия по меньшей мере одного SAP, так что система в минимальной конфигурации имеет два процессора: один CP и один SAP. Однако при использовании слова «процессор» в этой книге обычно имеется в виду CP-процессор, доступный для использования приложениями. Если обсуждается процессор, отличный от CP, это всегда оговаривается особо.

Соответствующее устройство управления (3990) обычно имеет четыре канала, подключенные к одному или нескольким процессорам (возможно, через коммутатор), и дисковое устройство 3390 обычно содержит восемь и более дисковых приводов. Каждый дисковый привод имеет свойства, описанные выше. На рисунке представлены устройства 3990 и 3390, что также представляет концепцию или архитектуру современных устройств.

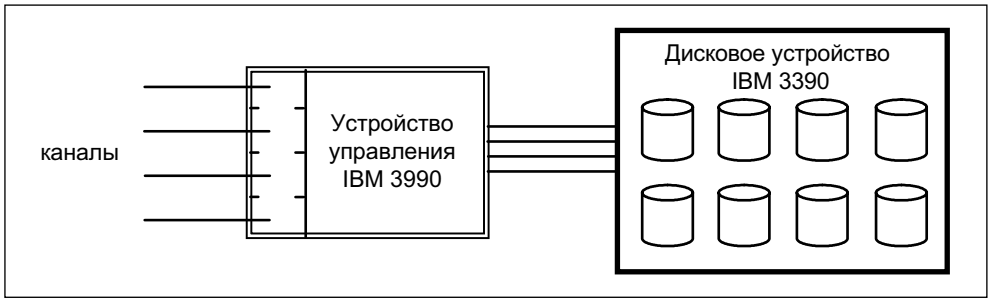


Рис. 2.7. Первоначальная реализация подключения дисков IBM 3390

Аналогичным современным устройством является IBM 2105 Enterprise Storage Server®, упрощенно представленный на рис. 2.8.

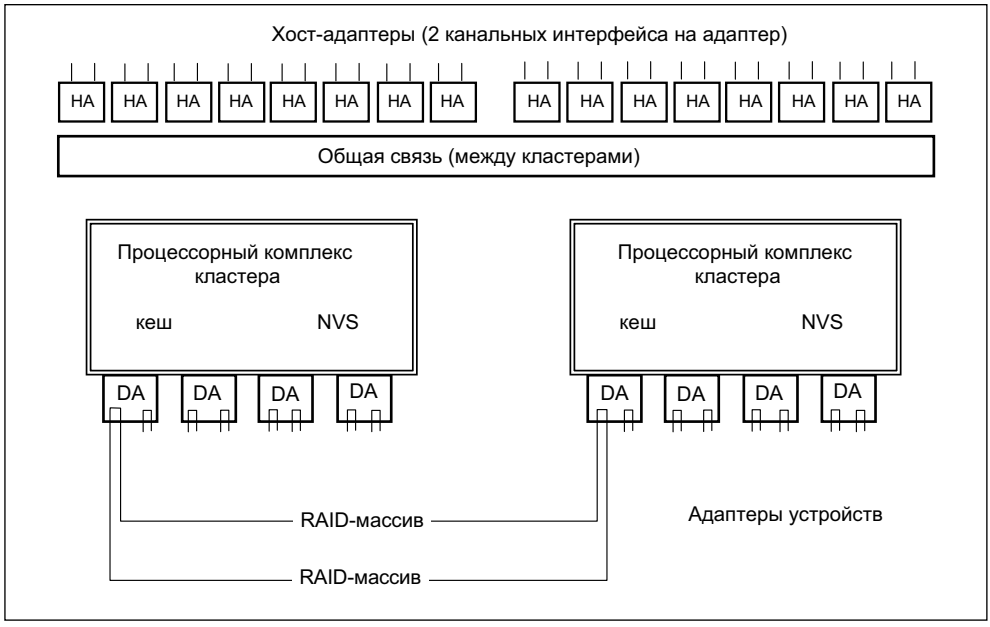


Рис. 2.8. Современная реализация 3390

Модуль 2105 представляет собой многофункциональное устройство, эмулирующее большое количество устройств управления и дисковых приводов 3390. Оно может иметь до 11 Тб дискового пространства, содержать до 32 канальных интерфейсов,

16 Гб кеша и 284 Мб энергонезависимой памяти (используемой для управления очередями записи). Хост-адаптеры представляют собой интерфейсы устройств управления и допускают подключение до 32 каналов (ESCON или FICON).

Физические дисковые приводы представляют собой обычные SCSI-подобные устройства (несмотря на использование последовательного интерфейса SSA для обеспечения быстрого и избыточного доступа к дискам). Возможно использование множества различных внутренних организаций, однако к наиболее распространенным относятся массивы RAID-5 с горячим резервированием. Практически каждый компонент модуля имеет соответствующий резервный компонент. Внутренняя обработка (для эмуляции устройств управления 3990 и дисков 3390) выполняется четырьмя высокопроизводительными процессорами RISC в двух процессорных комплексах; каждый комплекс может управлять всей системой. Внутренние батареи защищают передаваемые данные во время коротких перебоев питания. Для конфигурирования и управления модулем используется отдельная консоль.

Модуль 2105 содержит многие функции, недоступные в реальных устройствах 3390, включая FlashCopy®, расширенное удаленное копирование (Extended Remote Copy), одновременное копирование (Concurrent Copy), тома параллельного доступа (Parallel Access Volumes), множественную подчиненность (Multiple Allegiance), огромный кеш и т. д.

В простых дисковых приводах 3390 (с устройством управления) используется технология, отличная от описанной для 2105. Однако с точки зрения основного архитектурного представления для программного обеспечения они одинаковы. Это позволяет приложениям и системному программному обеспечению, написанному для дисковых приводов 3390, использовать новую технологию без изменений[1].

Новая технология, воспроизводящая дисковые приводы 3390, прошла несколько этапов; последним этапом является модуль 2105. Процесс поддержки архитектурного стандарта (в данном случае дискового привода 3390 с соответствующим устройством управления) с более новой и отличающейся технологией при обеспечении программной совместимости является одним из свойств разработки для мэйнфреймов. Как уже несколько раз говорилось, обеспечение совместимости приложений в течение длительного периода технологических изменений является важным свойством мэйнфреймов.

2.7 Кластеризация

Кластеризация на мэйнфреймах применялась со времен появления S/360, хотя термин *кластер* редко используется в этой сфере. Технологии кластеризации могут быть настолько простыми как конфигурация общего DASD, при которой для недопущения нежелательного наложения данных требуется ручное управление или планирование.

С годами были добавлены дополнительные технологии кластеризации. В следующих абзацах обсуждается три уровня кластеризации: простой общий DASD, CTC-

[1] Некоторые изменения в программном обеспечении необходимы для использования некоторых новых функций, однако они реализованы в виде совместимых расширений на уровне операционной системы и не влияют на приложения.

кольца и Parallel Sysplex. В настоящее время в большинстве инсталляций z/OS используется один или несколько из вышеперечисленных уровней; одиночные инсталляции z/OS встречаются относительно редко.

В этой книге мы используем термин «образ». Система z/OS (с одним или несколькими процессорами) является *образом z/OS*. Образ z/OS может существовать на сервере S/390, или zSeries (с логическими разделами), или на логическом разделе, либо он может выполняться под управлением z/VM (гипервизорной операционной системы, рассматриваемой в разделе 1.10, «z/OS и прочие операционные системы мэйнфреймов»). Система с шестью LPAR (каждый из которых представляет отдельную систему z/OS) имеет шесть образов z/OS. Мы используем термин «образ», чтобы показать, что не имеет значения, где именно выполняется система z/OS (на базовой системе, LPAR или в z/VM).

2.7.1 Простой общий DASD

Среда простого общего DASD представлена на рис. 2.9. На этом рисунке показаны образы z/OS, однако вместо них могут использоваться любые более ранние версии операционной системы. Может использоваться два логических раздела в одной системе или две отдельные системы; с точки зрения функционирования нет никакой разницы.

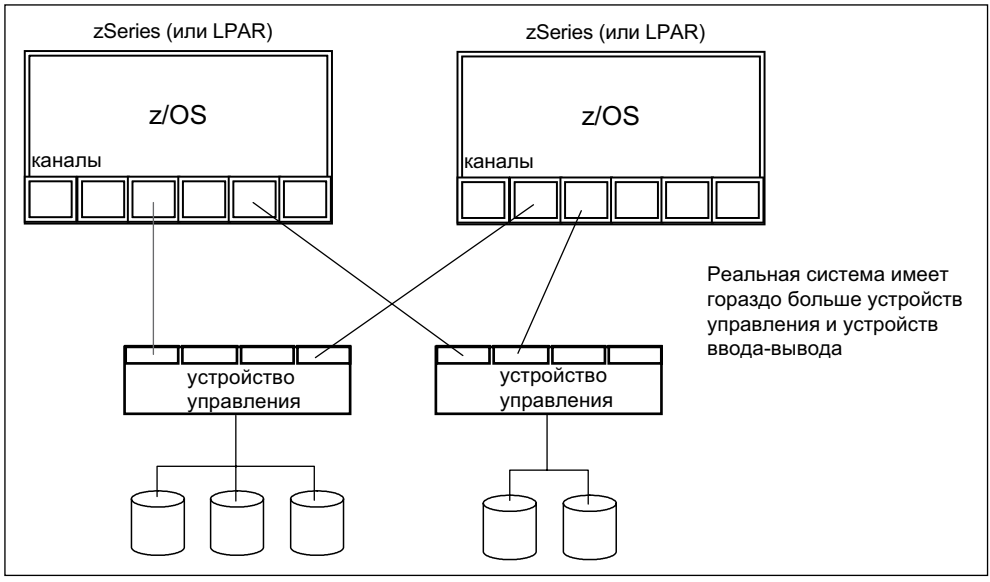


Рис. 2.9. Простой общий DASD

Возможности простой общей системы DASD ограничены. Операционные системы автоматически передают команды RESERVE и RELEASE в DASD перед изменением оглавления тома (VTOC) или каталога (как обсуждается в главе 5, «Работа с наборами данных», VTOC и каталоги являются структурами, содержащими метаданные для DASD, указывающими, где находятся различные наборы данных). Команда RESERVE ограничивает доступ к DASD только системой, передавшей команду, и это продолжа-

ется до передачи команды RELEASE. Эти команды хорошо работают для ограниченных периодов (например, при обновлении метаданных). Приложения также могут передавать команды RESERVE/RELEASE для защиты своих наборов данных при выполнении приложения. В рассматриваемой среде это не происходит автоматически и редко применяется на практике, так как при этом блокируется доступ других систем к DASD на длительный период времени.

Система с простым общим DASD обычно используется в случаях, когда операционный персонал управляет тем, какие задачи какими системами обрабатываются, и обеспечивает отсутствие конфликтов, например при попытке одновременного изменения одних и тех же данных двумя разными системами. Несмотря на такое ограничение, среда с простым общим DASD очень полезна для тестирования, восстановления и точной балансировки нагрузки.

Другие типы устройств ввода-вывода или устройств управления могут подключаться к обеим системам. Например, устройство управления для накопителей на магнитных лентах с несколькими приводами может быть подключено к обеим системам. При такой конфигурации операторы могут при необходимости назначать системам отдельные приводы для магнитных лент.

2.7.2 CTC-кольца

На рис. 2.10 показан следующий уровень кластеризации. Он использует такой же общий DASD, как и в приведенном выше описании, но также использует соединение канал-канал (*channel-to-channel*, *CTC*) между системами. Такая конфигурация называется *CTC-кольцом* (кольцевая природа становится более очевидной при использовании более двух систем).

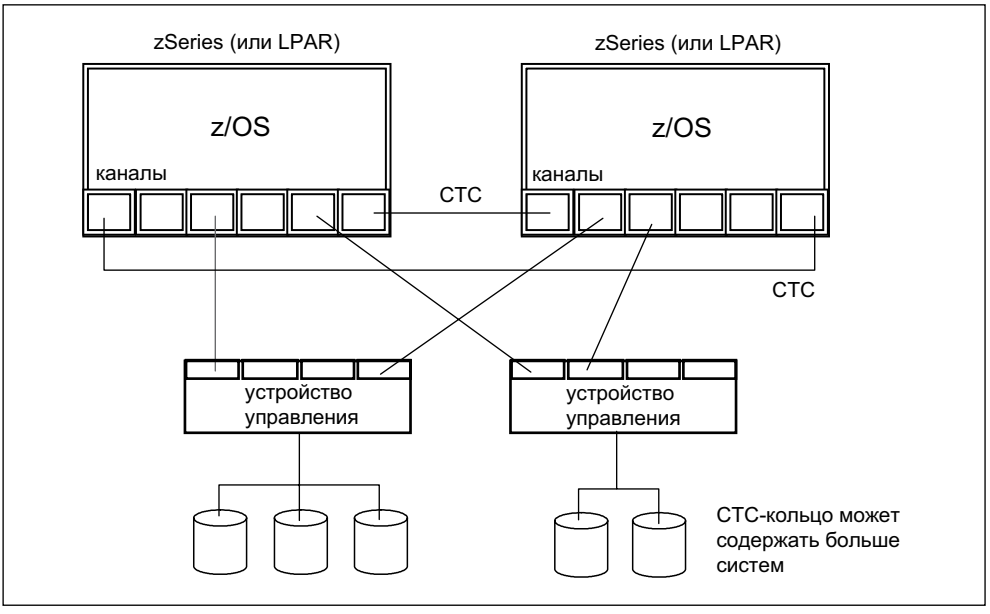


Рис. 2.10. Базовый сисплекс

z/OS может использовать CTC-кольцо для передачи управляющей информации между всеми системами в кольце. Таким образом, в частности, можно передавать следующую информацию:

- Информацию об использовании и блокировке наборов данных на дисках. Это позволяет системе автоматически не допускать нежелательное дублирование доступа к наборам данных. Такая блокировка основана на спецификациях JCL для заданий, передаваемых в систему, как описывается в главе 6 «Использование JCL и SDSF».
- Информацию об очередях заданий, например о том, что все системы в кольце могут принимать задания из одной входной очереди. Подобным образом, все системы могут передавать печатный вывод в одну выходную очередь.
- Параметры управления безопасностью, позволяющие принимать единообразные решения, связанные с безопасностью, для всех систем.
- Параметры управления дисковыми метаданными, что делает необязательным использование команд RESERVE и RELEASE.

В значительной степени пакетные задания и интерактивные пользователи могут работать в любой системе в этой конфигурации, так как все дисковые наборы данных доступны из любого образа z/OS.

Задания (и интерактивных пользователей) можно назначать любой системе с наименьшей нагрузкой на заданный момент времени.

CTC-подключение – подключение между двумя CHPID на одном или разных процессорах, либо прямое, либо через коммутатор

Когда конфигурации CTC использовались впервые, основной управляющей информацией совместного использования была информация о блокировке. Как обсуждается в разделе «Синхронизация использования ресурсов», занимающийся этим компонент z/OS называется функцией глобальной синхронизации ресурсов (global resource serialization); такая конфигурация называется GRS-кольцом. Основным ограничением GRS-кольца является задержка при отправлении сообщений по кольцу.

До появления кольцевой технологии использовалась другая конфигурация, CTC. Она требовала использования двух CTC-подключений от каждой системы к каждой другой системе в конфигурации. При использовании больше двух-трех систем это становилось сложным и требовало использования большого числа каналов.

Ранние конфигурации CTC («каждая система-каждая система» или кольцевая конфигурация) развились в конфигурацию базового *сисплекса (sysplex)*. Она включает управляющие наборы данных на общем DASD. Эти наборы данных используются для согласования операционных спецификаций на всех системах и для сохранения информации после перезапуска систем.

Конфигурации с общим DASD, CTC-подключениями и общими очередями заданий называются *слабосвязанными системами (loosely coupled systems)*. Мультипроцессоры, в которых несколько процессоров используются операционной системой, иногда называют *сильносвязанными системами (tightly coupled systems)*, но этот термин используется редко. Они также называются симметрическими мультипроцессорами (Symmetrical MultiProcessors, SMP); термин SMP часто используется на системах RISC, но на мэйнфреймах обычно не используется.

2.8 Что такое Parallel Sysplex?

Сисплексом (sysplex) называется набор систем z/OS, осуществляющих совместную обработку, используя определенные аппаратные и программные продукты. Он представляет собой технологию кластеризации, позволяющую обеспечить почти непрерывную доступность.

Традиционные большие компьютерные системы также используют аппаратные и программные продукты, осуществляющие совместную обработку. Основное различие между сисплексом и традиционной большой компьютерной системой состоит в более высоком потенциале роста и уровне доступности сисплекса. В сисплексе увеличено количество процессорных модулей и операционных систем z/OS, которые могут работать совместно, что, в свою очередь, повышает объем задач, который можно обработать. Для упрощения совместной работы были разработаны новые продукты и доработаны старые.

Parallel Sysplex представляет собой сисплекс, использующий технологию совместного применения данных несколькими системами. Это обеспечивает прямой параллельный доступ для чтения и записи к общим данных со всем узлов обработки (или серверов) в конфигурации без влияния на производительность или целостность данных.

Parallel Sysplex – сисплекс, использующий одно или несколько устройств сопряжения

Каждый узел может осуществлять одновременное кеширование совместно используемых данных в памяти локального процессора с применением средств аппаратной синхронизации и согласования в масштабе кластера.

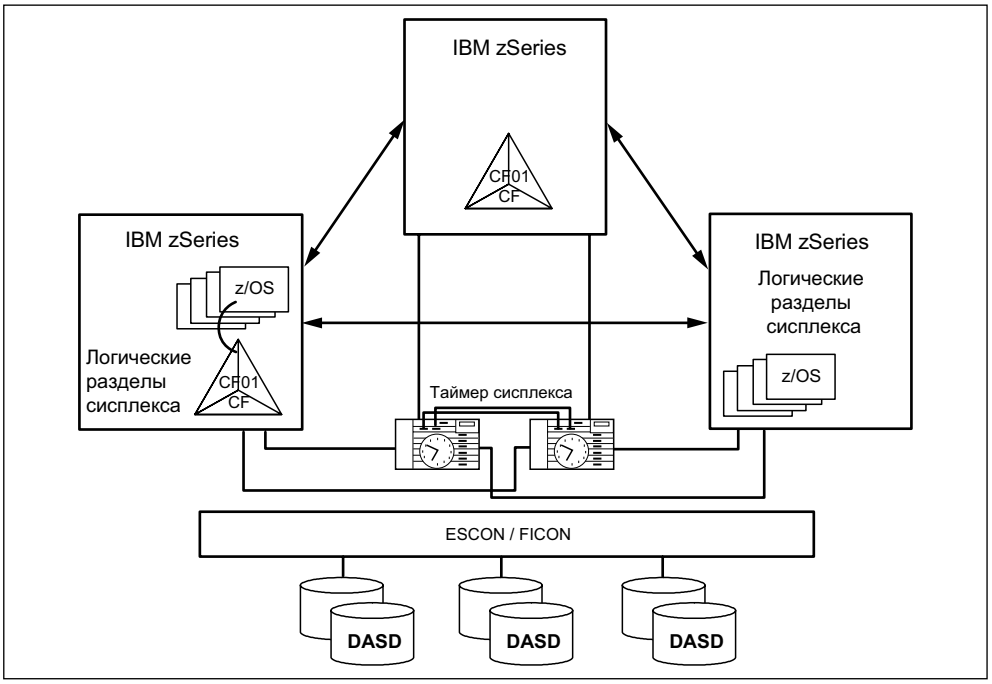


Рис. 2.11. Обзор оборудования сисплекса

В результате можно динамически распределить рабочие запросы, связанные с определенной задачей, например бизнес-транзакциями или запросами к базам данных, для параллельного выполнения на узлах в сисплексном кластере, исходя из доступной процессорной мощности.

На рис. 2.11 представлены видимые части Parallel Sysplex, а именно оборудование. Эти части являются основными компонентами Parallel Sysplex, реализованными в аппаратной архитектуре.

2.8.1 Что такое устройство сопряжения?

Parallel Sysplex использует одно или несколько устройств сопряжения (coupling facilities, CF). Устройство сопряжения представляет собой процессор мэйнфрейма с памятью и специальными каналами, а также со встроенной операционной системой. Оно не имеет каких-либо устройств ввода-вывода, кроме специальных каналов, и его операционная система очень мала[1].

Устройство сопряжения – специальный логический раздел, обеспечивающий высокоскоростное кеширование, обработку списков и функции блокировки в сисплексе

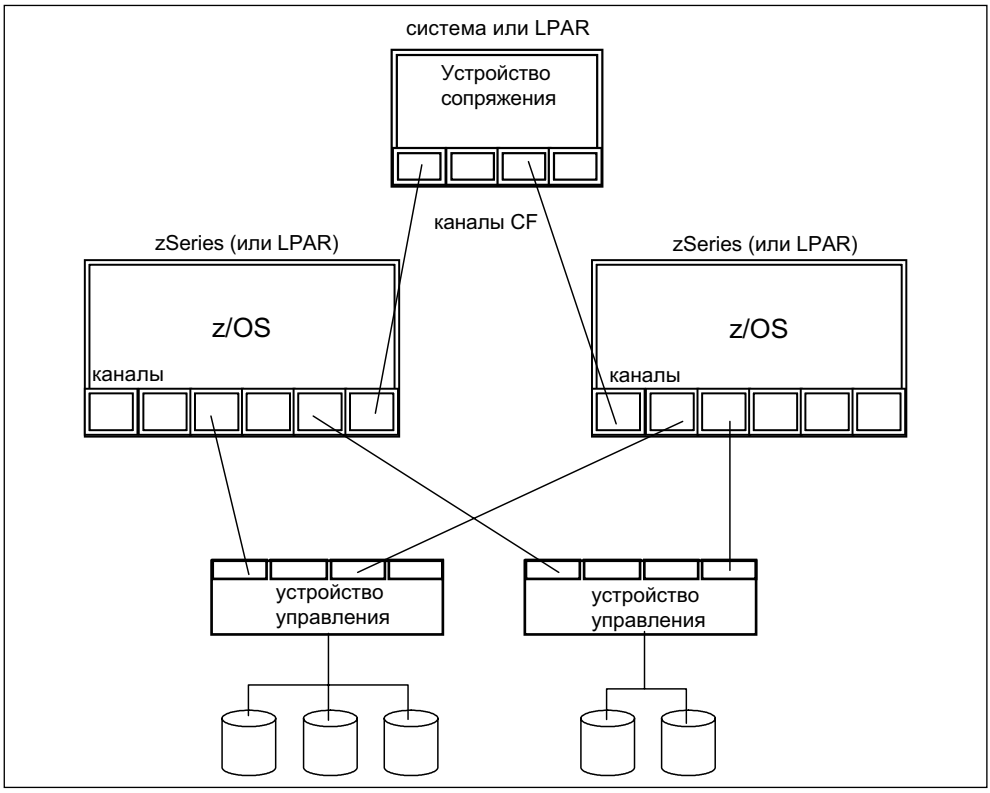


Рис. 2.12. Parallel Sysplex

[1] Операционная система CF совершенно не похожа на z/OS и не имеет прямых пользовательских интерфейсов.

Устройство сопряжения работает в значительной степени подобно быстрой разделяемой памяти. Оно используется для хранения следующей информации:

- информации о блокировках, совместно используемой всеми подключенными системами;
- информации кеша (например, для базы данных), совместно используемой всеми подключенными системами;
- информации списков данных, совместно используемой всеми подключенными системами.

Информация CF находится в памяти, и CF обычно имеет большую память. В качестве устройства сопряжения может использоваться отдельная система или логический раздел. На рис. 2.12 показан небольшой Parallel Sysplex с двумя образами z/OS. Опять же, эта конфигурация могла быть реализована на основе трех логических разделов одной системы, трех отдельных систем или смешанной комбинации.

Во многих отношениях система Parallel Sysplex работает как одна большая система. Она имеет единый интерфейс оператора (который управляет всеми системами). При правильном планировании и эксплуатации (обе эти задачи важны) сложные задачи могут разделяться между некоторыми или всеми системами в Parallel Sysplex и восстановление (с использованием другой системы в Parallel Sysplex) для многих задач может осуществляться автоматически.

2.8.2 Технологии кластеризации для мэйнфрейма

Технология Parallel Sysplex помогает обеспечить непрерывную доступность в современных средах больших систем. Parallel Sysplex позволяет выполнять связывание до 32 серверов с почти линейной масштабируемостью для создания мощной коммерческой вычислительной кластерной системы. Каждый сервер в кластере Parallel Sysplex может быть сконфигурирован для совместного доступа к ресурсам данных, и «клонированный» экземпляр приложения может выполняться на каждом сервере.

Конструктивные характеристики Parallel Sysplex позволяют предприятиям обеспечить непрерывную работу даже в периоды существенных изменений. Организации, использующие сисплекс, могут динамически добавлять и изменять системы в сисплексе и конфигурировать системы таким образом, чтобы они не содержали единых точек отказа.

При использовании этой последней кластерной технологии можно организовать совместную работу нескольких систем z/OS для более эффективной обработки крупнейших коммерческих задач.

Кластеризация общих данных

Технология Parallel Sysplex расширяет достоинства мэйнфрейм-компьютеров IBM, позволяя выполнять связывание до 32 серверов с почти линейной масштабируемостью для создания мощной коммерческой вычислительной кластерной системы. Каждый сервер в кластере Parallel Sysplex имеет доступ ко всем информационным ресурсам, и каждое «клонированное» приложение может выполняться на каждом сервере. Используя технологию сопряжения мэйнфреймов, Parallel Sysplex обеспечивает ме-

тод кластеризации «общих данных», позволяющий обеспечивать совместное использование данных несколькими системами с высокой производительностью и обеспечением целостности при чтении и записи.

Такой подход «общих данных» (в отличии от подхода «ничего общего») позволяет осуществлять динамическую балансировку задач между серверами в кластере Parallel Sysplex. Он позволяет критически важным бизнес-приложениям использовать суммарную мощность нескольких серверов для обеспечения максимальной пропускной способности и производительности системы в периоды пиковой нагрузки. В случае отключения аппаратного или программного обеспечения, как запланированного, так и незапланированного, задачи могут динамически перенаправляться на доступные серверы, обеспечивая почти непрерывную доступность приложений.

Оперативное обслуживание

Другим уникальным преимуществом использования технологии Parallel Sysplex является способность выполнять обслуживание и установку аппаратного и программного обеспечения оперативным образом.

Посредством совместного использования данных и динамического управления нагрузкой серверы могут динамически удаляться или добавляться в кластер, позволяя выполнять операции установки или обслуживания, тогда как остальные системы продолжают обработку. Кроме того, при соблюдении политики совместимости программного и аппаратного обеспечения компании IBM программные и аппаратные обновления могут применяться к системам поочередно. Эта способность позволяет клиентам выполнять внедрение изменений на системах с таким темпом, который приемлем для их предприятий.

Способность выполнять установку аппаратного обеспечения и обслуживание программного обеспечения оперативным образом позволяет предприятию реализовывать критические бизнес-функции и реагировать на быстрый рост без неблагоприятного воздействия на доступность.

2.9 Виды мэйнфрейм-систем

Рассмотрим три основных типа конфигурации. В этом разделе мы приводим не подробные описания, а только общие сведения.

2.9.1 Очень малые системы

Первые два примера, представленные на рис. 2.13, показывают, что термин *мэйнфрейм* больше относится к стилю вычислений, чем к типу аппаратного обеспечения. Показаны две разные системы, и ни одна из них не использует мэйнфрейм-оборудование в общепринятом смысле.

Первой показана система IBM Multiprise® 3000 (MP3000), которую IBM уже сняла с продажи на момент написания этой книги. Она представляла собой самую малую систему S/390, производимую в последние годы. MP3000 имеет один или два процессора S/390 и SAP-процессор. Она также имеет встроенные дисковые приводы, кото-

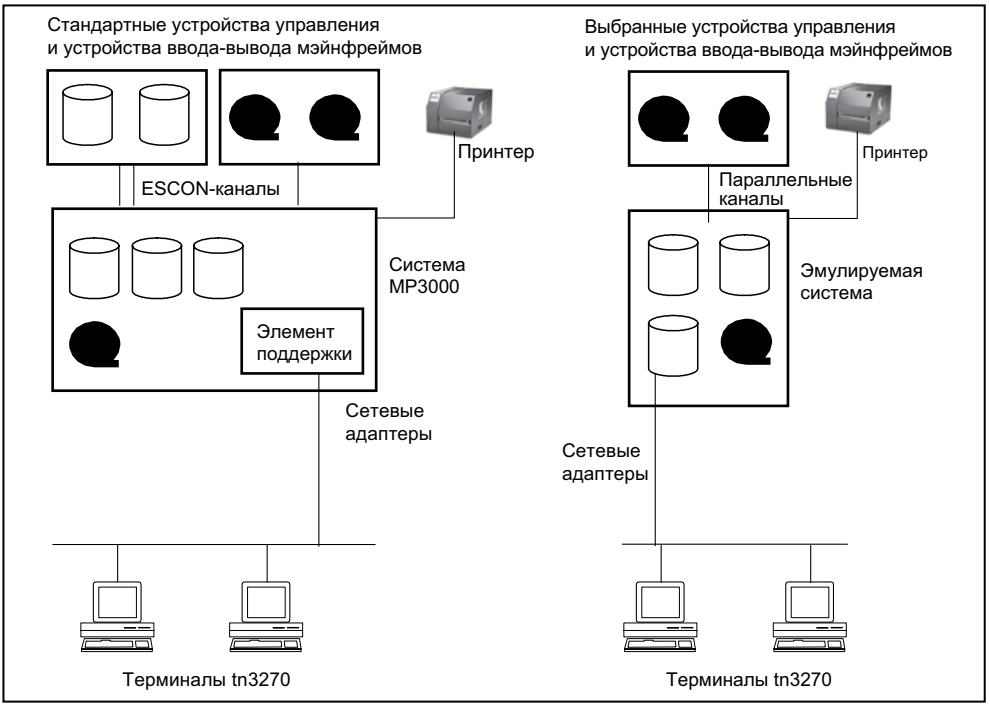


Рис. 2.13. Очень малые мейнфрейм-конфигурации

рые можно настроить на работу в режиме обычных дисковых приводов IBM 3390. Минимальный внутренний привод для магнитных лент обычно используется для установки программного обеспечения. MP3000 может иметь достаточное количество ESCON-каналов или параллельных каналов для подключения к традиционным внешним устройствам ввода-вывода.

Система MP3000 полностью совместима с мэйнфреймами S/390, но не имеет последних возможностей zSeries. Она может работать под управлением ранних версий z/OS и всех предыдущих версий операционных систем. Обычно она используется с операционными системами z/VM или z/VSE (которые вкратце описаны в разделе 1.10, «z/OS и прочие операционные системы мэйнфреймов»).

Вторая показанная система представляет собой эмуляцию системы zSeries и не имеет мэйнфрейм-оборудования. Она запущена на персональном компьютере (под управлением Linux или UNIX) и использует программное обеспечение для эмуляции z/OS. Специальные PCI-адаптеры каналов используются для подключения к требуемым устройствам ввода-вывода мэйнфреймов. Персональный компьютер с запущенным эмулятором z/OS может иметь достаточно внутренних дисков (обычно объединенных в RAID-массив) для эмуляции дисковых приводов IBM 3390.

Обеим системам недостает некоторых функций «настоящих» мэйнфреймов. Тем не менее обе они способны качественно выполнять свою работу. Приложения не смогут отличить эти системы от реальных мэйнфреймов. В действительности эти системы считаются мэйнфреймами, потому что их операционные системы, проме-

жуточное программное обеспечение, приложения и стиль использования соответствуют более крупным мэйнфреймам. В системе MP3000 можно создавать логические разделы и запускать тестовую и рабочую системы. Эмулируемая система не поддерживает логические разделы, но может вместо этого запускать несколько копий эмулятора.

Основная привлекательность этих систем состоит в том, что они являются «мэйнфреймом в одном корпусе». Во многих случаях не требуется использования внешних традиционных устройств ввода-вывода. Это значительно сокращает стоимость мэйнфрейм-системы начального уровня.

2.9.2 Средние одиночные системы

Рис. 2.14 иллюстрирует среднюю мэйнфрейм-систему и необходимые стандартные внешние элементы. В частности, представлена система IBM z890 с двумя современными внешними дисковыми контроллерами, несколькими приводами для магнитных лент, принтерами, подключениями к локальной сети и консолями.

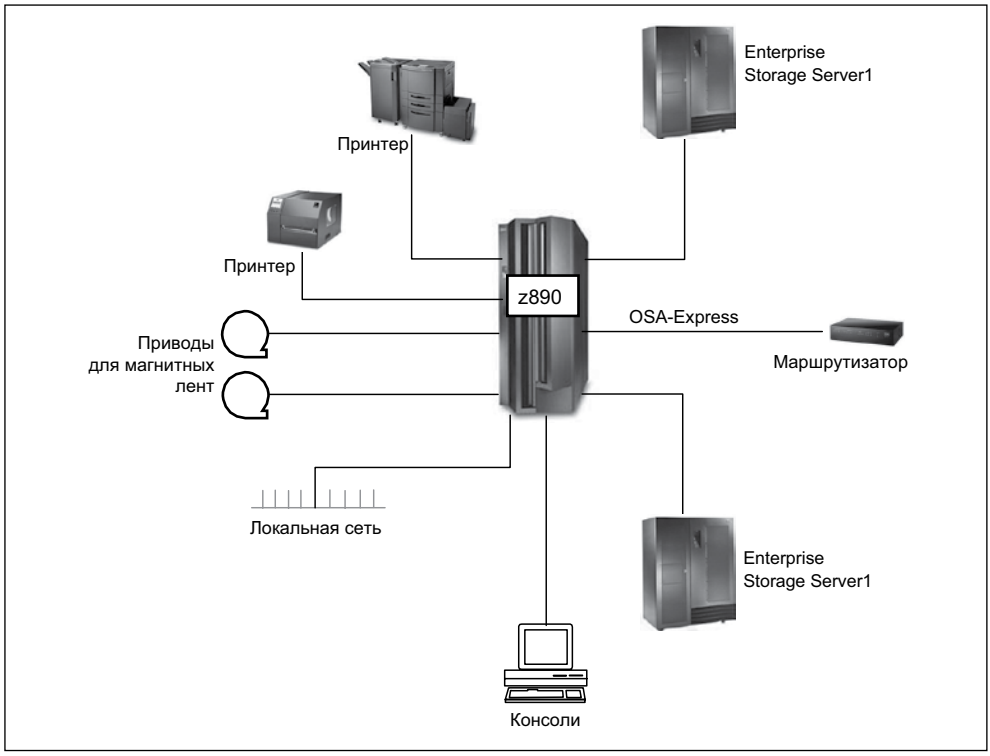


Рис. 2.14. Средняя мэйнфрейм-конфигурация

Это несколько идеализированная конфигурация, так как отсутствуют старые устройства. Описываемые здесь системы могут иметь несколько активных логических разделов, например:

- рабочую систему z/OS с запущенными интерактивными приложениями;

- вторую рабочую систему z/OS, предназначенную для крупных пакетных приложений (они также могут работать в первом LPAR, но в некоторых инсталляциях предпочтительнее использование отдельного LPAR в целях управления);
- тестовую систему z/OS для тестирования новых версий программного обеспечения, новых приложений и т. д;
- один или несколько Linux-разделов, возможно, выполняющих веб-приложения.

Дисковые контроллеры на рис. 2.14 содержат большое количество обычных приводов, работающих в различных RAID-конфигурациях. Устройство управления преобразует их интерфейсы таким образом, чтобы они воспринимались как стандартные дисковые приводы IBM 3390, которые являются самыми распространенными дисковыми устройствами для мэйнфреймов. Эти дисковые устройства управления имеют несколько канальных интерфейсов и могут работать параллельно.

2.9.3 Более крупные системы

На рис. 2.15 представлен более крупный мэйнфрейм, хотя такая конфигурация все же меньше *большой* мэйнфрейм-инсталляции. Этот пример является типичным, так как он включает как более старые, так и более новые мэйнфреймы, а также канальные

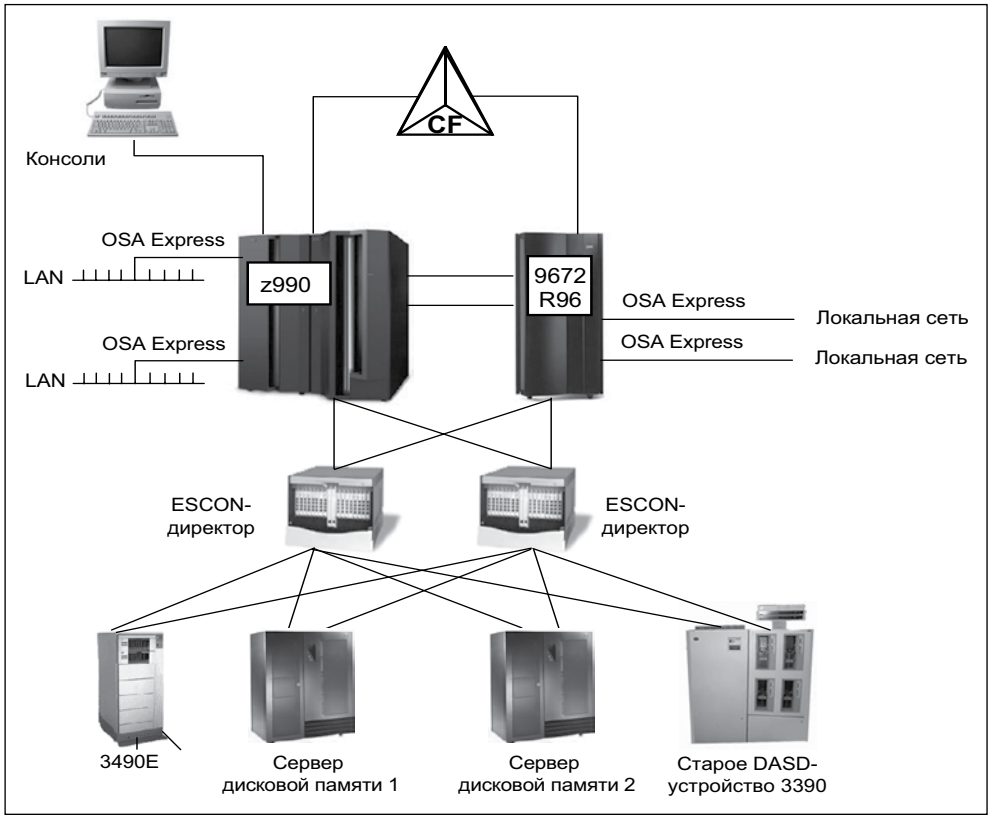


Рис. 2.15. Средняя по размеру мейнфрейм-конфигурация

коммутаторы, позволяющие всем системам осуществлять доступ к большинству устройств ввода-вывода. Подобным образом он включает новые и старые дисковые контроллеры (и устройства) и контроллеры (и устройства) приводов для магнитных лент. Вся система реализована в виде средней конфигурации Parallel Sysplex.

Если вкратце, на рис. 2.15 представлены следующие устройства:

- IBM 3745 – коммуникационный контроллер, оптимизированный для связи с удаленными терминалами и контроллерами и с локальными сетями. С точки зрения мэйнфрейма 3745 является устройством управления.
- Приводы для магнитных лент IBM 3490E, которые, хотя и несколько устарели, могут работать с большинством наиболее часто используемых мэйнфрейм-совместимых картриджей.
- Мэйнфрейм шестого поколения (G6).
- Новый мэйнфрейм z990.
- Сервер дисковой памяти (Enterprise Storage Server, ESS).
- ESCON-директоры.
- Подключения OSA Express к нескольким локальным сетям.
- CF (представленный отдельным блоком, однако может быть логическим разделом мэйнфрейма).

2.10 Непрерывная доступность мэйнфреймов

Parallel Sysplex является высокоэффективной технологией, позволяющей высоконадежным, избыточным и устойчивым мэйнфрейм-технологиям достичь почти непрерывной доступности. Правильно настроенный кластер Parallel Sysplex может оставаться доступным для пользователей и приложений с минимальными простоями.

- Аппаратные и программные компоненты предусматривают распараллеливание для обеспечения оперативного обслуживания, например для повышения производительности по запросу (Capacity Upgrade on Demand), при которой осуществляется постепенное добавление процессоров или устройств сопряжения без нарушения выполнения текущих задач.
- Подсистемы DASD используют зеркальное отображение дисков или RAID-технологии для защиты от потери данных, а также технологии, позволяющие осуществлять мгновенное резервное копирование без завершения работы приложений.
- Сетевые технологии включают такие функции, как универсальные ресурсы VTAM® (VTAM® Generic Resources), многоузловые устойчивые сеансы (Multi-Node Persistent Sessions), виртуальная IP-адресация (Virtual IP Addressing) и устройство распределения (Sysplex Distributor) для обеспечения отказоустойчивых сетевых подключений.
- Подсистемы ввода-вывода, поддерживающие несколько путей ввода-вывода и динамическую коммутацию для недопущения потери доступа к данным и повышения пропускной способности.

- Программные компоненты z/OS позволяют новым версиям программного обеспечения сосуществовать с более ранними версиями этих программных компонентов для упрощения развертывания.
- Бизнес-приложения поддерживают совместное использование данных и клонируются на различных серверах, что позволяет осуществлять балансировку рабочей нагрузки во избежание потери доступности приложения при сбоях.
- Процессы эксплуатации и восстановления полностью автоматизированы и прозрачны для пользователей, что сокращает или полностью устраняет необходимость вмешательства человека.

Parallel Sysplex представляет собой способ управления такой многосистемной средой, имеющий следующие преимущества:

- отсутствие единых точек отказа;
- мощность и масштабирование;
- динамическая балансировка нагрузки;
- простота в использовании;
- единый образ системы;
- совместимость изменений и рост без нарушения работы;
- совместимость приложений;
- сварийное восстановление.

Эти преимущества более подробно рассматриваются в последующих разделах этой главы.

2.10.1 Отсутствие единых точек отказа

В кластере Parallel Sysplex можно построить среду параллельной обработки без единых точек отказа. Так как все системы в кластере Parallel Sysplex могут иметь одновременный доступ ко всем критически важным приложениям и данным, отключение системы вследствие аппаратного или программного сбоя не обязательно вызывает потерю доступности приложения.

Равноправные экземпляры отказавшей подсистемы, выполняющиеся на оставшихся исправных узлах системы, могут взять на себя функции восстановления ресурсов, занимаемых отказавшим экземпляром. С другой стороны, отказавшая подсистема может быть автоматически перезапущена на исправных системах с использованием возможностей автоматического перезапуска для выполнения восстановления заданий, выполнявшихся на момент сбоя. Хотя отказавший экземпляр подсистемы недоступен, новые рабочие запросы могут быть перенаправлены на другие экземпляры подсистемы, осуществляющие совместный доступ к данным, на других узлах кластера для обеспечения непрерывной доступности приложений на время отказа и последующего восстановления. Это позволяет скрыть запланированные и незапланированные остановки для конечного пользователя.

Из-за избыточности конфигурации имеет место значительное сокращение количества единых точек отказа. Если Parallel Sysplex не используется, отказ сервера может серьезно повлиять на производительность приложения, а также вызвать трудности

в управлении системами при перераспределении нагрузки или выделении ресурсов до устранения отказа. В среде Parallel Sysplex отказ сервера может быть незаметен для приложения и нагрузка сервера может быть автоматически перераспределена в Parallel Sysplex с небольшим снижением производительности. Таким образом, события, которые в противном случае серьезно бы повлияли на доступность приложений, например отказы аппаратных элементов центрального процессорного комплекса (CPC) или критически важных компонентов операционной системы, в среде Parallel Sysplex оказывают незначительное воздействие.

Несмотря на то что узлы в кластере Parallel Sysplex работают совместно и представляют единый образ, они остаются отдельными системами, обеспечивающими непрерывность установки, эксплуатации и обслуживания. Системный программист может вносить изменения, например устанавливать обновления программного обеспечения, на одной системе зараз, тогда как остальные системы будут продолжать обработку. Это позволяет персоналу по обслуживанию мэйнфрейма выполнять развертывание изменений на своих системах в соответствии с удобным для предприятия графиком.

2.10.2 Мощность и масштабирование

Среда Parallel Sysplex допускает практически линейное масштабирование от 2 до 32 систем. Она может состоять из любых серверов, поддерживающих среду Parallel Sysplex. Суммарная мощность такой конфигурации достаточна для выполнения любых современных задач обработки.

2.10.3 Динамическая балансировка нагрузки

Для конечных пользователей и бизнес-приложений весь кластер Parallel Sysplex может представляться как единый логический ресурс. Подобно тому как обработка может динамически распределяться между отдельными процессорами на одном SMP-сервере, возможно перенаправление обработки на любой узел в кластере Parallel Sysplex, имеющий доступную мощность. Это устраняет необходимость разделения данных или приложений между отдельными узлами в кластере или репликации базы данных между несколькими серверами.

Балансировка нагрузки также позволяет предприятию выполнять разнородные приложения в кластере Parallel Sysplex, обеспечивая требуемую скорость реагирования. ИТ-директор по работе с мэйнфреймами определяет для каждого задания соответствующее соглашение об уровне сервиса, и компонент управления рабочей нагрузкой (workload management, WLM) в z/OS вместе с такими подсистемами, как CP/SM или IMS, выполняет автоматическую балансировку задач между всеми ресурсами кластера Parallel Sysplex для достижения поставленных бизнес-целей. Нагрузка может иметь множество источников, например пакетные задания, SNA, TCP/IP, DRDA® или WebSphere MQ.

Необходимо рассмотреть некоторые аспекты, связанные с восстановлением. Во-первых, при возникновении отказа важно его обойти путем автоматического перераспределения рабочей нагрузки для использования оставшихся доступных ресурсов.

Во-вторых, необходимо выполнить восстановление элементов заданий, обрабатывавшихся в момент отказа. И наконец, после восстановления отказавшего элемента его следует возвратить в конфигурацию максимально быстро и незаметно, чтобы снова начать обработку рабочей нагрузки. С технологией Parallel Sysplex все это возможно.

Распределение рабочей нагрузки

После изоляции отказавшего элемента необходимо перенаправить рабочую нагрузку на оставшиеся доступные ресурсы в кластере Parallel Sysplex без нарушения работы. В случае отказа в среде Parallel Sysplex рабочая нагрузка, связанная с оперативными транзакциями, перераспределяется автоматически без вмешательства оператора.

Управление универсальными ресурсами

Управление универсальными ресурсами обеспечивает возможность назначения общего сетевого интерфейса для VTAM. Его можно использовать с регионами-владельцами терминалов (terminal owning regions, TOR) в CICS, диспетчером транзакций IMS (IMS Transaction Manager), TSO или задачами DB2 DDF. Отказ одного из CICS TOR, например, влияет только на часть сети. Затрагиваемые терминалы могут сразу же повторно подключиться и продолжать обработку после подключения к другому TOR.

2.10.4 Простота в использовании

Система Parallel Sysplex удовлетворяет основным требованиям клиентов по обеспечению непрерывной круглосуточной доступности и включает методы осуществления упрощенного управления системами, соответствующие этому требованию. Некоторые возможности системы Parallel Sysplex, обеспечивающие высокую доступность, помогают также устранить некоторые задачи управления системами, например:

- компонент управления рабочей нагрузкой (WLM);
- менеджер отказов сисплекса (SFM);
- менеджер автоматического перезапуска (ARM);
- клонирование и символы;
- совместное использование ресурсов zSeries.

Компонент управления рабочей нагрузкой (WLM)

Компонент управления рабочей нагрузкой (Workload Management, WLM) операционной системы z/OS обеспечивает возможности управления рабочей нагрузкой в масштабе сисплекса на основе целей производительности заданной инсталляции и важности задач для предприятия. WLM пытается достичь целей производительности посредством динамического распределения ресурсов. WLM обеспечивает кластер Parallel Sysplex возможностями определения того, где и с каким приоритетом следует выполнять обработку задач. Приоритет определяется исходя из бизнес-целей клиента и управляется сисплекс-технологией.

Менеджер отказов сисплекса (SFM)

Политика управления отказами сисплекса (Sysplex Failure Management) позволяет инсталляции определять интервалы обнаружения отказов и операции восстановления, инициируемые в случае отказа системы в сисплексе.

Если SFM не используется, тогда при отказе одной из систем в Parallel Sysplex оператору выдается уведомление и предлагается выбрать операцию восстановления. Оператор может выбрать отделение неотвечающей системы от Parallel Sysplex или предпринять некоторые действия для восстановления системы. На период вмешательства оператора могут быть заблокированы критически важные системные ресурсы, нужные остальным активным системам. Менеджер отказов сисплекса позволяет инсталляции составить политику определения операций восстановления, автоматически инициируемых при обнаружении некоторых типов проблем, в частности изоляции отказавшего образа, блокирующего доступ к совместно используемым ресурсам, деактивации логического раздела или перераспределение центральной и расширенной памяти.

Менеджер автоматического перезапуска (ARM)

Менеджер автоматического перезапуска (Automatic Restart Manager) позволяет осуществлять быстрое восстановление подсистем, которые могут удерживать критические ресурсы при отказе. Если другим экземплярам подсистемы в Parallel Sysplex требуется доступ к этим критическим ресурсам, быстрое восстановление ускорит возобновление доступа к этим ресурсам. Несмотря на то что в настоящее время существуют автоматизированные пакеты для перезапуска подсистемы, позволяющие разрешать проблемы со взаимной блокировкой, ARM активируется быстрее после отказа.

ARM – функция восстановления системы, повышающая доступность пакетных заданий и запускаемых задач

ARM уменьшает необходимость вмешательства оператора для выполнения следующих задач:

- Обнаружение отказа критического задания или запускаемой задачи.
- Автоматический перезапуск после отказа запускаемой задачи или задания. После аварийного завершения задания или запускаемой задачи это задание или запускаемая задача может быть перезапущено с определенными условиями, например с замещением оригинального JCL или с определением зависимостей заданий без вмешательства оператора.
- Автоматическое переназначение заданий подходящей системе после отказа. Это устраняет длительный этап выбора оператором наиболее подходящей целевой системы для перезапуска заданий

Клонирование и символы

Под *клонированием* подразумевается репликация аппаратных и программных конфигураций между различными физическими серверами в Parallel Sysplex. Другими словами, приложение, для которого планируется осуществлять параллельную обработку, может иметь несколько одинаковых экземпляров, выполняющихся на всех об-

разах в Parallel Sysplex. Аппаратное и программное обеспечение, поддерживающее эти приложения, также может быть одинаково настроено на всех системах в Parallel Sysplex, чтобы уменьшить объем работы по определению и поддержке среды.

Понятие *симметрии* позволяет внедрять новые системы и обеспечивает автоматическое распределение рабочей нагрузки в случае отказа или при запланированном обслуживании отдельной системы. При этом также сокращается объем работы по настройке среды системным программистом. Обратите внимание на то, что симметрия *не* исключает возможности уникальной настройки конфигурации отдельных систем, в частности асимметричного подключения принтеров и коммуникационных контроллеров или обработки асимметричной нагрузки, не приспособленной к параллельной среде.

При управлении клонированием используется системные символы; z/OS обеспечивает поддержку подстановки значений в параметрах запуска, JCL, системных командах и запускаемых задачах. Эти значения можно использовать при определении параметров и процедур, чтобы обеспечить уникальную подстановку при динамическом составлении имени ресурса.

Совместное использование ресурсов zSeries

Многие базовые компоненты z/OS используют общую память устройства сопряжения IBM для обмена данными компонентов в целях управления ресурсами нескольких систем. Такое использование, называемое совместным использованием ресурсов zSeries (zSeries Resource Sharing), обеспечивает совместное использование физических ресурсов, таких, как файлы, приводы для магнитных лент, консоли и каталоги с уменьшением стоимости, повышением производительности и упрощением управления системами. *Не следует путать* это с совместным использованием данных в Parallel Sysplex подсистемами баз данных. Совместное использование ресурсов zSeries работает даже у клиентов, не применяющих совместное использование данных посредством его поддержки системой на уровне базового программного стека z/OS.

Одна из целей Parallel Sysplex состоит в обеспечении упрощенного управления системами путем уменьшения сложности в управлении, эксплуатации и обслуживании Parallel Sysplex без увеличения персонала поддержки и без снижения доступности.

2.10.5 Единый образ системы

Несмотря на то что Parallel Sysplex может содержать несколько серверов и образов z/OS, а также множество различных технологий, набор систем, входящих в Parallel Sysplex, должен выглядеть как единый объект с точки зрения оператора, конечного пользователя, администратора базы данных и т. д. Единый образ системы уменьшает сложность с точки зрения эксплуатации и определений.

Независимо от количества образов системы и сложности базового оборудования, система Parallel Sysplex обеспечивает представление единого образа системы в следующих областях:

- доступ к данным, обеспечивая динамическую балансировку нагрузки и более высокую доступность;

- динамическая маршрутизация транзакций, обеспечивая динамическую балансировку нагрузки и более высокую доступность;
- интерфейс конечного пользователя, обеспечивая подключение к логическому сетевому объекту;
- рперационные интерфейсы, упрощающие управление системами.

Единый пункт управления

Необходимо, чтобы набором систем, входящих в Parallel Sysplex, можно было управлять с логического единого пункта управления. Термин «единый пункт управления» означает способность осуществлять доступ к любым интерфейсам, необходимым для рассматриваемой задачи, без привязки к определенному физическому компоненту оборудования. Например, в кластере Parallel Sysplex из нескольких систем необходимо иметь возможность направлять команды или операции на любую систему кластера без физического подключения консоли или пункта управления к каждой системе в Parallel Sysplex.

Единый пункт управления – свойство сисплекса; способность выполнять заданный набор задач с единой рабочей станции

Устойчивость единого образа системы к отказам

В случае отказа отдельных элементов оборудования или целых систем, входящих в Parallel Sysplex, единый образ системы должен быть сохранен. Это означает, что, как и в случае с единым пунктом управления, представление единого образа системы не зависит от определенного физического элемента в конфигурации. С точки зрения конечного пользователя параллельность выполнения приложений в среде Parallel Sysplex должна быть незаметной. Приложение должно быть доступным независимо от того, какой физический образ z/OS его поддерживает.

2.10.6 Совместимость изменений и рост без нарушения работы

Основное назначение Parallel Sysplex состоит в обеспечения непрерывной доступности. Поэтому необходимо, чтобы изменения, в частности новые приложения, программное или аппаратное обеспечение, можно было установить без нарушения работы и чтобы они были совместимы с текущим уровнем программного и аппаратного обеспечения. Для поддержки совместимости изменений аппаратные и программные компоненты кластера Parallel Sysplex допускают сосуществование двух уровней, т. е. уровня N и уровня N+1. Это означает, например, что ни в один программный продукт компании IBM нельзя внести изменение, которое бы не могло совместно работать с неизмененной версией.

2.10.7 Совместимость приложений

С точки зрения устройства цель кластеров Parallel Sysplex состоит в том, чтобы для их использования не требовалось вносить изменения в приложения. В большинстве случаев это требование было выполнено, хотя для того, чтобы получить максимальную пользу от конфигурации, необходимо исследовать некоторые свойства.

С точки зрения разработчика архитектуры приложения существует три основные причины для запуска приложения в кластере Parallel Sysplex:

- Технологические преимущества.
  Масштабируемость (даже при обновлениях без нарушения работы), доступность и динамическое управление нагрузкой являются теми средствами, которые позволяют разработчику архитектуры обеспечить соответствие требованиям клиента в тех случаях, когда приложение играет ключевую роль в бизнес-процессе клиента. При применении технологии совместного использования данных несколькими системами все узлы обработки в кластере Parallel Sysplex имеют одновременный доступ для чтения и записи общих данных без неблагоприятного воздействия на целостность и производительность.
- Интеграционные преимущества.
  Так как многие приложения традиционно основаны на S/390 и z/OS, новые приложения для z/OS имеют преимущества с точки зрения производительности и обслуживания, особенно если они связаны с существующими приложениями.
- Инфраструктурные преимущества.
  При использовании Parallel Sysplex для интеграции нового приложения не требуется значительное изменение инфраструктуры. Во многих случаях инсталляция не требует интеграции новых серверов. Вместо этого она может использовать существующую инфраструктуру и преимущества существующего сисплекса. При использовании Geographically Dispersed Parallel Sysplex™ (GDPS®), связывающего несколько сисплексов, расположенных в разных местах, ИТ-персонал по обслуживанию мэйнфрейма может создать конфигурацию, допускающую аварийное восстановление.

2.10.8 Аварийное восстановление

Geographically Dispersed Parallel Sysplex (GDPS) представляет собой основную систему обеспечения аварийного восстановления и непрерывной доступности для географически распределенного предприятия, использующего мэйнфреймы. GDPS автоматически осуществляет зеркальное копирование критически важных данных и эффективную балансировку нагрузки между вычислительными центрами.

GDPS – приложение, повышающее доступность приложений и обеспечивающее аварийное восстановление в Parallel Sysplex

GDPS также использует автоматизацию и технологию Parallel Sysplex для управления базами данных, процессорами, сетевыми ресурсами и зеркальным отображением подсистем хранения в географически распределенном предприятии. Эта технология обеспечивает непрерывную доступность, эффективный перенос нагрузки между вычислительными центрами, управление ресурсами и быстрое восстановление данных для критически важных мэйнфрейм-приложений. В настоящее время максимально допустимое расстояние между двумя центрами в GDPS составляет 100 км (около 62 миль) оптоволоконного провода, хотя существуют некоторые другие ограничения. Это обеспечивает синхронную систему, позволяющую обеспечить недопущение потери данных.

Существует также технология GDPS/XRC, которую можно использовать на сверхбольших расстояниях и которая обеспечивает восстановление менее чем за 2 минуты (другими словами, требуется не больше 2 минут для восстановления данных).

2.11 Заключение

Знание различных значений терминов *система*, *процессор*, *CP* и т. д. является важным для понимания мэйнфрейм-компьютеров. Знание первоначальной архитектуры S/360, использовавшей центральные процессоры, память, каналы, устройства управления и устройства ввода-вывода, а также способа их адресации является чрезвычайно важным для понимания аппаратного обеспечения мэйнфрейма, несмотря на то что почти каждая деталь изначального дизайна была изменена. Понятия и терминология изначального дизайна все еще используются в описаниях и структуре мэйнфреймов.

Возможность разделения большой системы на несколько менее крупных систем (логических разделов, LPAR) в настоящее время является основным требованием практически во всех мэйнфрейм-инсталляциях. Гибкость устройства аппаратного обеспечения, позволяющая любому процессору (CP) осуществлять доступ и принимать прерывания для любого канала, устройства управления и устройства ввода-вывода, подключенного к заданному LPAR, обеспечивает гибкость, надежность и высокую производительность всей системы. Доступность пула процессоров (PU), которые могут быть настроены (в IBM) в качестве клиентских процессоров (CP), вспомогательных системных процессоров (SAP), выделенных Linux-процессоров (IFL), выделенных Java-процессоров (zAAP) и резервных процессоров является уникальной особенностью мэйнфреймов и опять же обеспечивает высокую гибкость в обеспечении соответствия требованиям клиентов. Некоторые из этих требований основаны на структуре стоимости некоторых программных продуктов для мэйнфреймов.

zAAP – специализированный вспомогательный процессорный модуль, настроенный на выполнение Java-программ на компьютерах zSeries

Помимо основных описанных процессоров (процессорных модулей и всех их специализаций), мэйнфреймы имеют сеть контроллеров (специальных микропроцессоров), управляющих системой в целом. Эти контроллеры невидимы для операционной системы и приложений.

С начала 1970-х годов мэйнфреймы разрабатывались как многопроцессорные системы, даже если устанавливался только один процессор. Все программное обеспечение операционных систем разрабатывалось для использования нескольких процессоров; система с одним процессором считается особым случаем общего многопроцессорного дизайна. Все мэйнфрейм-инсталляции, за исключением самых малых, обычно применяют технологии кластеризации, хотя при этом обычно не используются термины *кластер* или *кластеризация*.

Как говорилось выше, технология кластеризации может представлять простую конфигурацию с общим DASD, при которой для недопущения нежелательного наложения данных требуется ручное управление или планирование. Однако в настоящее время чаще используются конфигурации, обеспечивающие общее использование

данных об управлении блокировками и очередями для всех систем. Помимо остальных преимуществ при этом осуществляется автоматическое управление доступом к наборам данных, что позволяет избежать нежелательного одновременного использования данных.

Самой современной кластерной технологией является технология Parallel Sysplex. Она позволяет соединять до 32 серверов с почти линейной масштабируемостью для создания мощной коммерческой кластерной системы обработки. Каждый сервер в кластере Parallel Sysplex имеет доступ ко всем информационным ресурсам, и каждое «клонированное» приложение может выполняться на каждом сервере. При использовании с технологией сопряжения Parallel Sysplex обеспечивает совместное использование данных, которое позволяет осуществлять обмен данными между несколькими системами с высокой производительностью операций чтения и записи и поддержкой целостности. Конструктивные характеристики Parallel Sysplex позволяют предприятиям обеспечить непрерывную работу даже в периоды существенных изменений. Организации, использующие сисплексы, могут динамически добавлять и изменять системы в сисплексе и конфигурировать системы таким образом, чтобы они не содержали единых точек отказа.

При использовании этой последней кластерной технологии можно организовать совместную работу нескольких систем z/OS для более эффективной обработки крупнейших коммерческих задач.

Основные термины в этой главе		
Менеджер автоматического перезапуска (ARM)	Центральный процессорный комплекс (CPC)	Центральный процессор (CPU)
Идентификатор канального пути (CHPID)	Соединение канал-канал (CTC)	Устройство сопряжения
ESCON-канал	Geographically Dispersed Parallel Sysplex (GDPS)	Консоль управления аппаратными средствами (HMC)
Логический раздел (LPAR)	Мультипроцессор	Parallel Sysplex
Единый пункт управления	z/Architecture	zSeries Application Assist Processor (zAAP).

2.12 Контрольные вопросы

Чтобы проверить понимание материала этой главы, ответьте на следующие вопросы:

1. Почему расчет цен на программное обеспечение для мэйнфреймов является настолько сложным?
2. Почему IBM имеет так много моделей (или «параметров мощности») в последних мэйнфрейм-компьютерах?
3. Почему вычислительная мощность, необходимая для работы традиционного COBOL-приложения не имеет линейной связи с мощностью, необходимой для работы нового Java-приложения?

4. Термин *мультипроцессор* указывает на наличие нескольких процессоров (и на то, что эти процессоры используются операционной системой и приложениями). Что означает термин *мультипрограммирование*?
5. Какое различие между слабосвязанными системами и сильносвязанными системами?
6. Какие изменения в приложении z/OS нужно сделать, чтобы оно работало в LPAR?

2.13 Темы для дальнейшего обсуждения

Посетите организацию, в которой установлен мэйнфрейм, если это возможно. Список новых, более старых и совсем старых систем и устройств, используемых в типичной инсталляции, обычно представляет интерес и помогает почувствовать атмосферу непрерывности работы, которая очень важна для клиентов, использующих мейнфреймы.

1. Каковы преимущества кластера Parallel Sysplex, обеспечивающего внешнее представление единого образа? Существуют ли недостатки?
2. Почему на современном рынке необходимо обеспечивать непрерывную доступность?
3. Как можно обосновать стоимость «избыточного» оборудования и стоимость лицензий на программное обеспечение, необходимых для построения кластера Parallel Sysplex?

2.14 Упражнения

1. Для вывода конфигурации CPU:
   а) вызовите SDSF из главного меню ISPF;
   б) в поле ввода команд введите `/D M=CPU` и нажмите Enter;
   в) используйте опцию ULOG в SDSF для просмотра результатов выполнения команды.
2. Для вывода данных об использовании страничного набора данных:
   а) в поле ввода команд введите `/D ASM` и нажмите Enter;
   б) нажмите PF3 для возврата в предыдущие экраны.

3

Общие сведения о z/OS

Цель. Как новый член отдела информационных технологий компании, использующего мэйнфрейм, вам нужно знать основные функциональные характеристики операционной системы мэйнфрейма. В этой книге изучается z/OS – широко используемая операционная система для мэйнфреймов, которая известна своей способностью одновременно обслуживать тысячи пользователей и обрабатывать очень большую нагрузку безопасным, надежным и наиболее подходящим способом.

После завершения работы над этой главой вы сможете:

- назвать несколько определяющих характеристик операционной системы z/OS;
- привести примеры отличий z/OS от однопользовательской операционной системы;
- перечислить основные типы памяти, используемой в z/OS;
- описать понятие виртуальной памяти и ее использование в z/OS;
- описать связь между страницами, фреймами и слотами;
- назвать несколько программных продуктов, используемых в z/OS для обеспечения полнофункциональной системы;
- перечислить несколько сходств и различий между операционными системами z/OS и UNIX.

3.1 Что такое операционная система?

Попросту говоря, *операционная система* представляет собой набор программ, управляющих внутренней работой компьютерной системы. Операционные системы предназначены для обеспечения наилучшего использования различных ресурсов компьютера и достижения наибольшей эффективности обработки максимального объема работы. Хотя операционная система не может увеличить скорость компьютера, она может максимизировать его использование, в результате чего компьютер может сделать больше работы за определенный период времени и возникает ощущение, как будто он работает быстрее.

Архитектура компьютера состоит из функций, которые обеспечивает компьютерная система. Понятие архитектуры не равнозначно понятию физического устройства, и фактически машины различного устройства могут относиться к одной компьютерной архитектуре. В некотором смысле под архитектурой понимается то, как компьютер представляется пользователю, например системному программисту. Например, к архитектуре относится набор машинных инструкций, которые компьютер может распознавать и выполнять. В среде мэйнфреймов программное и аппаратное обеспечение системы составляет очень прогрессивную компьютерную архитектуру, являющую собой результат десятилетий технологических инноваций.

3.2 Что такое z/OS?

В этой книге изучается z/OS[1] – широко используемая операционная система для мэйнфреймов.

z/OS предназначена для обеспечения стабильной, безопасной и постоянно доступной среды для приложений, выполняемых на мэйнфрейме.

В настоящее время операционная система z/OS являет собой результат десятилетий технологических инноваций. Она прошла путь от операционной системы, способной одновременно обрабатывать только одну программу, до операционной системы, которая может параллельно обрабатывать тысячи программ и интерактивных пользователей. Чтобы понять, как и почему z/OS работает свойственным ей образом, важно иметь представление о некоторых основных понятиях z/OS и среды, в которой она работает. Эта глава описывает некоторые понятия, которые вам необходимо знать для понимания операционной системы z/OS.

В большинстве ранних операционных систем запросы на обработку вводились в систему поочередно. Операционная система обрабатывала каждый запрос или *задание* как единое целое и не запускала следующее задание до тех пор, пока не завершится обработка текущего. Такая схема хорошо работала, когда задание могло непрерывно выполняться от начала до конца. Но часто заданию нужно было ожидать завершения ввода или вывода информации при работе с такими устройствами, как привод для магнитных лент или принтер. Ввод и вывод занимает много времени в сравнении со скоростью процессора. Когда задание ожидало завершения операций ввода-вывода, процессор бездействовал.

[1] z/OS разработана таким образом, чтобы использовать архитектуру IBM zSeries или z/Architecture, которая появилась в 2000 году.

Нахождение способа продолжить работу процессора в то время, пока задание находилось в режиме ожидания, позволило бы увеличить общий объем нагрузки, которую процессор мог обработать без использования дополнительного оборудования; z/OS выполняет работу путем ее разделения на фрагменты и передачи фрагментов задания различным компонентам системы и подсистемам, работающим независимо. В любой момент времени тот или иной компонент принимает управление процессором, выполняет свою часть работы и передает управление пользовательской программе или другому компоненту.

3.2.1 Аппаратные ресурсы, используемые в z/OS

Операционная система z/OS выполняется процессором и во время выполнения находится в памяти процессора; z/OS часто называется *системным* программным обеспечением.

Оборудование мэйнфрейма состоит из процессоров и множества периферийных устройств, таких, как дисковые приводы (называемые устройствами хранения с прямым доступом – direct access storage devices, DASD), приводами для магнитных лент и различными типами пользовательских консолей (рис. 3.1). Приводы для магнитных лент и DASD используются для системных функций и пользовательскими программами, выполняемыми в z/OS.

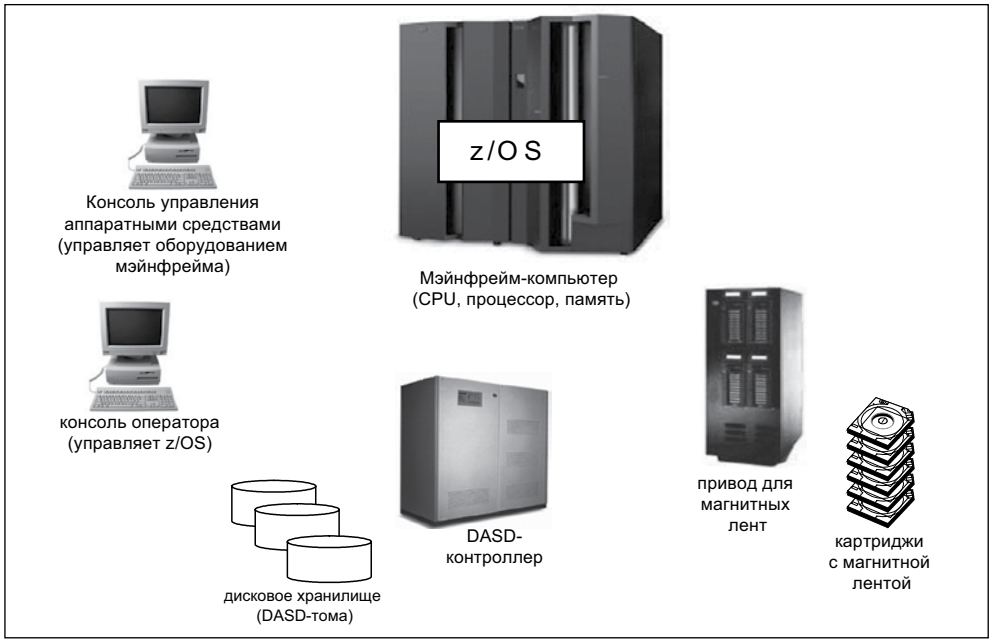


Рис. 3.1. Аппаратные ресурсы, используемые в z/OS

При выполнении нового заказа на систему z/OS IBM поставляет код системы клиенту через Интернет или (в зависимости от желания клиента) на картриджах с магнитной лентой. В организации клиента системный программист z/OS получает заказ

и копирует новую систему на DASD-тома. После того как система настроена и готова к работе, для запуска и эксплуатации системы z/OS необходимы системные консоли.

Операционная система z/OS предназначена для обеспечения полного использования самого современного оборудования IBM и множества сложных периферийных устройств. На рис. 3.1 приведено упрощенное представление концепции мэйнфрейма, используемой нами в этой книге:

- Программное обеспечение – операционная система z/OS состоит из загрузочных модулей или *исполняемого кода*. В процессе установки системный программист копирует эти загрузочные модули в загрузочные библиотеки, расположенные на DASD-томах.
- Аппаратное обеспечение – оборудование системы включает все устройства, контроллеры и процессоры, составляющие среду мэйнфрейма.
- Периферийные устройства – включают приводы для магнитных лент, DASD и консоли. Существует множество других типов устройств, некоторые из которых обсуждались в главе 2, «Аппаратные системы мэйнфрейма и высокая доступность».
- Память процессора (processor storage) – часто называется основной, реальной (real) или центральной (central) памятью; в ней происходит выполнение операционной системы z/OS. Кроме того, все пользовательские программы используют память процессора совместно с операционной системой.

Являясь «общим представлением» типичной аппаратной конфигурации мэйнфрейма, рис. 3.1 далек от полноты. Не показаны, например, аппаратные устройства управления, соединяющие мэйнфрейм с другими приводами для магнитных лентах, DASD и консолями.

Дополнительные сведения. Стандартным справочником, описывающим основные средства z/Architecture, является публикация *z/Architecture Principles of Operation*. Эту и другие публикации IBM можно найти на веб-сайте z/OS Internet Library

`http://www.ibm.com/servers/eserver/zseries/zos/bkserv/`

3.2.2 Мультипрограммирование и мультипроцессирование

Ранние операционные системы использовались для управления однопользовательскими компьютерными системами. В те времена операционная система считывала одно задание, находила данные и устройства, требуемые заданию, продолжала выполнение задания до завершения, после чего считывала другое задание. В отличие от ранних систем компьютерные системы, управляемые операционной системой z/OS, поддерживают *мультипрограммирование*, т. е. одновременное выполнение множества программ.

Мультипрограммирование – одновременное выполнение множества программ

При мультипрограммировании, когда задание не может использовать процессор, система может приостановить или прервать задание, освобождая процессор для обработки другого задания.

Для того чтобы сделать мультипрограммирование возможным, z/OS собирает и сохраняет всю необходимую информацию о прерываемой программе, прежде чем перейти к выполнению другой программы. Когда прерванная программа опять готова к выполнению, она может возобновить выполнение с того места, в котором она была остановлена.

Мультипрограммирование позволяет z/OS одновременно выполнять тысячи программ для пользователей, работающих над различными проектами в разных точках мира.

z/OS может также осуществлять *мультипроцессирование*, под которым понимается одновременная работа двух и больше процессоров, совместно использующих различные аппаратные ресурсы, в частности память и внешние дисковые устройства. Технологии мультипрограммирования и мультипроцессирования делают z/OS идеальной операционной системой для обработки задач, требующих множества операций ввода-вывода. К типичным задачам для мэйнфреймов относятся приложения длительного выполнения, записывающие изменения в миллионы записей базы данных, и оперативные приложения, обрабатывающие тысячи интерактивных пользователей одновременно. Сравним это с операционной системой, используемой в однопользовательской компьютерной системе. Такая операционная система может выполнять программы только от лица одного пользователя. При использовании персонального компьютера, например, все ресурсы компьютера часто находятся в распоряжении одного пользователя.

Мультипроцессирование – одновременная работа двух и больше процессоров, совместно использующих различные аппаратные ресурсы

Когда множество пользователей выполняет много разных программ, помимо большого количества сложного оборудования, z/OS требуется большой объем памяти для обеспечения надлежащей системной производительности. Большие компании используют сложные бизнес-приложения, осуществляющие доступ к большим базам данных, и отраслевые программные продукты промежуточного уровня. Такие приложения требуют, чтобы операционная система осуществляла защиту конфиденциальности пользовательских данных, а также поддерживала совместное использование баз данных и программных служб.

Таким образом, мультипрограммирование, мультипроцессирование и потребность в большом количестве памяти означают, что z/OS должна обеспечивать нечто большее, чем просто выполнение простых однопользовательских приложений. В следующих разделах в общих чертах рассматриваются атрибуты, позволяющие z/OS управлять сложными компьютерными конфигурациями. Последующие разделы книги описывают эти возможности более подробно.

3.2.3 Модули и макросы

z/OS состоит из программных инструкций, управляющих работой компьютерной системы. Эти инструкции обеспечивают эффективное использование оборудования компьютера и возможность выполнения приложений; z/OS включает наборы инструкций, которые, например, принимают задания, преобразуют задания в распознаваемую компьютером форму, следят за заданиями, выделяют ресурсы для заданий, выполняют задания, осуществляют мониторинг заданий и осуществляют вывод выходных данных. Набор связанных инструкций называется *подпрограммой (routine)* или *модулем (module)*. Набор связанных модулей, обеспечивающих работу определенной функции системы, называется *системным компонентом (system component)*. Компонент управления рабочей нагрузкой (WLM) в z/OS, например, управляет системными ресурсами, тогда как менеджер завершения и восстановления (recovery termination manager, RTM) отвечает за восстановление системы.

Последовательности инструкций, выполняющие часто используемые системные функции, могут вызываться посредством выполняемых макроинструкций или *макросов (macros)*. В z/OS существуют макросы для таких функций, как открытие и закрытие файлов данных, загрузка и удаление программ и отправка сообщений оператору компьютера.

3.2.4 Управляющие блоки

В процессе выполнения программами работы системы z/OS они регистрируют выполняемые операции в областях хранения, называемых *управляющими блоками (control blocks)*. В общем, существует четыре типа управляющих блоков z/OS:

- управляющие блоки системы;
- управляющие блоки ресурсов;
- управляющие блоки заданий;
- управляющие блоки задач.

Каждый управляющий блок системы соответствует одной системе z/OS и содержит информацию по этой системе, в частности информацию о количестве используемых процессоров. Каждый управляющий блок ресурсов соответствует одному ресурсу, например процессору или устройству хранения. Каждый управляющий блок заданий соответствует одному заданию, выполняемому в системе. Каждый управляющий блок задач соответствует одной единице работы.

Управляющий блок – структура данных, служащая средством связи в z/OS

Управляющие блоки служат средством связи в z/OS. Такая связь возможна потому, что структура управляющего блока известна программам, которые его используют, вследствие чего эти программы могут найти нужную информацию о единице работы или ресурсе. Управляющие блоки, представляющие много модулей одного типа, можно объединять в очереди, в которых каждый управляющий блок указывает на следующий в цепи. Операционная система может осуществлять в очереди поиск информации об определенной единице работы или ресурсе, например:

- адресе управляющего блока или требуемой подпрограмме;
- действительных данных, в частности значении, числовой величине, параметре или имени;
- флагах состояния (обычно выражающихся отдельными битами в байте, где каждый бит имеет определенное значение).

z/OS использует огромное количество управляющих блоков, многие из которых имеют узкоспециализированное назначение. В этой главе рассматриваются три наиболее используемых управляющих блока:

- блок управления задачей (task control block, TCB), представляющий единицу работы или *задачу;*
- блок запроса обслуживания (service request block, SRB), представляющий запрос на системное обслуживание;
- блок управления адресным пространством (address space control block, ASCB), представляющий адресное пространство.

3.2.5 Физическая память, используемая в z/OS

С абстрактной точки зрения мэйнфреймы и все остальные компьютеры имеют два типа физической памяти[1]:

- Физическая память, расположенная непосредственно в процессоре мэйнфрейма. Она называется процессорной памятью (processor storage), реальной памятью (real storage) или *основной памятью (central storage)*; ее можно считать памятью мэйнфрейма.
- Физическая память, внешняя по отношению к мэйнфрейму; включает хранилища на устройствах с прямым доступом, таких, как дисковые приводы и приводы для магнитных лент. Такую память часто называют страничной памятью (paging storage) или *вспомогательной памятью (auxiliary storage)*.

Основное различие между двумя типами памяти связано со способом доступа, в частности:

- К основной памяти процессор осуществляет синхронный доступ. Другими словами, процессор должен ждать, пока данные будут извлечены из основной памяти[2].
- К вспомогательной памяти осуществляется асинхронный доступ. Для доступа к вспомогательной памяти процессор использует запросы ввода-вывода, выполнение которых происходит между другими рабочими запросами в системе. Во время запроса ввода-вывода процессор свободен для выполнения другой работы.

Основная память – физическая память в процессоре. Вспомогательная память – физическая память, внешняя по отношению к мейнфрейму; включает хранилища на устройствах с прямым доступом, таких как дисковые приводы и приводы для магнитных лент

Как и в случае с памятью персонального компьютера, основная память мэйнфрейма является сильно связанной с самим процессором, тогда как вспомогательная память мэйнфрейма размещается на (относительно) более медленных внешних дисковых приводах или приводах для магнитных лент. Так как основная память имеет более тесную интеграцию с процессором, процессору требуется гораздо меньше времени для доступа к данным в основной памяти, чем к вспомогательной памяти. Однако вспомогательная память является менее дорогостоящей, чем основная память. Большинство инсталляций z/OS используют большие объемы обоих типов памяти.

3.3 Обзор средств z/OS

Благодаря широкому набору системных средств и уникальных атрибутов z/OS хорошо подходит для обработки больших, сложных задач, в частности требующих множества операций ввода-вывода, доступа к большим объемам данных или полной безопасности.

---

[1] Многие компьютеры также используют быструю память, локальную с точки зрения процессора, называемую кешем процессора. Кеш невидим для программиста, приложений и даже для операционной системы.

[2] Некоторые процессоры используют технологии предварительной выборки (prefetching) или «конвейеризации» («pipelining») инструкций или данных для повышения производительности. Эти технологии невидимы для приложения и даже для операционной системы, однако хороший компилятор может организовать производимый им код таким образом, чтобы использовать эти технологии.

К типичным задачам для мэйнфреймов относятся приложения длительного выполнения, записывающие изменения в миллионы записей базы данных, и оперативные приложения, обрабатывающие тысячи интерактивных пользователей одновременно.

На рис. 3.2 представлен «снимок» операционной среды z/OS.

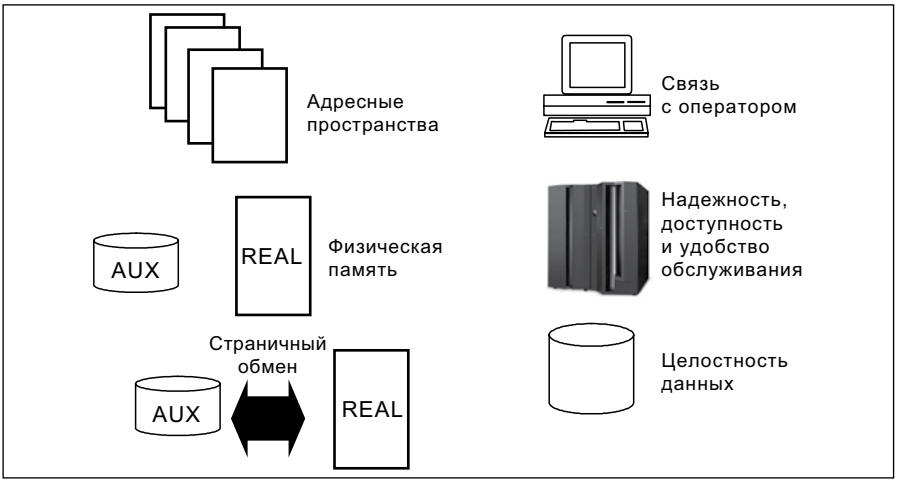


Рис. 3.2. Операционная среда z/OS

Эти средства подробно описываются в последующих разделах книги, а здесь рассматриваются вкратце.

- Адресное пространство описывает диапазон адресов виртуальной памяти, доступный для оперативного пользователя или выполняющейся программы.
- Доступны два типа физической памяти: основная память и вспомогательная память (AUX). Основная память также часто называется *реальной памятью* или *реальным хранилищем*.
- z/OS перемещает программы и данные между основной и вспомогательной памятью посредством процессов страничного обмена (paging) и свопинга (swapping).
- z/OS передает работу для выполнения (не показано на рисунке). Другими словами, выбираются программы для запуска, исходя из приоритета и возможности выполнения, после чего происходит загрузка программы и данных в основную память. Все программные инструкции и данные во время выполнения должны находиться в основной памяти.
- Широкий набор средств осуществляет управление файлами, хранящимися на устройствах хранения с прямым доступом (DASD) или картриджах с магнитной лентой.
- Операторы используют консоли для запуска и остановки z/OS, ввода команд и управления операционной системой.

Кроме того, z/OS имеет много других операционных характеристик, таких, как безопасность, возможность восстановления, целостность данных и управление рабочей нагрузкой.

3.4 Виртуальная память и другие понятия мэйнфреймов

z/OS использует оба типа физической памяти (основную и вспомогательную) для реализации еще одного типа памяти – *виртуальной памяти (virtual storage)*. В z/OS каждый пользователь имеет доступ скорее к виртуальной памяти, чем к физической памяти. Такое использование виртуальной памяти является важным условием для обеспечения уникальной возможности z/OS одновременно взаимодействовать с большим числом пользователей, обрабатывая большие объемы рабочей нагрузки.

3.4.1 Что такое виртуальная память?

При использовании виртуальной памяти каждая выполняемая программа может предполагать, что у нее есть доступ ко всей памяти, определенной схемой адресации архитектуры. Единственным ограничением является количество разрядов в адресе памяти. Такая возможность использования большого числа мест хранения является важной, так как программа может быть длинной и сложной, и код программы и нужные ей данные должны располагаться в основной памяти, чтобы процессор мог осуществлять к ним доступ.

z/OS поддерживает 64-разрядные адреса, что позволяет программе адресовать до 18 446 744 073 709 600 000 байт (16 экзабайт) памяти. В действительности основная память мэйнфрейма может иметь *гораздо меньший* объем. Настолько меньше зависит от модели компьютера и конфигурации системы.

Для того чтобы каждый пользователь мог работать так, как будто такой объем памяти действительно существует в компьютерной системе, z/OS хранит в основной памяти только активные фрагменты каждой программы. Остальной код и данные хранятся в файлах, называемых *страничными наборами данных (page data sets)*, во вспомогательной памяти, которая обычно представляет собой некоторое количество высокоскоростных устройств хранения с прямым доступом (DASD).

Таким образом, виртуальная память является сочетанием основной и вспомогательной памяти. Операционная система z/OS использует набор таблиц и индексов для отображения участков вспомогательной памяти в участки основной памяти. Она использует специальные параметры (битовые параметры) для отслеживания идентификационных данных и полномочий каждого пользователя или программы. z/OS использует множество компонентов менеджера памяти для управления виртуальной памятью. В этой главе кратко рассматриваются основные аспекты этого процесса.

Более подробно этот процесс описывается в разделе 3.4.4, «Обзор виртуальной памяти».

> **Термины.** Специалисты по работе с мэйнфреймами используют термины «центральная память», «реальная память», «реальное хранилище» и «основная память» как взаимозаменяемые. Подобным образом они используют термины «виртуальная память» и «виртуальное хранилище» как синонимы.

3.4.2 Что такое адресное пространство?

Диапазон виртуальных адресов, которые операционная система назначает пользователю или отдельно работающей программе, называется *адресным пространством (address space)*. Он представляет собой набор идущих подряд виртуальных адресов, доступных для выполнения инструкций и хранения данных. Диапазон виртуальных адресов в адресном пространстве начинается с нуля и может достигать максимального адреса, разрешенного архитектурой операционной системы.

Адресное пространство – диапазон виртуальных адресов, которые операционная система назначает пользователю или программе

z/OS выделяет для каждого пользователя уникальное адресное пространство и обеспечивает разделение между программами и данными, относящимися к каждому адресному пространству. В каждом адресном пространстве пользователь может запускать несколько задач, применяя *блоки управления задачами (task control blocks, TCB)*, поддерживающие мультипрограммирование.

В некотором смысле адресное пространство z/OS подобно процессу в UNIX, а идентификатор адресного пространства (address space identifier, ASID) подобен идентификатору процесса (process ID, PID). Кроме того, TCB подобны потокам в UNIX в том смысле, что каждая операционная система поддерживает одновременную обработку нескольких единиц работы.

Однако использование нескольких виртуальных адресных пространств в z/OS предоставляет некоторые особые преимущества. Виртуальная адресация позволяет использовать диапазоны адресов, превышающие возможности основной памяти системы. Использование нескольких виртуальных адресных пространств обеспечивает возможность виртуальной адресации для каждого задания в системе путем назначения каждому заданию собственного виртуального адресного пространства. Потенциально большое количество адресных пространств обеспечивает большой объем виртуальной адресации системы.

При использовании нескольких виртуальных адресных пространств ошибки ограничиваются одним адресным пространством, за исключением ошибок в памяти с общей адресацией, что повышает надежность системы и упрощает восстановление после ошибок. Программы в отдельных адресных пространствах защищены друг от друга. Изоляция данных в отдельном адресном пространстве также позволяет защитить данные.

z/OS использует множество адресных пространств. Применяется по меньшей мере одно адресное пространство для каждого выполняемого задания и еще одно адресное пространство для каждого пользователя, подключенного через TSO, telnet, rlogin или FTP (пользователи, подключенные к z/OS через такие подсистемы, как CICS или IMS, применяют адресное пространство подсистемы, а не собственные адресные пространства). Используется множество адресных пространств для реализации функций операционной системы, в частности связи с оператором, автоматизации, сетевых функций, безопасности и т. д.

Изоляция адресного пространства

Использование адресных пространств позволяет z/OS обеспечивать разделение между программами и данными, относящимися к отдельным адресным пространствам. Приватные области в адресном пространстве одного пользователя изолированы от приватных областей в других адресных пространствах, что в значительной степени обеспечивает безопасность операционной системы.

Кроме того, каждое адресное пространство содержит общую область, доступную для всех остальных адресных пространств. Так как адресное пространство отображает все доступные адреса, оно включает системный код и данные, а также пользовательский код и данные. Таким образом, не все отображаемые адреса доступны для пользовательского кода и данных.

Возможность совместного использования одних и тех же ресурсов многими пользователями подразумевает необходимость защиты пользователей друг от друга и защиты самой операционной системы. Наряду с такими методами, как использование «ключей» для защиты основной памяти и кодовых слов для защиты файлов данных и программ, отдельные адресные пространства обеспечивают отсутствие наложения пользовательских программ и данных.

Связь адресных пространств

В среде с несколькими виртуальными адресными пространствами приложениям необходимы способы связи адресных пространств; z/OS содержит два метода связи между адресными пространствами:

- назначение блока запроса обслуживания (SRB) – асинхронный процесс;
- использование служб межпространственной связи (cross-memory services) и регистров доступа – синхронный процесс.

Программа использует SRB для запуска процесса в другом адресном пространстве или в том же адресном пространстве. SRB по сути является асинхронным и работает независимо от программы, которая его создает, повышая, таким образом, доступность ресурсов в среде мультипроцесссирования. Более подробно SRB рассматриваются в разделе «Что такое SRB?».

Программа использует службы межпространственной связи для прямого доступа к адресному пространству другого пользователя. Можно сравнить службы межпространственной связи в z/OS с функциями разделяемой памяти (Shared Memory) в UNIX которые можно использовать без специальных полномочий. Однако в отличие от UNIX, службы межпространственной связи в z/OS требуют, чтобы обращающаяся программа имела специальные полномочия, контролируемые средством авторизации программ (Authorized Program Facility, APF). Этот метод позволяет обеспечить эффективный и безопасный доступ к данным, принадлежащим другим пользователям, к данным, принадлежащим пользователю, но для удобства хранящимся в другом адресном пространстве, а также быструю и защищенную связь с такими службами, как менеджеры транзакций и менеджеры баз данных.

Дополнительные сведения. Подробное описание использования служб межпространственной связи см. в руководстве *z/OS MVS Programming: Extended Addressability Guide*. Эту и другие публикации IBM можно найти на веб-сайте z/OS Internet Library

```
http://www.ibm.com/servers/eserver/zseries/zos/bkserv/
```

3.4.3 Что такое динамическая трансляция адреса?

Динамическая трансляция адреса (dynamic address translation, DAT) представляет собой процесс преобразования виртуального адреса при ссылке на участок памяти в соответствующий реальный адрес. Если виртуальный адрес уже ссылается на основную память, процесс динамической трансляции адреса может быть ускорен посредством использования буфера быстрого преобразования адреса (translation lookaside buffer). Если виртуальный адрес не ссылается на основную память, возникает прерывание по отсутствию страницы (page fault interrupt), после чего z/OS извлекает страницу из вспомогательной памяти.

При более внимательном рассмотрении этого процесса выясняется, что компьютер может обнаруживать ошибки типа, региона, сегмента или страницы, в зависимости от того, в каком месте структуры DAT обнаруживаются недопустимые записи. Ошибки повторяются в структуре DAT до тех пор, пока в конечном счете не выдается ошибка отсутствия страницы, после чего выполняется первоначальное выделение виртуальной страницы в основной памяти (если копия отсутствует во вспомогательной памяти) либо передача страницы из вспомогательной памяти.

Реализация DAT осуществляется как аппаратным, так и программным обеспечением посредством использования таблиц страниц, таблиц сегментов, таблиц регионов и буферов быстрого преобразования адреса. DAT позволяет различным адресным пространствам совместно использовать одну программу или другие данные с доступом только для чтения. Это связано с тем, что виртуальные адреса в разных адресных пространствах могут преобразовываться к одному фрейму основной памяти. В противном случае требовалось бы использовать много копий программы или данных, по одной для каждого адресного пространства.

3.4.4 Обзор виртуальной памяти

Как мы говорили, для выполнения процессором программной инструкции эта инструкция и данные, на которые она ссылается, должны находиться в основной памяти. В первых операционных системах было требование, чтобы при выполнении программных инструкций вся программа целиком находилась в основной памяти. Однако всей программе необязательно находиться в основной памяти при выполнении инструкции. Вместо того, перенося фрагменты программы в основную память по мере готовности процессора выполнять их, и перемещая их во вспомогательную память, когда они не нужны, операционная система может одновременно выполнять большее количество программ большего размера.

Итак, каким же образом операционная система отслеживает каждый фрагмент программы? Откуда она знает, где он находится: в основной или во вспомогательной памяти. Специалистам в z/OS важно понимать, каким образом операционная система это делает.

Физическая память разделена на области одинакового размера, доступ к которым осуществляется по уникальному адресу. В основной памяти эти области называются *фреймами (frames)*; во вспомогательной памяти они называются *слотами (slots)*. Подобным образом операционная система может разделить программу на фрагменты размером с фрейм или слот и назначить каждому фрагменту уникальный адрес. Такое назначение позволяет операционной системе отслеживать эти фрагменты. В z/OS фрагменты программы называются *страницами (pages)*. Эти области более подробно рассматриваются в разделе «Фреймы, страницы и слоты».

Для адресации страниц используются виртуальные, а не реальные адреса. С момента ввода программы в систему до завершения ее выполнения виртуальный адрес страницы остается неизменным, независимо от того, находится ли страница в основной или во вспомогательной памяти. Каждая страница состоит из отдельных элементов, называемых байтами, каждый из которых имеет уникальный виртуальный адрес.

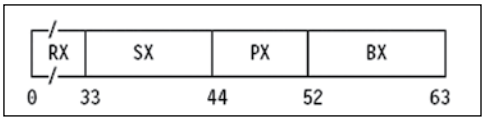
Формат виртуального адреса

Как говорилось выше, виртуальная память является иллюзией, создаваемой архитектурой, что система как будто имеет больше памяти, чем в действительности. Каждый пользователь или программа получает адресное пространство, и каждое адресное пространство содержит одинаковый диапазон адресов памяти. В действительности в основную память загружаются только те фрагменты адресного пространства, которые необходимы в данный момент времени; z/OS оставляет неактивные фрагменты адресных пространств во вспомогательной памяти; z/OS управляет адресными пространствами, используя единицы различного размера, в частности:

Страница	Адресные пространства разделяются на 4-килобайтовые блоки виртуальной памяти, называемые страницами.
Сегмент	Адресные пространства разделяются на 1-мегабайтовые блоки, называемые сегментами. Сегмент представляет собой блок последовательных виртуальных адресов, охватывающий мегабайт, начиная с 1-мегабайтового предела; 2-гигабайтовое адресное пространство, например, содержит 2 048 сегментов.
Регион	Адресные пространства разделяются на блоки размером от 2 до 8 Гб, называемые регионами. Регион представляет собой блок последовательных виртуальных адресов, охватывающий 2-8 Гб, начиная с 2-гигабайтового предела; 4-терабайтовое адресное пространство, например, содержит 2 048 регионов.

Виртуальный адрес соответственно имеет четыре основных поля: разряды 0–32 называются индексом региона (region index, RX), разряды 33–43 называются индексом сегмента (segment index, SX), разряды 44–51 называются индексом страницы (page index, PX), и разряды 52–63 называются индексом байта (byte index, BX).

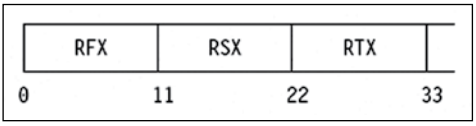
Виртуальный адрес имеет следующий формат:



В зависимости от значения управляющего элемента адресного пространства (address-space-control element), виртуальное адресное пространство может представлять

собой как 2-гигабайтовое пространство, содержащее один регион, так и 16-экзабайтовое пространство. Компонент RX виртуального адреса для 2-гигабайтового адресного пространства должен содержать одни нули; в противном случае возникает исключение.

Компонент RX виртуального адреса разделен на три поля. Разряды 0–10 называются первым индексом региона (region first index, RFX), разряды 11–21 называются вторым индексом региона (region second index, RSX), и разряды 22–32 называются третьим индексом региона (region third index, RTX). Разряды 0–32 виртуального адреса имеют следующий формат:



Виртуальный адрес, в котором самым старшим компонентом является RTX (42-разрядный адрес), может выполнять адресацию 4 Тб (2 048 регионов); виртуальный адрес, в котором самым старшим компонентом является RSX (53-разрядный адрес), может выполнять адресацию 8 Пб (4 миллиона регионов), и виртуальный адрес, в котором самым старшим компонентом является RFX (64-разрядный адрес), может выполнять адресацию 16 Эб (8 миллиардов регионов).

Каким образом в z/OS выполняется адресация виртуальной памяти

Как говорилось выше, использование виртуальной памяти в z/OS означает, что во время обработки в основной памяти должны находиться только активные фрагменты программы. Неактивные фрагменты содержатся во вспомогательной памяти. На рис. 3.3 представлена концепция виртуальной памяти, реализованная в z/OS, в действии.

На рис. 3.3 обратите внимание на следующее:

- Адрес является идентификатором нужного фрагмента информации, но не описанием того, где в основной памяти находится этот фрагмент информации. Благодаря этому размер адресного пространства (т. е. все адреса, доступные программе) может превышать размер доступной основной памяти.
- В большинстве пользовательских программ во всех ссылках на основную память используются адреса виртуальной памяти[1].
- Для преобразования виртуального адреса в физическое расположение в основной памяти используется механизм динамической трансляции адреса (DAT). Как показано на рис. 3.3, может существовать несколько виртуальных адресов 10254000, так как эти виртуальные адреса соответствуют различным адресам в основной памяти.
- Если запрашиваемого адреса нет в основной памяти, в z/OS возникает аппаратное прерывание и операционная система выполняет загрузку нужных инструкций и данных в основную память.

[1] Некоторые инструкции, в основном те, которые используются программами операционной системы, требуют использования реальных адресов.

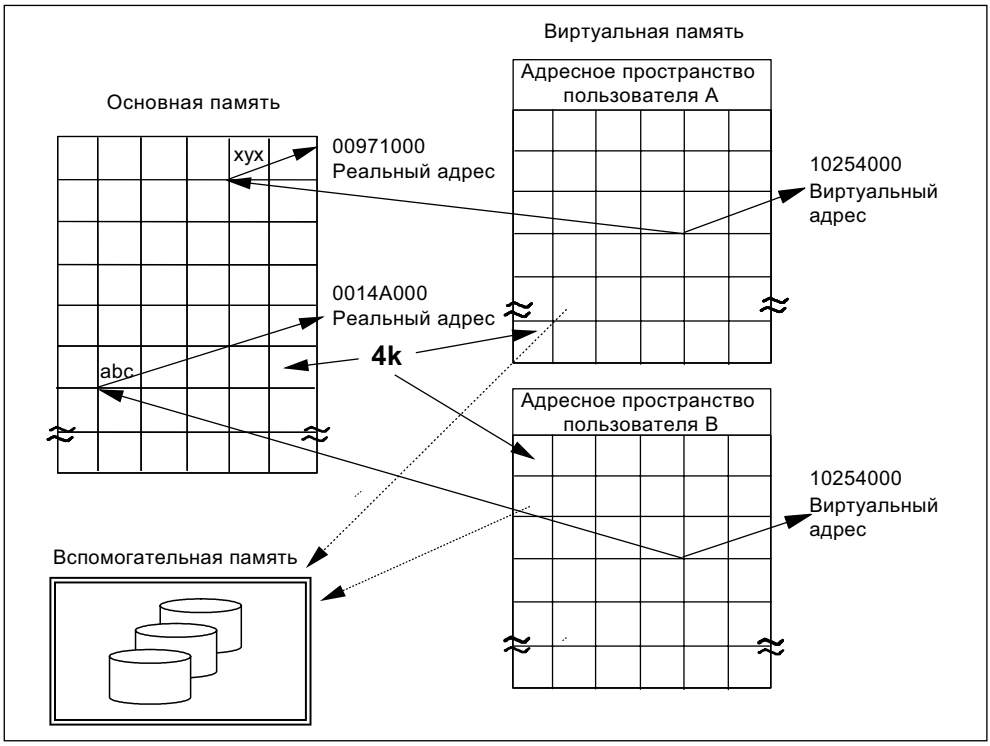


Рис. 3.3. Сочетание основной и вспомогательной памяти для создания иллюзии виртуальной памяти

Фреймы, страницы и слоты

После выбора программы для выполнения система переносит ее в виртуальную память, разделяет ее на страницы по 4 Кб, передает страницы в основную память для выполнения. С точки зрения программиста вся программа представляется таким образом, как будто она постоянно занимает непрерывное пространство в памяти. В действительности не все страницы программы обязательно находятся в основной памяти, а те страницы, которые находятся в основной памяти, необязательно занимают непрерывное пространство.

Фреймы – области в основной памяти, имеющие одинаковый размер и доступные по уникальному адресу.
Слоты – области во вспомогательной памяти, имеющие одинаковый размер и доступные по уникальному адресу

Фрагменты программы, выполняющиеся в виртуальной памяти, необходимо перемещать между основной и вспомогательной памятью. Для этого z/OS при управлении хранением работает с *блоками* размером 4 Кб. Определены следующие блоки:

- блок в основной памяти называется *фреймом (frame)*;
- блок в виртуальной памяти называется *страницей (page)*;
- блок во вспомогательной памяти называется *слотом (slot)*.

Страница, фрейм и слот имеют одинаковый размер – 4 Кб. Активная страница виртуальной памяти находится во фрейме основной памяти. Страница виртуальной памяти, которая становится неактивной, находится в слоте вспомогательной памяти (в страничном наборе данных). На рис. 3.4 показана связь между страницами, фреймами и слотами.

На рис. 3.4 показано, каким образом z/OS выполняет загрузку страниц для программы, выполняющейся в виртуальной памяти. Квадраты, обозначенные буквами, представляют части программы. В этом упрощенном представлении части программы A, E, F и H активны и выполняются во фреймах основной памяти, тогда как части B, C, D и G неактивны и перемещены в слоты вспомогательной памяти. Тем не менее все части программы находятся в виртуальной памяти и имеют виртуальные адреса.

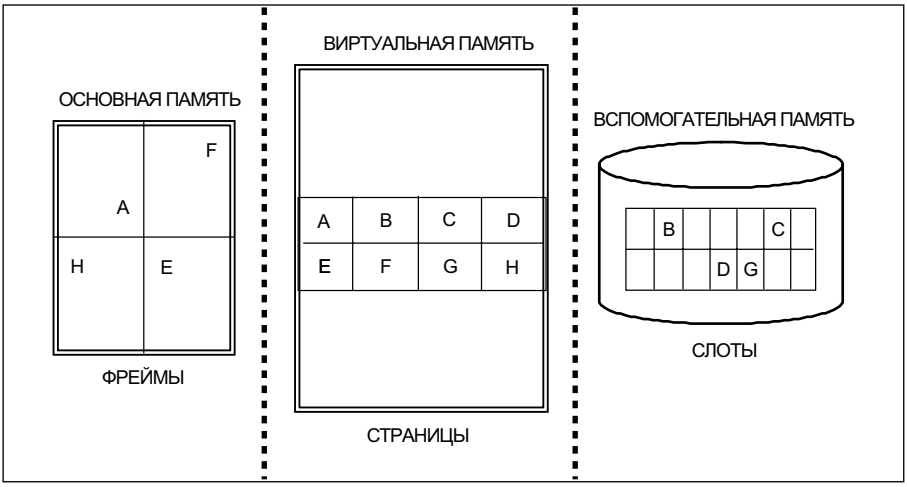


Рис. 3.4. Фреймы, страницы и слоты

3.4.5 Что такое страничный обмен?

Как говорилось выше, z/OS использует набор таблиц для того, чтобы определить, находится ли страница в основной или вспомогательной памяти и где именно. Чтобы найти страницу программы, z/OS ищет виртуальный адрес страницы в таблице, вместо того чтобы просматривать всю физическую память. Затем z/OS при необходимости передает страницу в основную память или во вспомогательную память. Такое перемещение страниц между слотами вспомогательной памяти и фреймами основной памяти называется *страничным обменом (paging)*. Понимание выполнения страничного обмена важно для понимания использования виртуальной памяти в z/OS.

В z/OS страничный обмен невидим для пользователя. Во время выполнения задания происходит передача (или *подкачка*) только нужных фрагментов приложения в основную память. Страницы остаются в основной памяти до тех пор, пока они не станут ненужными или пока то же приложение или приложение с более высоким

приоритетом не затребует другую страницу и при этом в основной памяти не будет свободного места. При выборе страниц для вытеснения во вспомогательную память z/OS использует алгоритм наименьшего использования. Другими словами, z/OS предполагает, что страница, которая определенное время не использовалась, скорее всего не будет использоваться и в ближайшем будущем.

Как выполняется страничный обмен в z/OS

Помимо DAT-оборудования и таблиц сегментов и страниц, нужных для трансляции адресов, при страничном обмене используется множество системных компонентов для управления перемещением страниц, а также несколько дополнительных таблиц для определения наиболее актуальной версии каждой страницы.

Для того чтобы понять принцип выполнения страничного обмена, рассмотрим ситуацию, когда DAT при трансляции адресов обнаруживает недопустимую запись в таблице страниц; это указывает на то, что необходима страница, не находящаяся во фрейме основной памяти. Чтобы разрешить ошибку отсутствия страницы, система должна перенести страницу из вспомогательной памяти. Однако прежде она должна найти свободный фрейм в основной памяти. Если такого фрейма нет, необходимо сохранить запрос и освободить выделенный фрейм. Для освобождения фрейма система копирует его содержимое во вспомогательную память и помечает соответствующую запись в таблице страниц как недействительную. Эта операция называется вытеснением страницы (page-out).

После нахождения фрейма для требуемой страницы содержимое страницы копируется из вспомогательной памяти в основную память и бит недействительности записи таблицы страниц сбрасывается. Эта операция называется подкачкой страницы (page-in).

Страничный обмен может также выполняться, когда z/OS загружает всю программу в виртуальную память; z/OS получает виртуальную память для пользовательской программы и выделяет фреймы основной памяти для каждой страницы. После этого каждая страница является активной и допускает выполнение обычных операций страничного обмена; другими словами, самые активные страницы удерживаются в основной памяти, тогда как страницы, неактивные на данный момент, вытесняются во вспомогательную память.

Изъятие страницы

z/OS пытается обеспечивать достаточный запас доступных фреймов основной памяти. Когда программа обращается к странице, не находящейся в основной памяти, z/OS использует фрейм страницы основной памяти из запаса доступных фреймов.

Когда этот запас становится недостаточным, z/OS выполняет *изъятие страницы (page stealing)* для его пополнения; другими словами, выбирается фрейм, выделенный для активного пользователя, который делается доступным для других задач. Решение об изъятии определенной страницы основано на истории активности каждой страницы, находящейся во фрейме основной памяти. Страницы, которые относительно долгое время были неактивны, являются подходящими кандидатами для изъятия.

Счетчик интервалов отсутствия обращений

z/OS использует сложный алгоритм страничного обмена для эффективного управления виртуальной памятью, основанный на информации о недавно использовавшихся страницах. Счетчик интервалов отсутствия обращений показывает, сколько времени прошло с момента обращения программы к странице. Система через регулярные интервалы проверяет бит обращения для каждого страничного фрейма. Если бит обращения выключен (т. е. к фрейму не было обращений), система увеличивает значение счетчика интервалов отсутствия обращений для этого фрейма. Добавляется количество секунд с момента последней проверки счетчика для этого адресного пространства. Если бит обращения включен, это означает, что имело место обращение к фрейму, после чего система его отключает и сбрасывает счетчик интервалов отсутствия обращений для фрейма в нуль. Фреймы с наивысшими значениями счетчика интервалов отсутствия обращений – наиболее вероятные кандидаты для изъятия.

z/OS также использует различные менеджеры памяти для отслеживания всех страниц, фреймов и слотов в системе. Более подробно это описывается в разделе 3.4.8, «Роль менеджеров памяти».

3.4.6 Свопинг и рабочий набор

Свопинг (swapping) представляет собой процесс передачи всех страниц адресного пространства между основной и вспомогательной памятью. Загруженное адресное пространство является активным и содержит страницы во фреймах основной памяти и в слотах вспомогательной памяти. Выгруженное адресное пространство является неактивным; адресное пространство находится во вспомогательной памяти и не может выполняться до тех пор, пока оно не будет загружено.

Свопинг – процесс передачи всего адресного пространства между основной и вспомогательной памятью

В основной памяти в любой момент времени чаще всего находится только подмножество страниц адресного пространства (называемое его *рабочим набором*), тогда как свопинг выполняет перемещение всего адресного пространства. Свопинг является одним из методов, используемых z/OS для балансировки системной нагрузки и обеспечения наличия достаточного запаса доступных фреймов основной памяти.

Свопинг выполняется менеджером управления ресурсами (System Resource Manager, SRM) в ответ на рекомендации, получаемые от менеджера управления рабочей нагрузкой (Workload Manager, WLM). Подробное описание WLM см. в разделе 3.5, «Что такое управление рабочей нагрузкой?».

3.4.7 Что такое защита памяти?

До сих пор мы рассматривали виртуальную память только в контексте одного пользователя или программы. В действительности, конечно же, за использование системы соревнуется много программ и пользователей. z/OS применяет следующие методы обеспечения целостности работы каждого пользователя:

- личное адресное пространство для каждого пользователя;
- защита страниц;
- защита зоны начальных адресов;
- несколько ключей защиты памяти, описываемых в этом разделе.

Как используются ключи защиты памяти

В z/OS защита информации в основной памяти от несанкционированного использования осуществляется посредством нескольких ключей защиты памяти. С каждым 4-килобайтовым фреймом основной памяти связано контрольное поле, называемое ключом.

При создании запроса на изменение содержимого участка в основной памяти выполняется сравнение ключа, связанного с запросом, с ключом защиты памяти. Если ключи совпадают или если программа выполняется с ключом 0, запрос выполняется. Если ключ, связанный с запросом, не соответствует ключу памяти, система отклоняет запрос и выдает прерывание программного исключения.

При создании запроса на чтение (или извлечение) содержимого участка основной памяти запрос автоматически выполняется, если только не включен бит защиты от извлечения, указывающий, что фрейм защищен от извлечения. При создании запроса на доступ к содержимому участка основной памяти с защитой от извлечения происходит сравнение ключа в памяти с ключом, связанным с запросом. Если ключи совпадают или если инициатор запроса имеет ключ 0, запрос выполняется. Если ключи не совпадают и инициатор запроса имеет ключ, отличный от нуля, система отклоняет запрос и выдает прерывание программного исключения.

Как назначаются ключи защиты памяти

z/OS использует 16 ключей защиты памяти. Каждый конкретный ключ назначается в соответствии с типом выполняемой работы. Как показано на рис. 3.5, ключ хранится в битах 8–11 слова состояния программы (program status word, PSW). PSW назначается каждому заданию в системе.

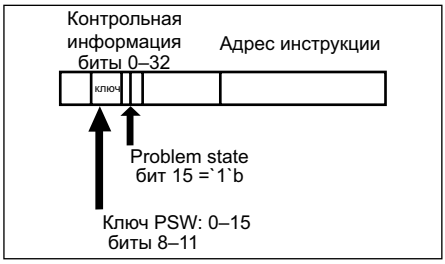


Рис. 3.5. Расположение ключа защиты памяти

Ключи защиты памяти 0–7 используются в z/OS базовой управляющей программой (base control program, BCP) и различными подсистемами и программными продуктами промежуточного уровня. Ключ защиты памяти 0 является главным ключом (master key). Его использование ограничено теми частями BCP, которые требуют

почти неограниченных возможностей записи и извлечения. Почти всегда, когда с запросом на доступ или изменение содержимого участка основной памяти связан ключ защиты памяти 0, это означает, что запрос будет выполнен.

Ключи защиты памяти 8–15 назначаются пользователям. Так как все пользователи изолированы в личных адресных пространствах, большинство пользователей, чьи программы выполняются в виртуальном регионе, могут использовать одинаковый ключ защиты памяти. Эти пользователи обозначаются V=V (virtual = virtual) и им назначается ключ 8. Однако некоторые пользователи должны работать в регионе основной памяти. Эти пользователи обозначаются V=R (virtual = real) и требуют отдельных ключей защиты памяти, так как их адреса не защищены процессом DAT, поддерживающим разделение между адресными пространствами. Без использования отдельных ключей пользователи V=R могут обращаться к коду и данным друг друга. Эти ключи находятся в диапазоне от 9 до 15.

3.4.8 Роль менеджеров памяти

Управление фреймами основной памяти и слотами вспомогательной памяти, а также поддерживаемыми ими страницами виртуальной памяти осуществляют различные компоненты z/OS. Эти компоненты называются менеджером реальной памяти (а не менеджером *основной* памяти), менеджером вспомогательной памяти и менеджером виртуальной памяти. Ниже мы вкратце рассмотрим роли каждого из них.

Менеджер реальной памяти

Менеджер реальной памяти (real storage manager, RSM™) отслеживает содержимое основной памяти. Он управляет операциями страничного обмена, описанными выше, в частности подкачкой, вытеснением и изъятием страниц, и помогает осуществлять загрузку и выгрузку адресных пространств. RSM также выполняет *фиксацию страницы (page fixing)* – пометку страниц как недоступных для изъятия.

Менеджер вспомогательной памяти

Менеджер вспомогательной памяти (auxiliary storage manager, ASM) использует страничные наборы данных системы для отслеживания слотов вспомогательной памяти, в частности:

- слотов для страниц виртуальной памяти, не находящихся во фреймах основной памяти;
- слотов для страниц, не занимающих фреймы, но которые в связи с отсутствием изменений в содержимом фреймов все еще действительны.

Когда требуется выполнить подкачку или вытеснение страницы, ASM совместно с RSM обнаруживают подходящие фреймы основной памяти и слоты вспомогательной памяти.

Менеджер виртуальной памяти

Менеджер виртуальной памяти (virtual storage manager, VSM™) обрабатывает запросы на получение и освобождение виртуальной памяти. Кроме того, VSM управляет выде-

лением памяти для любой программы, которая должна выполняться в основной, а не в виртуальной памяти. Основная память выделяется для кода и данных при их загрузке в виртуальную память. При их выполнении программы могут запросить больший объем памяти, используя средства системного сервиса, например макроса GETMAIN. Для освобождения памяти программы используют макрос FREEMAIN.

VSM отслеживает схему отображения виртуальной памяти для каждого адресного пространства. При этом адресное пространство представляется в виде набора из 256 *подпулов (subpools)*, которые являются логически связанными областями виртуальной памяти, обозначаемыми числами от 0 до 255. Логическая связанность означает, что области памяти в подпуле имеют общие свойства, в частности:

- ключ защиты памяти;
- защиту от извлечения, поддержку страничного обмена и поддержку свопинга;
- местонахождение в виртуальной памяти (до или после 16 Мб);
- возможность совместного использования несколькими задачами.

Некоторые подпулы (обозначаемые числами от 128 до 255) предопределены для использования системными программами. Подпул 252, например, предназначен для программ из авторизованных библиотек. Другие подпулы (обозначаемые числами от 0 до 127) определяются пользовательскими программами.

3.4.9 Краткая история виртуальной памяти и 64-разрядной адресуемости

В 1970 году IBM выпустила System/370 – первую свою архитектуру, использующую виртуальную память и адресные пространства. С тех пор операционная система претерпела множество изменений. Одной из основных областей развития и изменения является адресуемость.

Программа, выполняемая в адресном пространстве, может обращаться ко всем участкам памяти, связанным с этим адресным пространством. В этой книге способность программы обращаться ко всем участкам памяти, связанным с адресным пространством, называется *адресуемостью (addressability)*.

Адресуемость – способность программы обращаться ко всем участкам памяти, связанным с адресным пространством

В System/370 адреса памяти имели длину 24 разряда, что означало, что максимальным доступным адресом был адрес 16 777 215 байт (2^{24}–1 байт)[1]. Использование 24-разрядной адресуемости позволяло MVS/370 (операционной системе того времени) выделять каждому пользователю адресное пространство размером 16 Мб. Несколько лет спустя, когда в MVS/370 было добавлено больше функций и требовалось обрабатывать более сложные приложения, доступ даже к 16 Мб виртуальной памяти стал недостаточным для удовлетворения потребностей пользователей.

С выходом архитектуры System/370-XA в 1983 IBM расширила адресуемость архитектуры до 31 разряда. С 31-разрядной адресацией операционная система (те-

[1] Адресация начинается с 0, поэтому последний адрес всегда на единицу меньше общего числа адресуемых байтов.

перь называвшаяся MVS Extended Architecture или MVS/XA™) увеличила адресуемость виртуальной памяти с 16 Мб до 2 Гб. Другими словами, MVS/XA обеспечивала адресное пространство для пользователей, которое было в 128 раз больше, чем адресное пространство, обеспечиваемое в MVS/370; 16-мегабайтовый адрес стал точкой деления между двумя архитектурами, которую часто называют *«линией» (line)* (рис. 3.6).

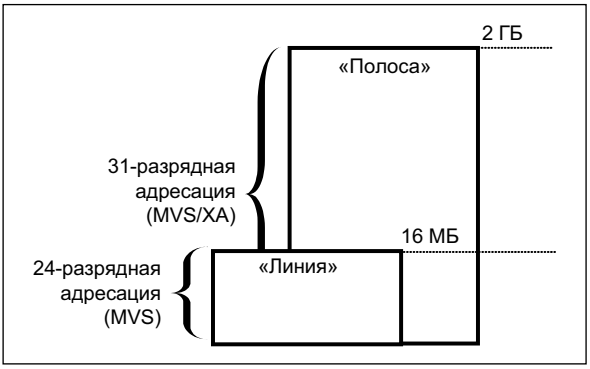


Рис. 3.6. 31-разрядная адресуемость позволяет использовать 2-гигабайтовые адресные пространства в MVS/XA

Новая архитектура не требовала, чтобы клиенты изменяли существующие приложения. Для обеспечения совместимости с существующими программами MVS/XA поддерживала программы, изначально разработанные для 24-разрядной адресации в MVS/370, в то же время позволяя разработчикам приложений создавать новые программы, использующие технологию 31-разрядной адресации.

Для обеспечения совместимости между разными схемами адресации MVS/XA не использовал *старший разряд (high-order bit)* адреса (бит 0) при адресации. Вместо этого MVS/XA зарезервировал этот разряд для обозначения количества разрядов, используемых для разрешения адреса: 31-разрядная адресация (бит 0 включен) или 24-разрядная адресация (бит 0 выключен).

С выходом мэйнфреймов zSeries в 2000 году IBM еще больше расширила адресуемость архитектуры до 64 разрядов. При 64-разрядной адресации потенциальный размер адресного пространства z/OS расширяется настолько, что нужны новые термины для его описания. Каждое адресное пространство, называемое 64-разрядным адресным пространством, имеет размер 16 Эб; экзабайт составляет немного больше 1 000 000 000 Гб. Новое адресное пространство имеет 2^{64} адресов. Это в 8 миллиардов раз больше размера предыдущего 2-гигабайтового адресного пространства и составляет 18 446 744 073 709 600 000 байт (рис. 3.7).

Мы говорим, что потенциальный размер составляет 16 Эб, потому что по умолчанию z/OS продолжает создавать адресные пространства размером 2 Гб. Адресное пространство превышает этот предел, только если выполняющаяся в нем программа выделяет виртуальную память размером больше 2 Гб. В этом случае z/OS увеличивает размер памяти, доступной для пользователя, с 2 Гб до 16 Эб.

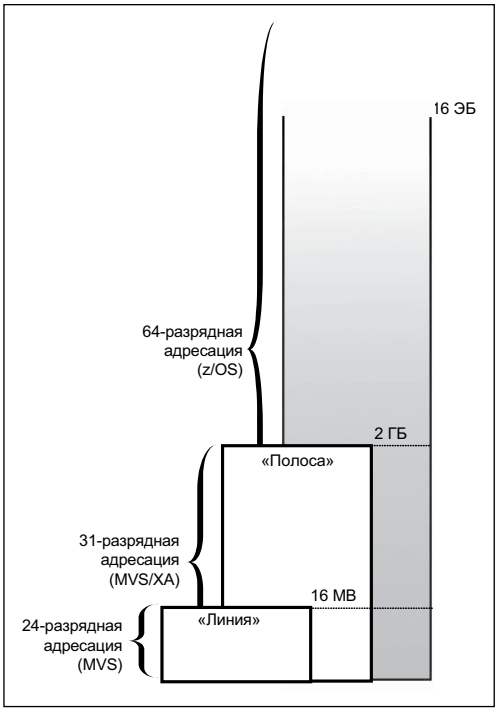


Рис. 3.7. 64-разрядная адресуемость позволяет обеспечить 16 Эб адресуемой памяти

Программа, выполняющаяся в z/OS на мэйнфрейме zSeries, может использовать 24-, 31- или 64-разрядную адресацию (и при необходимости может переключаться между ними). Для адресации больших объемов виртуальной памяти, доступных при использовании 64-разрядной архитектуры, программа использует 64-разрядные инструкции. Несмотря на то что архитектура поддерживает уникальные 64-разрядные инструкции, программа при необходимости может использовать как 31-разрядные, так и 64-разрядные инструкции.

В целях совместимости, структура областей ниже 2 Гб адресного пространства осталась такой же, что обеспечивает среду, которая может поддерживать и 24-разрядную, и 31-разрядную адресацию. Область, отделяющая область виртуальной памяти, расположенную ниже 2-гигабайтового адреса, от приватной области пользователя, называется *«полосой» (bar)*, как показано на рис. 3.8. Приватная область пользователя выделяется для кода приложений, а не для кода операционной системы.

$0 – 2^{31}$	Схема такая же (см. рис. 3.8).
$2^{31} – 2^{32}$	Область от 2 до 4 Гб называется «полосой». Все, что находится ниже «полосы», адресуется с использованием 31-разрядного адреса. Все, что находится выше «полосы», требует 64-разрядной адресации.
$2^{32} – 2^{41}$	Неразделяемая область (приватная область пользователя), начинается с 4 Гб и продолжается до 2^{41}.
$2^{41} – 2^{50}$	Разделяемая область (для совместного использования памяти) начинается с 2^{41} и продолжается до 2^{50} или выше, если требуется.

$2^{50} - 2^{64}$ Верхняя неразделяемая область (приватная область пользователя), начинается с 2^{50} или там, где заканчивается разделяемая область, и продолжается до 2^{64}.

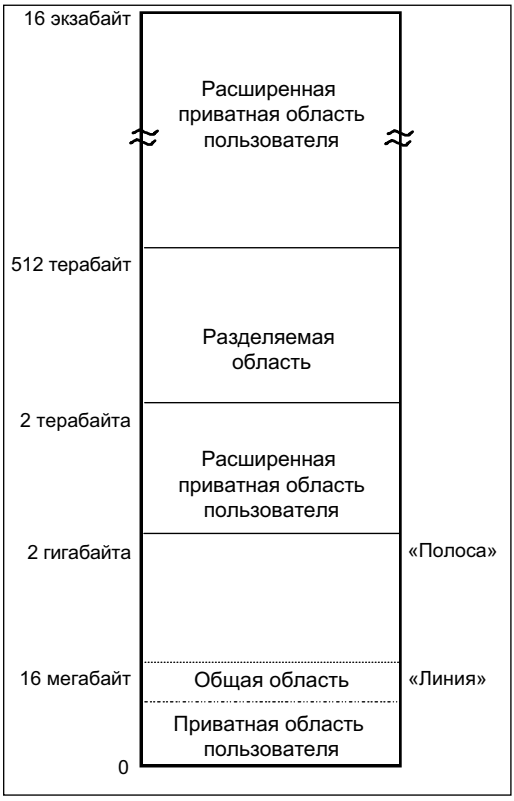


Рис. 3.8. Схема хранения для 64-разрядного адресного пространства

В 16-экзабайтовом адресном пространстве с 64-разрядной адресацией виртуальной памяти существует три дополнительных уровня таблиц трансляции, называемых таблицами регионов: третья таблица регионов (region third table, R3T), вторая таблица регионов (region second table, R2T) и первая таблица регионов (region first table, R1T). Таблицы регионов имеют длину 16 Кб, и каждая таблица содержит 2 048 записей. Каждый регион имеет размер 2 Гб.

Форматы таблиц сегментов и таблиц страниц остались такими же, как и для виртуальных адресов ниже «полосы». При трансляции 64-разрядного виртуального адреса, после того как система определила соответствующую запись 2-гигабайтового региона, указывающую на таблицу сегментов, процесс идентичен описанному выше.

3.4.10 Что означает выражение «память ниже "линии"»?

Программы и данные z/OS находятся в виртуальной памяти, которая при необходимости отображается в основной памяти. Большинство программ и данных не зависят от своих реальных адресов. Однако некоторые программы z/OS зависят от реальных

адресов, и некоторые требуют, чтобы реальные адреса были меньше 16 Мб. Программисты в z/OS называют эту часть памяти памятью, находящейся «ниже 16-мегабайтовой "линии"».

В z/OS программа включает такой атрибут, как *режим размещения (residence mode, RMODE)*, который определяет, должна ли программа находиться (загружаться) в область памяти ниже 16 Мб. Программа с RMODE(24) должна находиться в области ниже 16 Мб, тогда как программа с RMODE(31) может находиться где угодно в виртуальной памяти.

К программам, требующим размещения ниже «линии», относятся все программы, выделяющие блок управления данными (data control block, DCB). Эти программы тем не менее часто могут иметь 31-разрядный режим размещения RMODE(31); так как они могут выполняться в 31-разрядном режиме адресации AMODE(31). z/OS резервирует максимально возможный объем в области основной памяти ниже 16 Мб для таких программ и главным образом обеспечивает реализацию их зависимости от основной памяти без внесения каких-либо изменений.

Тысячи программ, используемых в настоящее время, имеют режим адресации AMODE(24), т. е. и режим размещения RMODE(24). Это относится ко всем программам, написанным до выхода MVS/XA и впоследствии не изменявшихся. В настоящее время сравнительно немного причин для создания новых программ в режиме AMODE(24), так что новые приложения вряд ли будут использовать режим RMODE(24).

3.4.11 Что в адресном пространстве?

По-другому адресное пространство можно представить как карту виртуальной памяти, доступной для кода и данных. Адресное пространство дает каждому программисту доступ ко всем адресам, доступным через компьютерную архитектуру (выше мы определили это как адресуемость).

z/OS выделяет каждому пользователю уникальное адресное пространство и поддерживает разделение между программами и данными, относящимися к каждому адресному пространству. Однако так как адресное пространство осуществляет отображение всех доступных адресов, оно включает системный код и данные, равно как и пользовательский код и данные. Таким образом, не все отображаемые адреса доступны для пользовательского кода и данных.

Понимание разделения областей памяти в адресном пространстве легче представить с помощью диаграммы. Диаграмма, представленная на рис. 3.9, является более подробной, чем это необходимо на данном этапе курса, однако она представлена для того, чтобы показать, что адресное пространство поддерживает разделение между программами и данными, принадлежащими пользователю, и программами и данными, принадлежащими к операционной системе.

На рис. 3.9 представлены основные области памяти в каждом адресном пространстве. Рассмотрим эти области.

- Вся память выше 2 Гб.

  Эта область называется *верхней виртуальной памятью (high virtual storage)*, и она адресуется только программами, работающими в 64-разрядном режиме.

Она разделена верхней виртуальной разделяемой областью, которая представляет собой область размера, определяемого инсталляцией, которую можно использовать для создания связей между адресными пространствами через полученные области в пределах этой области.

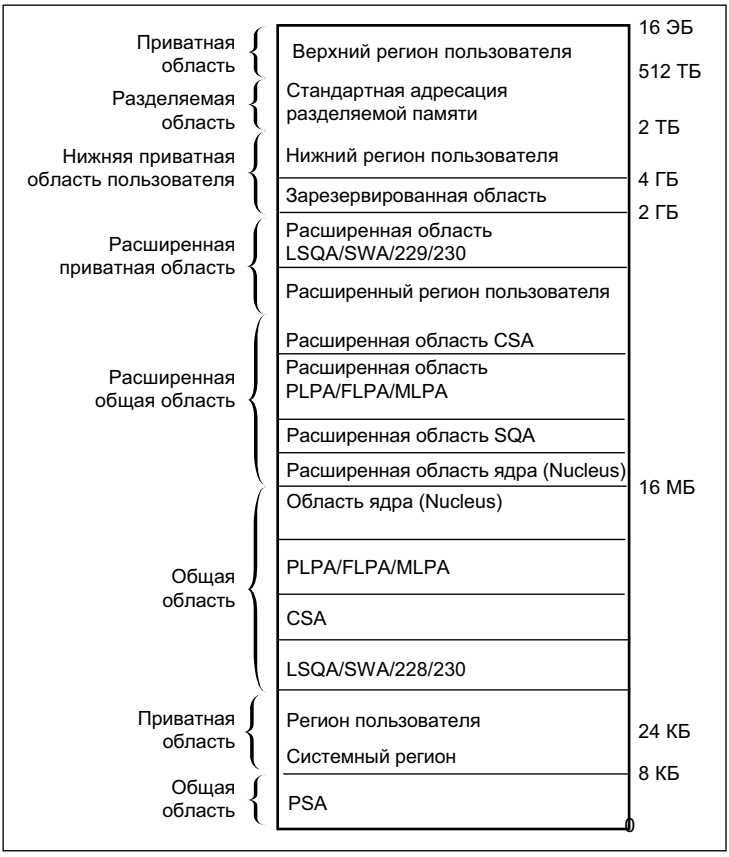


Рис. 3.9. Области памяти в адресном пространстве

- Расширенные области выше 16 Мб.
  Диапазон областей, находящихся выше «линии» (16 Мб), но ниже «полосы» (2 Гб), является своего рода «зеркальным отображением» общей области ниже 16 Мб. Они имеют такие же атрибуты, что и соответствующие им области ниже «линии», но из-за дополнительной памяти выше «линии» их размеры гораздо больше.
- Область ядра (Nucleus).
  Представляет собой область общей памяти с ключом 0, содержащую управляющие программы операционной системы.
- SQA.
  Эта область содержит данные системного уровня (с ключом 0), используемые несколькими адресными пространствами. Область SQA является фиксированной (неперемещаемой); это означает, что она находится в основной памяти до

ее освобождения запрашивающей программой. Размер области SQA предопределен инсталляцией и не может изменяться, пока операционная система активна. Кроме того, она имеет уникальную способность к переполнению с выходом в область CSA, если существует неиспользуемая область CSA, допускающая преобразование в SQA.

- PLPA/FLPA/MLPA.
  Эта область содержит области загрузки модулей (перемещаемая область загрузки модулей, фиксированная область загрузки модулей и изменяемая область загрузки модулей), содержащие программы системного уровня, часто выполняемые несколькими адресными пространствами. По этой причине области загрузки модулей находятся в общей области, адресуемой всеми адресными пространствами, что устраняет необходимость наличия копий программы в каждом адресном пространстве. Эта область памяти находится ниже «линии» и является адресуемой программами, работающими в 24-разрядном режиме.
- CSA.
  Эта часть общей области памяти (адресуемая всеми адресными пространствами) доступна для всех приложений. CSA часто применяется для хранения данных, часто используемых несколькими адресными пространствами. Размер области CSA устанавливается во время инициализации системы (IPL) и не может изменяться, пока операционная система активна.
- LSQA/SWA/подпул 228/подпул 230.
  Этот набор подпулов, каждый из которых имеет определенные атрибуты, используется главным образом системными функциями, когда функции требуют изоляции памяти на уровне адресного пространства. Находясь ниже «линии», эти области адресуются программами, работающими в 24-разрядном режиме.
- Регион пользователя (User Region).
  Эта область доступна для любой программы, выполняющейся в адресном пространстве пользователя, включая основные программы пользователя. Она расположена ниже «линии» и поэтому допускает адресацию программами, выполняющимися в 24-разрядном режиме.
- Системный регион (System Region).
  Эта небольшая область (обычно содержащая только четыре страницы) зарезервирована для использования задачей управления регионами каждого адресного пространства.
- Область префиксации (Prefixed Save Area, PSA).
  Эта область также называется «нижним ядром» («Low Core»). PSA представляет собой общую область виртуальной памяти с адресами от 0 до 8191 в каждом адресном пространстве. Используется по одному PSA для каждого процессора, установленного в системе. PSA сопоставляет архитектурно фиксированные участки памяти аппаратного и программного обеспечения с процессором. Так как каждому процессору соответствует отдельная область PSA, с точки зрения программы, выполняющейся в z/OS, содержимое PSA может измениться в любой момент, как только программа будет передана другому процессору. Эта возможность является уникальной для области PSA и осуществляется посредством особого метода управления DAT, называемого префиксацией (prefixing).

По причине огромного диапазона адресуемой памяти в адресном пространстве схема на рис. 3.9 представлена не в масштабе.

Каждое адресное пространство в системе представлено блоком управления адресным пространством (address space control block, ASCB). Для того чтобы представить адресное пространство, система создает ASCB в общей памяти (области системных очередей – SQA), что делает его доступным для других адресных пространств.

3.4.12 Системные адресные пространства и главный планировщик

Многие системные функции z/OS выполняются в собственных адресных пространствах. Подсистема главного планировщика, например, выполняется в адресном пространстве *MASTER* и используется для установления связи между z/OS и его адресными пространствами.

При запуске z/OS главные подпрограммы инициализации выполняют инициализацию системных служб, таких, как системный журнал и задача связи, и запускают адресное пространство главного планировщика. Затем главный планировщик может запустить подсистему управления заданиями (JES2 или JES3). JES является первичной подсистемой управления заданиями. На многих рабочих системах JES не запускается сразу же; вместо этого пакет автоматизации запускает все задачи в контролируемой последовательности. Затем происходит запуск других подсистем.

Подсистемы определяются в специальном файле системных параметров, называемом библиотекой параметров (parameter library, PARMLIB). Эти подсистемы являются *вторичыми подсистемами (secondary subsystems)*.

Каждое создаваемое адресное пространство имеет связанный с ним номер, называемый идентификатором адресного пространства (address space ID, ASID). Так как главный планировщик является первым адресным пространством, созданным в системе, он становится адресным пространством под номером 1 (ASID=1). Другие системные адресные пространства запускаются позже в процессе инициализации z/OS.

На данном этапе вам требуется понять лишь то, что z/OS и связанным с ней подсистемам нужны собственные адресные пространства для обеспечения функционирования операционной системы. Ниже приведены описания каждого типа адресного пространства.

- Системные адресные пространства.
  Запуск системных адресных пространств z/OS происходит после инициализации главного планировщика. Эти адресные пространства выполняют функции для всех других типов адресных пространств, запускаемых в z/OS.
- Адресные пространства подсистем.
  z/OS требует использования различных подсистем, таких, как первичная подсистема управления заданиями (job entry subsystem, JES; описание JES см. в главе 7, «Пакетная обработка и JES»). Кроме того, используются адресные пространства для программных продуктов промежуточного уровня, таких, как DB2, CICS и IMS.

Помимо системных адресных пространств, обычно используется много адресных пространств для пользователей и отдельно выполняющихся программ, например:

- Адресные пространства TSO/E создаются для каждого пользователя, подключающегося к z/OS (описание см. в главе 4, «TSO/E, ISPF и UNIX: интерактивные средства z/OS»).
- Адресное пространство создается для каждого пакетного задания, выполняющегося в z/OS. Адресные пространства пакетных заданий запускаются подсистемой JES.

3.5 Что такое управление рабочей нагрузкой?

В z/OS за управление системными ресурсами отвечает компонент управления рабочей нагрузкой (WLM). WLM управляет обработкой рабочей нагрузки в системе в соответствии с бизнес-целями компании, такими, как время реагирования. Кроме того, WLM управляет использованием системных ресурсов, в частности процессорами и памятью, для достижения этих целей.

3.5.1 Чем занимается WLM?

Попросту говоря, WLM имеет три цели:

- Достижение бизнес-целей, определяемых инсталляцией, путем автоматического выделения ресурсов сисплекса рабочим задачам в зависимости от их важности и целей. Эта цель называется *достижением цели (goal achievement)*.
- Достижение оптимального использования системных ресурсов с точки зрения системы. Эта цель называется *пропускной способностью (throughput)*.
- Достижение оптимального использования системных ресурсов с точки зрения отдельного адресного пространства. Эта цель называется *временем реагирования (response time)* или *оборотным временем (turnaround time)*.

Управление рабочей нагрузкой – компонент z/OS, управляющий системными ресурсами в соответствии с заявленными бизнес-целями

Достижение цели является первой и самой важной задачей WLM. Затем идет оптимизация пропускной способности и минимизация оборотного времени адресных пространств. Часто последние две цели являются взаимно противоречащими. Оптимизация пропускной способности предполагает обеспечение полной занятости ресурсов. С другой стороны, оптимизация времени реагирования и оборотного времени требует, чтобы ресурсы были доступны в случае необходимости. Достижение цели для важного адресного пространства может вызвать ухудшение показателя оборотного времени для менее важного адресного пространства. Таким образом, WLM должен принимать решения, являющиеся компромиссом между противоречащими целями.

Чтобы сбалансировать пропускную способность с показателями времени реагирования и оборотного времени, WLM выполняет следующие операции:

- мониторинг использования ресурсов различными адресными пространствами;
- мониторинг использования ресурсов в масштабе системы для определения полноты их использования;

- определение необходимости (и времени выполнения) свопинга адресных пространств;
- препятствование созданию новых адресных пространств или изъятие страниц при нехватке основной памяти;
- изменение диспетчерского приоритета адресных пространств, контролирующего коэффициент использования системных ресурсов адресными пространствами;
- выбор устройств для распределения, если есть выбор между устройствами, для балансировки использования устройств ввода-вывода.

Прочие компоненты z/OS, менеджеры транзакций и менеджеры баз данных могут сообщить WLM об изменениях состояния определенного адресного пространства (или системы в целом) или обратиться к функциям принятия решений, реализованным в WLM.

Например, WLM уведомляется в следующих случаях:

- при подключении или отключении основной памяти в системе;
- при создании адресного пространства;
- при удалении адресного пространства;
- при запуске или завершении свопинга адресного пространства;
- при выборе подпрограммами распределения ресурсов устройств, выделяемых для запроса.

До этого момента мы рассматривали WLM только в контексте одной системы z/OS. На практике в клиентских инсталляциях часто используются кластеры из нескольких взаимодействующих систем z/OS, совместно обрабатывающих сложные задачи. Можно вспомнить приведенное выше обсуждение кластерных систем z/OS (сисплекс).

Компонент WLM особенно хорошо подходит для использования в среде сисплекса. Он следит за использованием системы и достижением цели рабочей нагрузки по всем системам в Parallel Sysplex и средах совместного использования данных. Например, WLM может выбрать систему z/OS для выполнения пакетного задания, исходя из доступности ресурсов для быстрой обработки задания.

3.5.2 Как используется WLM?

Мэйнфрейм-инсталляция может влиять почти на все решения, принимаемые WLM, путем создания набора *политик*, позволяющих инсталляции связать системную производительность со своими бизнес-целями. Рабочей нагрузке назначаются цели (например, требуемое среднее время реагирования) и важность (определяющая, насколько она важна для предприятия, цели которого она выполняет).

Соглашение об уровне сервиса (SLA) – письменное соглашение об обслуживании, предоставляемое пользователям вычислительной системы

До появления WLM единственный способ информирования z/OS о бизнес-целях компании состоял в том, чтобы системный программист перевел абстрактные цели в сугубо технические тер-

мины, понимаемые системой. Такой перевод требовал высококвалифицированных специалистов и при этом был длительным, подверженным ошибкам, и в конечном итоге противоречил действительным бизнес-целям.

Кроме того, часто бывало сложно предсказать результаты необходимых изменений параметров системы, например после увеличения мощности системы. Это может вызвать несбалансированное выделение ресурсов, при котором задача будет лишена критических системных ресурсов. Такой режим работы, называвшийся режимом совместимости, становился неуправляемым при добавлении новых задач и совместном управлении несколькими системами.

При работе системы в целевом режиме WLM обеспечивает меньшее количество более простых и более согласованных внешних параметров системы, отражающих цели для задачи, выраженные в терминах, используемых для описания бизнес-целей; при этом WLM и менеджер управления ресурсами (System Resource Manager, SRM) выполняют сопоставление ресурсов для достижения этих целей путем постоянного мониторинга и подстройки системы. Менеджер управления рабочей нагрузкой представляет собой метод управления распределением рабочей нагрузки, балансировки рабочей нагрузки и распределения ресурсов между соревнующимися задачами.

Политики WLM часто основаны на соглашении об уровне сервиса (service level agreement, SLA), представляющим собой письменное соглашение об обслуживании информационных систем, предоставляемое пользователям вычислительной системы. WLM стремится достичь целей рабочих задач (времени реагирования), описанных в SLA, пытаясь достичь надлежащего распределения ресурсов без их чрезмерного выделения. В то же время WLM максимизирует использование системы (пропускную способность) для получения максимальной пользы от установленного оборудования и программной платформы.

3.6 Ввод-вывод и управление данными

Почти вся работа в системе предполагает ввод или вывод данных. В мэйнфрейме использованием устройств ввода-вывода (например, дисков, накопителей на магнитных лентах и принтеров) управляет канальная подсистема. Операционная система должна связывать данные определенной задачи с устройством и управлять созданием, размещением, мониторингом, миграцией, резервным копированием, возвратом, восстановлением и удалением файлов.

Эти операции по управлению данными можно выполнять либо вручную, либо с использованием автоматизированных процессов. При автоматизированном управлении данными система определяет размещение объектов и автоматически управляет резервным копированием, перемещением, пространством и безопасностью объектов. Стандартная рабочая система z/OS включает как ручные, так и автоматизированные процессы управления данными.

В зависимости от настройки системы z/OS и ее устройств хранения пользователь или программа может напрямую контролировать многие аспекты управления данными, и на заре развития операционных систем пользователи были обязаны это делать. Однако в настоящее время инсталляции z/OS все больше полагаются на параметры управления данными и ресурсами, характерные для инсталляции, а также на

дополнительные средства управления хранением, автоматизирующие использование хранилищ. Основным средством управления хранением в z/OS является компонент DFSMS, рассматриваемый в главе 5, «Работа с наборами данных».

3.7 Наблюдение за выполнением задач в системе

Для обеспечения мультипрограммирования z/OS требует использования множества средств наблюдения, в частности следующих:

- Средств обработки прерываний.
  Мультипрограммирование требует использования определенного метода переключения управления от одной программы к другой, чтобы, например, когда программа A ожидает выполнения запроса ввода-вывода, могла выполняться программа B. В z/OS такое переключение обеспечивается путем прерываний (interrupts), представляющих собой события, изменяющие последовательность выполнения инструкций процессором. При возникновении прерывания система сохраняет состояние выполнения прерванной подпрограммы, после чего анализирует и обрабатывает прерывание.
- Создания диспетчеризуемых единиц работы (dispatchable units of work).
  Для идентификации и отслеживания своей работы операционная система z/OS представляет каждую единицу работы управляющим блоком. Диспетчеризуемые единицы работы представлены двумя типами управляющих блоков: *блоками управления задачами* (task control blocks, TCB), представляющими задачи, выполняющиеся в адресном пространстве; *блоками запросов обслуживания* (service request blocks, SRB), представляющими системные службы с более высоким приоритетом.
- Диспетчеризации работы.
  После обработки прерываний операционная система определяет, какая единица работы (из всех единиц работы в системе) готова к выполнению и имеет наивысший приоритет, и передает управление этой единице работы.
- Синхронизации использования ресурсов.
  В системе мультипрограммирования почти любую последовательность инструкций можно прервать с последующим возобновлением выполнения. Если этот набор инструкций осуществляет управление или изменение ресурса (например, управляющего блока или файла данных), операционная система должна не допустить использования ресурса другими программами до тех пор, пока прерванная программа не завершит работу с ресурсом.

Существует несколько методов синхронизации использования ресурсов; наиболее распространенными являются *организация очередей (enqueuing)* и *блокировка (locking)*; третий метод называется *фиксацией (latching)*. Организация очередей доступна для всех пользователей, тогда как блокировка для синхронизации использования ресурсов доступна только подпрограммам с соответствующими полномочиями.

3.7.1 Что такое обработка прерываний?

Прерывание представляет собой событие, изменяющее последовательность выполнения инструкций процессором. Прерывание может быть запланированным (специ-

ально запрошенным выполняющейся программой) или незапланированным (вызванным событием, связанным или не связанным с выполняющейся программой); z/OS использует шесть типов прерываний, в частности:

- Вызовы супервизора или SVC-прерывания.
  Возникают, когда программа выдает SVC-прерывание для запрашивания определенной системной службы. SVC прерывает выполняемую программу и передает управление супервизору, чтобы он мог выполнять обслуживание. Программы запрашивают эти службы через макросы, такие, как OPEN (открытие файла), GETMAIN (получение памяти) или WTO (отправление сообщения оператору системы).
- Прерывания ввода-вывода.
  Эти прерывания возникают, когда канальная подсистема сигнализирует об изменении состояния, например о завершении операции ввода-вывода, возникновении ошибки или готовности устройства ввода-вывода (например, принтера) к работе.
- Внешние прерывания.
  Могут указывать на некоторые события, например на завершение временного интервала, нажатие оператором клавиши прерывания на консоли или на получение процессором сигнала от другого процессора.
- Прерывания рестарта.
  Возникают при выборе оператором команды рестарта на консоли или при получении инструкции перезапуска (SIGP, signal processor) от другого процессора.
- Программные прерывания.
  Вызываются ошибками программы (например, если программа пытается выполнить недопустимую операцию), событиями отсутствия страницы (если программа обращается к странице, отсутствующей в основной памяти) или запросами мониторинга события.
- Прерывания от схем контроля работы машины.
  Вызываются неисправностями машины.

При возникновении прерывания аппаратные средства сохраняют нужную информацию о прерванной программе и, если возможно, отключает реагирование процессора на последующие прерывания такого же типа. Затем аппаратные средства передают управление соответствующей подпрограмме обработки прерываний. Основным ресурсом в этом процессе является слово состояния программы.

Каким образом используется слово состояния программы?

Слово состояния программы (program status word, PSW) представляет собой 128-разрядную область данных в процессоре, которая наряду со множеством других типов регистров (управляющих регистров, регистров времени и регистров префикса), содержит сведения, критически важные как для аппаратного, так и для программного обеспечения. Текущее слово состояния программы содержит адрес следующей программной инструкции и контрольную информацию о выполняющейся программе. Каждый процессор имеет только одно текущее слово состояния программы. Таким образом, процессор может одновременно выполнять только одну задачу.

PSW контролирует порядок подачи инструкций в процессор и отображает состояние системы относительно текущей выполняющейся программы. Несмотря на то что каждый процессор имеет только одно слово состояния программы, для понимания обработки прерываний полезно рассматривать три типа PSW:

- текущее PSW,
- новое PSW,
- старое PSW.

Текущее PSW указывает следующую выполняемую инструкцию. Оно также указывает, включена ли в процессоре поддержка прерываний ввода-вывода, внешних прерываний, прерываний от схем контроля работы машины и некоторых программных прерываний. Если поддержка прерываний включена, могут возникать эти прерывания. Если поддержка прерываний отключена, эти прерывания игнорируются или остаются в режиме ожидания.

Существуют также новое PSW и старое PSW, связанные с каждым из шести типов прерываний. Новое PSW содержит адрес подпрограммы, которая может обрабатывать соответствующее прерывание. Если в процессоре включена поддержка прерываний, тогда при возникновении прерывания происходит переключение PSW с использованием следующего метода:

1. Сохранение текущего PSW в старом PSW, связанном с типом возникшего прерывания.
2. Загрузка содержимого нового PSW для возникшего прерывания в текущее PSW. Текущее PSW, указывающее следующую выполняемую инструкцию, теперь содержит адрес требуемой подпрограммы для обработки прерывания. Это переключение вызывает передачу управления требуемой подпрограмме обработки прерываний.

Регистры и PSW

Архитектура мэйнфрейма содержит регистры для слежения за происходящими событиями. PSW, например, представляет собой регистр, используемый для записи информации, необходимой для выполнения текущей активной программы. Мэйнфреймы содержат и другие регистры, в частности:

- *регистры доступа (access registers)* – используются для определения адресного пространства, в котором находятся данные;
- *общие регистры (general registers)* – используются для адресации данных в памяти, а также для хранения пользовательских данных;
- *регистры с плавающей точкой (floating point registers)* – используются для хранения численных данных в формате с плавающей точкой;
- *управляющие регистры (control registers)* – используются самой операционной системой, например для ссылки на таблицы трансляции.

Дополнительные сведения. Подробное описание аппаратных средств переключения состояния системы, включая состояния CPU, режимы управления, PSW и управляющие регистры, см. в руководстве *z/Architecture Principles of Operation*. Эту и другие публикации IBM можно найти на веб-сайте z/OS Internet Library

`http://www.ibm.com/servers/eserver/zseries/zos/bkserv/`

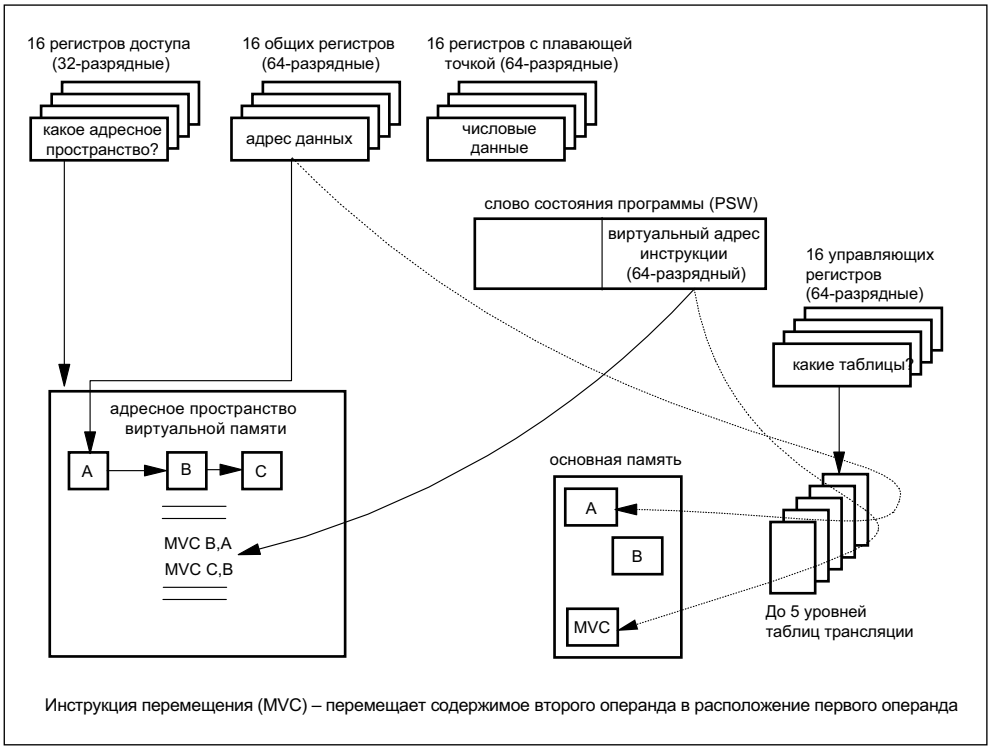


Рис. 3.10. Регистры и PSW

3.7.2 Создание диспетчеризуемых единиц работы

В z/OS диспетчеризуемые единицы работы представлены двумя видами управляющих блоков:

- блоками управления задачами (task control blocks, TCB).
  Представляют задачи, выполняющиеся в адресном пространстве, в частности пользовательские программы и системные программы, поддерживающие пользовательские программы.
- блоками запросов обслуживания (service request blocks, SRB).
  Представляют запросы на выполнение подпрограммы системного сервиса. SRB обычно создаются, когда одно адресное пространство обнаруживает событие, влияющее на другое адресное пространство; они обеспечивают единый механизм связи между адресными пространствами.

Что такое TCB?

TCB – управляющий блок, представляющий задачу, например вашу программу, при ее выполнении в адресном пространстве. TCB содержит информацию о запущенной задаче, в частности адрес всех созданных ею областей памяти. Не путайте термин

TCB, используемый в z/OS, со структурой данных PCB (process control block; блок управления процессом) в UNIX.

TCB создаются в ответ на макрос ATTACH. Выдавая макрос ATTACH, пользовательская программа или системная подпрограмма начинает выполнение программы, указанной в макросе ATTACH, в качестве подзадачи задачи, выдавшей ATTACH. Как подзадача данная программа может соревноваться за процессорное время и использовать некоторые ресурсы, уже выделенные для задачи, выдавшей ATTACH.

Задача управления регионом (region control task, RCT), отвечающая за подготовку адресного пространства к загрузке и выгрузке, является задачей наивысшего приоритета в адресном пространстве. Все задачи в адресном пространстве являются подзадачами RCT.

Что такое SRB?

SRB – управляющий блок, представляющий подпрограмму, выполняющую определенную функцию или службу в заданном адресном пространстве. Обычно SRB создается, когда выполняется одно адресное пространство и возникает событие, воздействующее на другое адресное пространство.

Подпрограмма, выполняющая функцию или службу, называется *SRB-подпрограммой (SRB routine)*; инициация процесса называется *планированием SRB (scheduling an SRB)*; SRB-подпрограмма выполняется в операционном режиме, называемом *режимом SRB (SRB mode)*.

SRB подобен TCB в том, что он определяет единицу работы в системе. В отличие от TCB SRB не может «владеть» областями памяти. SRB-подпрограммы могут получать, обращаться, использовать и освобождать области памяти, но этими областями должен владеть TCB. В многопроцессорной среде SRB-подпрограмма после планирования может быть передана на другой процессор и выполняться одновременно с программой, ее запланировавшей. Программа, запланировавшая SRB, может продолжать выполнять обработку других задач параллельно с подпрограммой SRB. Как говорилось выше, SRB представляет средство асинхронной связи между адресными пространствами для программ, выполняющихся в z/OS.

SRB могут создавать только программы, работающие в режиме более высокой авторизации, называемом *супервизорным режимом (supervisor state)*. Такие авторизованные программы получают память и инициализируют управляющий блок с такими элементами, как идентификатор целевого адресного пространства и указатели на код, обрабатывающий запрос. Программа, создающая SRB, затем выдает макрос SCHEDULE и указывает, имеет ли SRB глобальный (в масштабе системы) или локальный (в масштабе адресного пространства) приоритет. Система помещает SRB в нужную очередь диспетчера, где он остается до тех пор, пока не станет заданием с наивысшим приоритетом в очереди.

SRB с глобальным приоритетом имеют более высокий приоритет, чем SRB с приоритетом адресного пространства, независимо от действительного адресного пространства, в котором они выполняются. SRB с локальным приоритетом имеют приоритет, равный приоритету адресного пространства, в котором они выполняются, но более высокий, чем любой TCB в этом адресном пространстве. Назначение глобаль-

ного или локального приоритета зависит от «важности» запроса; например, SRB прерываний ввода-вывода планируются с глобальным приоритетом для минимизации задержки ввода-вывода.

Дополнительные сведения. Описание использования SRB см. в руководстве *z/OS MVS Authorized Assembler Services Guide*. Эту и другие публикации IBM можно найти на веб-сайте z/OS Internet Library

```
http://www.ibm.com/servers/eserver/zseries/zos/bkserv/
```

3.7.3 Вытесняемые и невытесняемые единицы работы

То, какая подпрограмма получит управление после обработки прерывания, зависит от того, была ли прерванная единица работы вытесняемой (preemptable). Если была, тогда операционная система определяет, какую единицу работы нужно выполнять следующей. Другими словами, система определяет, какая единица работы из всей работы в системе имеет наивысший приоритет, и передает управление в эту единицу работы.

Невытесняемая (non-preemptable) единица работы может быть прервана, но должна получить управление после обработки прерывания. Например, SRB часто являются невытесняемыми[1]. Другими словами, если подпрограмма, представленная невытесняемым SRB, прервана, после обработки прерывания она получит управление. С другой стороны, подпрограммы, представленные TCB, например пользовательские программы, обычно являются вытесняемыми[2]. Если ее выполнение прерывается, тогда после обработки прерывания управление возвращается операционной системе. Затем z/OS определяет, какая задача из всех готовых задач будет выполняться следующей.

3.7.4 Что делает диспетчер?

Выбор новой работы происходит, например, когда задача была прервана или стала недиспетчеризуемой (non-dispatchable) либо после выполнения SRB или его приостановки (другими словами, если SRB приостановлен в связи с недоступностью требуемого ресурса).

В z/OS компонент диспетчера отвечает за передачу управления единице работы с наивысшим приоритетом, готовой к выполнению. Процессы диспетчера работают в следующем порядке:

1. Специальные «выходы».

   Представляют собой «выходы» (exits) на подпрограммы, имеющие высокий приоритет из-за особых условий в системе. Например, при отказе одного из процессоров в многопроцессорной системе вызывается процесс восстановления альтернативного CPU посредством специального «выхода» для восстановления работы, выполнявшейся на отказавшем процессоре.
2. SRB, имеющие глобальный приоритет.
3. Готовые адресные пространства в порядке приоритета.

---

[1] Инициирующая программа может сделать SRB вытесняемым, что позволит задачам с равным или более высоким приоритетом получить доступ к процессору. Кроме того, клиентские SRB и SRB «анклавов» (enclave SRB) также являются вытесняемыми. Эти вопросы выходят за рамки данной книги.

[2] TCB является невытесняемым, если он выполняет SVC.

Адресное пространство готово к выполнению, если оно загружено и не ожидает завершения какого-либо события. Приоритет адресного пространства определяется диспетчерским приоритетом, определяемым пользователем или инсталляцией.

После выбора адресного пространства с наивысшим приоритетом z/OS (посредством диспетчера) сначала диспетчеризует SRB с локальным приоритетом, запланированные для этого адресного пространства, и затем TCB в этом адресном пространстве.

Если в системе нет работы, готовой к выполнению, z/OS переходит в режим *разрешенного ожидания (enabled wait)* до тех пор, пока новая работа не поступит в систему.

Различные модели оборудования z/Series могут иметь от одного до 54 центральных процессоров (CP)[1]. Все CP могут выполнять инструкции одновременно. Диспетчерские приоритеты определяют, когда следует начать диспетчеризацию готовых к выполнению адресных пространств.

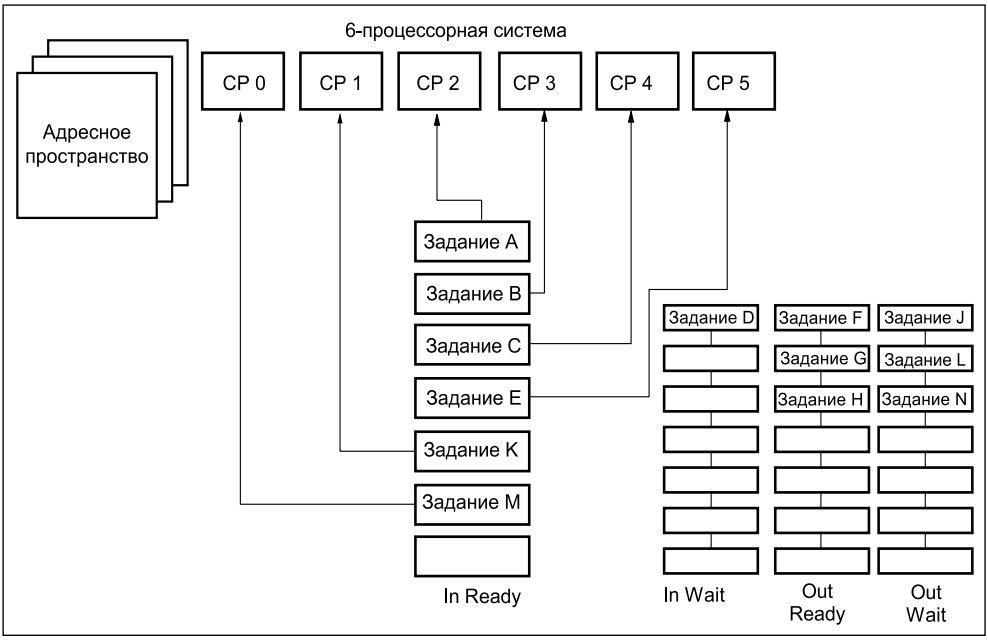


Рис. 3.11. Диспетчеризация работы

Адресное пространство может находиться в одной из четырех очередей:

- IN-READY – находится в основной памяти и ожидает диспетчеризации;
- IN-WAIT – находится в основной памяти, но ожидает завершения некоторого события;
- OUT-READY – готово к выполнению, но выгружено из основной памяти;
- OUT-WAIT – выгружено из основной памяти и ожидает завершения некоторого события.

Только задачи в очереди IN-READY можно выбрать для диспетчеризации.

[1] IBM z9-109 Model S54 может иметь до 54 центральных процессоров (номер модели соответствует максимальному количеству процессоров, которое можно установить на сервере).

3.7.5 Синхронизация использования ресурсов

В многозадачной многопроцессорной среде синхронизация использования ресурсов представляет собой метод координирования доступа к ресурсам, используемых несколькими приложениями. Программам, изменяющим данные, требуется исключительный доступ к этим данным. В противном случае, если несколько программ одновременно выполнят изменение одних и тех же данных, это может привести к повреждению данных (нарушению целостности данных). С другой стороны, программы, которым требуется только чтение данных, могут осуществлять одновременный доступ к одним и тем же данным.

Наиболее распространенными методами синхронизации использования ресурсов являются организация очередей (enqueuing) и блокировка (locking). Эти методы предусматривают упорядоченный доступ к системным ресурсам, требуемым нескольким пользователям в среде мультипрограммирования и мультипроцессирования. В z/OS организацией очередей управляет компонент глобальной синхронизации ресурсов и блокировкой управляет несколько программ управления блокировкой, входящих в супервизорный компонент.

Что такое глобальная синхронизация ресурсов?

Компонент глобальной синхронизации ресурсов обрабатывает запросы на доступ к ресурсам от программ, выполняющихся в z/OS. Глобальная синхронизация ресурсов осуществляет синхронизацию доступа к ресурсам для защиты их целостности. В инсталляции две или больше систем z/OS могут быть связаны адаптерами канал-канал (CTC) для создания GRS-комплекса и синхронизации доступа к ресурсам, совместно используемым различными системами.

При получении программными запросами доступа к многократно используемому ресурсу этот доступ может быть исключительным либо разделяемым. Когда компонент глобальной синхронизации ресурсов назначает разделяемый доступ к ресурсу, пользователи, требующие исключительного доступа, не могут получить доступ к этому ресурсу. Подобным же образом, когда компонент глобальной синхронизации ресурсов назначает исключительный доступ к ресурсу, все остальные инициаторы запросов на доступ к ресурсу должны ожидать до тех пор, пока инициатор запроса с исключительным доступом не освободит ресурс.

Что такое организация очередей?

Организация очередей (enqueuing) является средством запрашивания программой, выполняющейся в z/OS, управления ресурсами с последовательным многократным использованием. Организацией очередей занимаются макросы ENQ (enqueue) и DEQ (dequeue), доступные для всех программ, запущенных в системе. Для устройств, разделяемых между несколькими системами z/OS, организация очередей выполняется макросами RESERVE и DEQ.

В макросах ENQ и RESERVE программа задает имена одного или нескольких ресурсов и запрашивает разделяемое или исключительное управление этими ресурсами. Если требуется внести изменения в ресурсы, программа должна запросить ис-

ключительное управление; если в ресурсы не требуется вносить изменений, программе *следует* запросить разделяемое управление, позволяющее осуществлять доступ совместно с другими программами, не требующими исключительного управления. Если ресурс недоступен, система приостанавливает выполнение запрашивающей программы до тех пор, пока ресурс не освободится. Когда программе больше не требуется управление ресурсом, она использует макрос DEQ для его освобождения.

Что такое блокировка?

Посредством блокировки (locking) система осуществляет синхронизацию использования системных ресурсов авторизованными подпрограммами и (в Parallel Sysplex) процессорами. Блокировка (lock) представляет собой именованное поле в памяти, указывающее, используется ли ресурс и кто его использует. В z/OS существует два типа блокировок: глобальные блокировки, предназначенные для ресурсов, связанных с несколькими адресными пространствами, и локальные блокировки, предназначенные для ресурсов, назначенных определенному адресному пространству. Глобальные блокировки накладываются для подпрограмм, не допускающих многократное или совместное использование, и различных ресурсов.

Для того чтобы использовать ресурс, защищенный блокировкой, подпрограмма должна сначала запросить блокировку для этого ресурса. Если блокировка недоступна (другими словами, если она уже получена другой программой или процессором), то программа или процессор, запросившие блокировку, предпринимают действие, зависящее от того, используется ли *спин-блокировка (spin lock)* или *отсроченная блокировка (suspend lock)*:

- Если спин-блокировка недоступна, запрашивающий процессор продолжает проверять доступность блокировки до тех пор, пока другой процессор ее не освободит. Как только блокировка освобождена, запрашивающий процессор может получить блокировку и таким образом получить управление защищенным ресурсом. Большинство глобальных блокировок являются спин-блокировками. Держатель спин-блокировки должен быть недоступен для большинства прерываний (если прервать его выполнение, он может больше никогда не получить управление, чтобы освободить блокировку).
- Если отсроченная блокировка недоступна, единица работы, запрашивающая блокировку, приостанавливается до тех пор, пока блокировка не будет доступна. На запрашивающий процессор передается другая работа. Все локальные блокировки являются отсроченными.

Вам может быть интересно, что случится, если каждый из двух пользователей запросит блокировку, удерживаемую другим? Если каждому из них придется ждать вечно, пока другой не освободит блокировку первым, возникает тупиковая ситуация. В z/OS такая ситуация называется тупиковой ситуацией (deadlock). К счастью, используемая в z/OS методология блокировки не допускает возникновения тупиковых ситуаций.

Для того чтобы избежать тупиковой ситуации, блокировки организованы в иерархию и процессор или подпрограмма может запросить только блокировки, расположенные выше по иерархии, чем блокировки, удерживаемые ей на данный момент. Например, тупиковая ситуация может возникнуть, если процессор 1 удерживает бло-

кировку A и запрашивает блокировку B, тогда как процессор 2 удерживает блокировку B и запрашивает блокировку A. Возникновение такой ситуации невозможно, так как блокировки можно получать только в иерархической последовательности. Предположим, в данном примере, что блокировка A предшествует блокировке B в иерархии. В этом случае процессор 2 не сможет безусловно запросить блокировку A, удерживая блокировку B. Вместо этого он должен сначала освободить блокировку B, запросить блокировку A и затем запросить блокировку B. При такой иерархии возникновение тупиковой ситуации невозможно.

Дополнительные сведения. Публикация *z/OS Diagnosis Reference* содержит таблицу, в которой представлена иерархия блокировок z/OS вместе с их описаниями и свойствами.

3.8 Определяющие свойства z/OS

Ниже представлены определяющие свойства z/OS.

- Использование адресных пространств в z/OS имеет много преимуществ: изоляция приватных областей в разных адресных пространствах обеспечивает безопасность системы, хотя каждое адресное пространство содержит также общую область, доступную для всех адресных пространств.
- Система разработана таким образом, чтобы обеспечивать *целостность данных*, независимо от числа пользователей; z/OS не позволяет пользователям осуществлять доступ или изменение каких-либо объектов в системе, включая пользовательские данные, какими либо средствами, кроме системных интерфейсов, применяющих правила полномочий.
- Система разработана таким образом, чтобы осуществлять управление большим количеством параллельных пакетных заданий, не требуя внешнего управления балансировкой рабочей нагрузки или устранения проблем целостности данных, которые в противном случае могут возникнуть из-за одновременного и противоречивого использования набора данных.
- Дизайн системы безопасности распространяется как на системные функции, так и на обычные файлы. Средства безопасности можно внедрить в приложения, ресурсы и профили пользователей.
- Система позволяет использовать несколько коммуникационных подсистем одновременно, обеспечивая необычную гибкость в одновременном выполнении разнообразных коммуникационных приложений (при использовании множества тестовых, рабочих и резервных версий каждого из них). Например, несколько стеков TCP/IP могут работать одновременно, каждый из которых имеет разные IP-адреса и обслуживает различные приложения.
- Система обеспечивает несколько уровней восстановления программного обеспечения, что делает незапланированные перезапуски системы в рабочей среде очень редкими. Системные интерфейсы позволяют приложениям обеспечивать собственные уровни восстановления. Эти интерфейсы редко используются простыми приложениями, однако часто используются полнофункциональными приложениями.

- Система разработана таким образом, чтобы осуществлять постоянное управление очень разнообразными задачами с автоматической балансировкой ресурсов для обеспечения соответствия рабочим требованиям, установленным системным администратором.
- Система разработана таким образом, чтобы постоянно управлять большими конфигурациями ввода-вывода, которые могут распространяться на тысячи дисковых приводов, множество автоматизированных библиотек магнитных лент, множество больших принтеров, больших сетей терминалов и т. д.
- Система контролируется с одного или нескольких терминалов операторов или через интерфейсы программирования приложений (API), позволяющие автоматизировать стандартные функции оператора.
- Интерфейс оператора является критической функцией z/OS. Он предоставляет информацию о состоянии, сообщения об исключительных ситуациях, средства управления потоком заданий, управление аппаратными устройствами и позволяет оператору управлять нештатными ситуациями восстановления.

3.9 Дополнительные программные продукты для z/OS

Система z/OS обычно содержит дополнительные, отдельно продаваемые продукты, необходимые для создания полноценной рабочей системы. Например, рабочая система z/OS обычно включает средство управления безопасностью и средство управления базами данных. Говоря о z/OS, люди часто предполагают наличие этих дополнительных продуктов. Это обычно понятно из контекста обсуждения, однако иногда может быть необходимо спросить, является ли определенная функция частью «базовой z/OS», или же она представляет собой дополнительный продукт. IBM называет свои дополнительные продукты *лицензированными программами IBM (IBM licensed programs)*.

Лицензированная программа – дополнительный, отдельно продаваемый программный продукт, не являющийся частью базовой z/OS

В условиях, когда многие независимые изготовители программного обеспечения (ISV) предлагают множество продуктов с несколько различающимися, но похожими функциями, в частности средствами управления безопасностью и средствами управления базами данных, возможность выбрать требуемые лицензированные программы для выполнения задач значительно повышает гибкость операционной системы z/OS и позволяет группе, отвечающей за эксплуатацию мэйнфрейма, настраивать используемые ей продукты для соответствия требованиям компании.

Мы не будем перечислять все лицензированные программы z/OS в этой книге (их сотни); чаще всего выбирать приходится из следующих:

- Система безопасности.
  z/OS предоставляет инфраструктуру, в которой клиенты могут усиливать безопасность путем добавления продукта управления безопасностью (лицензированная программа компании IBM называется Resource Access Control Facility или RACF®). Доступны также лицензированные программы других разработчиков.

- Компиляторы.
  z/OS включает компилятор ассемблера и C. Другие компиляторы, например компиляторы COBOL и PL/1, распространяются как отдельные продукты.
- Реляционная база данных.
  Примером такой системы является DB2. Доступны также другие продукты, например системы управления иерархическими базами данных.
- Средство обработки транзакций.
  IBM предлагает несколько таких средств, в том числе:
  - Customer Information Control System (CICS);
  - Information Management System (IMS);
  - WebSphere Application Server для z/OS.
- Программа сортировки.
  Быстрая и эффективная сортировка больших объемов данных очень важна в пакетной обработке. IBM и другие изготовители предлагают полнофункциональные системы сортировки.
- Большое количество служебных программ.
  Например, программа System Display and Search Facility (SDSF), широко используемая нами в этой книге для просмотра выходных данных пакетных заданий, является лицензированной программой. Не каждая инсталляция содержит SDSF; существуют и альтернативные продукты.

Существует множество других продуктов от различных *независимых изготовителей программного обеспечения (independent software vendors, ISV)*, как они часто называются в отрасли.

3.10 Промежуточное программное обеспечение для z/OS

Промежуточное программное обеспечение – программное обеспечение, обеспечивающее важные функции, отсутствующие в операционной системе

Промежуточное программное обеспечение обычно представляет собой уровень между операционной системой и конечным пользователем или пользовательскими приложениями. Оно обеспечивает важные функции, отсутствующие в операционной системе. Этот термин часто относится к крупным программным продуктам, таким, как менеджеры баз данных, мониторы транзакций, веб-серверы и т. д. Этот тип программного обеспечения также часто обозначается термином *подсистема (subsystem)*. К нему обычно относятся лицензированные программы, хотя есть и значительные исключения, например HTTP Server.

z/OS является основой для использования многих программных продуктов и функций промежуточного уровня. Часто используется множество разных программных функций промежуточного уровня, некоторые из которых запускаются в нескольких экземплярах. Повседневное использование разнообразных задач (сочетания пакетных задач, задач по обработке транзакций, задач веб-сервера, запросов и обновлений баз данных и т. д.) является одним из свойств z/OS.

К промежуточному программному обеспечению в z/OS относятся:

- системы управления базами данных;
- веб-серверы;
- функции организации очередей и маршрутизации сообщений;
- менеджеры транзакций;
- виртуальные машины Java;
- функции обработки XML.

Программные продукты промежуточного уровня часто включают интерфейс программирования приложений (API). В некоторых случаях приложения написаны таким образом, чтобы выполняться полностью под управлением этого API промежуточного уровня, тогда как в других случаях они используются для необычных целей. К примерам API промежуточного уровня на мэйнфреймах относятся:

- Набор продуктов WebSphere, включающий полный API, переносимый между различными операционным системами. Среди них WebSphere MQ поддерживает кроссплатформенные API и межплатформенную передачу сообщений.
- Система управления базами данных DB2, содержащая API (выраженный в языке SQL), используемый со многими другими языками и приложениями.

Веб-сервер считается продуктом промежуточного уровня, и веб-программирование (веб-страницы, CGI и т. д.) в значительной степени привязано к интерфейсам и стандартам, представленным веб-сервером, а не к интерфейсам, представленным операционной системой. Java является еще одним примером создания приложений для выполнения в среде виртуальной машины Java (Java Virtual Machine, JVM™)[1], в значительной мере независимых от используемой операционной системы.

3.11 Краткое сравнение z/OS и UNIX

Что бы мы обнаружили, если бы сравнили z/OS и UNIX? Во многих случаях мы нашли бы довольно много понятий, понятных для пользователей обеих операционных систем, несмотря на различия в терминологии.

Для опытных пользователей UNIX в табл. 3.1 представлена небольшая подборка знакомых компьютерных терминов и понятий. Для нового пользователя z/OS многие термины z/OS могут звучать незнакомо. Однако в процессе изучения этой книги вы найдете объяснения понятий, используемых в z/OS, и увидите, что многие элементы, используемые в UNIX, имеют аналоги в z/OS.

Для пользователей UNIX, переходящих на z/OS, основное различие состоит в том, что пользователь является лишь одним из *множества* других пользователей. При переходе с системы UNIX в среду z/OS пользователи часто задают такие вопросы, как: «можно ли мне иметь пароль пользователя root, так как мне нужно?» или «можно ли изменить то или другое и перезапустить систему?». Новым пользователям z/OS важно понимать, что в одной системе могут работать тысячи пользователей и что все действия пользователей и перезапуски системы в z/OS и z/OS UNIX тщательно контролируются во избежание неблагоприятного воздействия на других пользователей и приложения.

[1] JVM не связана с виртуальными машинами, создаваемыми z/VM.

В z/OS не существует единого пароля root или пользователя root. Идентификаторы пользователей являются внешними для системных служб z/OS UNIX. Идентификаторы пользователей хранятся в базе данных безопасности, используемой совместно с функциями UNIX и отличными от UNIX в системе z/OS, и возможно даже используемой совместно с другими системами z/OS. Обычно некоторые идентификаторы пользователей имеют полномочия root, но они остаются отдельными идентификаторами пользователей с отдельными паролями. Кроме того, некоторые идентификаторы пользователей не имеют полномочий root, но могут их включить, если это будет необходимо.

И z/OS и UNIX содержат API, позволяющие обеспечить совместное использование данных в памяти между процессами. В z/OS пользователь может осуществлять доступ к адресным пространствам другого пользователя посредством *служб межпространственной связи*. Подобным образом в UNIX существует понятие функций разделяемой памяти (Shared Memory functions), которые можно использовать в UNIX без специальных полномочий.

Однако службы межпространственной связи z/OS требуют, чтобы инициирующая программа имела специальные полномочия, контролируемые средством APF (Authorized Program Facility). Этот метод позволяет обеспечить эффективный и безопасный доступ к данным, принадлежащим другим пользователям, к данным, принадлежащим пользователю, но для удобства хранящимся в другом адресном пространстве, а также быструю и защищенную связь с такими службами, как менеджеры транзакций и менеджеры баз данных.

Таблица 3.1. Сравнение терминов и понятий UNIX и z/OS

Термин или понятие	UNIX	z/OS
Запуск операционной системы	Загрузка (boot) системы	Начальная загрузка (IPL) системы
Виртуальная память, выделяемая для каждого пользователя системы	Пользователи получают столько виртуальной памяти, сколько им нужно, в пределах возможностей оборудования и операционной системы	Пользователи получают адресное пространство – диапазон адресов виртуальной памяти до 2 Гб (или даже до 16 Эб), хотя в этой же памяти хранится системный код, общий для всех пользователей
Хранение данных	Файлы	Наборы данных (иногда называемые файлами)
Формат данных	Ориентированный на байты; организация данных осуществляется приложением	Ориентированный на записи; часто используются 80-байтовые записи, соответствующие формату перфокарт
Данные системной конфигурации	Управление свойствами осуществляет файловая система/etc	Параметры в PARMLIB управляют начальной загрузкой системы и функционированием адресных пространств

Окончание табл. 3.1.

Термин или понятие	UNIX	z/OS
Языки создания скриптов	Скрипты shell, Perl, awk и другие языки	CLISTS (command lists, командные списки) и исполняемые модули REXX
Наименьший элемент, выполняющий работу	Поток. Ядро поддерживает много потоков	Задача или блок запросов обслуживания (SRB). Базовая управляющая программа z/OS поддерживает множество задач и SRB
Единица работы длительного выполнения	Демон	Запускаемая задача или долгосрочное задание; часто является подсистемой z/OS
Порядок поиска системой программ для выполнения	Программы загружаются из файловой системы в соответствии с пользовательской переменной окружения PATH (список каталогов, в которых следует выполнять поиск)	Система ищет программу, которую следует загрузить, в следующих библиотеках: TASKLIB, STEPLIB, JOBLIB, LPALST и linklist
Интерактивные инструменты, предоставляемые операционной системой (не считая интерактивных приложений, которые могут быть добавлены позже)	Пользователи выполняют вход (log in) в системы и запускают сеансы оболочки в среде shell. Они могут использовать команды `rlogin` или `telnet` для подключения к системе. Каждый пользователь может иметь несколько открытых сеансов подключения одновременно	Пользователи подключаются (log on) к системе посредством TSO/E и его панельного интерфейса ISPF. Идентификатор пользователя может иметь только один активный сеанс подключения – TSO/E. Пользователи также могут подключаться к среде оболочки z/OS UNIX, используя telnet, rlogin или ssh
Редактирование данных или кода	Существует множество редакторов, например vi ed, sed и emacs	Редактор ISPF[1]
Источник и приемник входных и выходных данных	stdin и stdout	SYSIN и SYSOUT SYSUT1 и SYSUT2 используются утилитами. SYSTSIN и SYSTSPRT применяются пользователями TSO/E
Управление программами	Команда shell `ps` позволяет пользователям просматривать процессы и потоки, тогда как команда `kill` позволяет удалять процессы	SDSF дает возможность пользователям просматривать и завершать свои задания.

[1] Существует также редактор TSO, который однако редко используется. Например, при отправке электронной почты через TSO исполняемый модуль SENDNOTE открывает сеанс TSO EDIT, через который пользователь может создать сообщение электронной почты.

3.12 Заключение

Операционная система представляет собой набор программ, управляющих внутренней работой компьютерной системы. В этой книге изучается z/OS – широко используемая операционная система для мэйнфреймов. Технологии мультипрограммирования и мультипроцессирования, а также способность осуществлять доступ и управлять огромными объемами памяти и множеством операций ввода-вывода делают z/OS идеальной операционной системой для обработки задач для мэйнфреймов.

Понятие виртуальной памяти является ключевым в z/OS. Виртуальная память является иллюзией, создаваемой архитектурой, что система как будто имеет больше памяти, чем в действительности. Виртуальная память создается посредством использования таблиц отображения страниц виртуальной памяти на фреймы основной памяти или на слоты вспомогательной памяти. В основную память загружаются только те части программы, которые нужны на данный момент; z/OS хранит неактивные фрагменты адресных пространств во вспомогательной памяти.

z/OS построена на использовании адресных пространств, представляющих собой диапазоны адресов в виртуальной памяти. Каждый пользователь z/OS получает адресное пространство, содержащее одинаковый диапазон адресов памяти. Использование адресных пространств в z/OS позволяет изолировать приватные области в разных адресных пространствах в целях системной безопасности и при этом осуществлять совместное использование программ и данных между адресными пространствами посредством общей области, доступной для каждого адресного пространства.

В общеупотребительном смысле термины «центральная память», «реальная память», «реальное хранилище» и «основная память» используются как взаимозаменяемые. Подобным же образом термины «виртуальная память» и «виртуальное хранилище» также являются синонимами.

Количество основной памяти, необходимой для поддержки виртуальной памяти в адресном пространстве, зависит от рабочего набора используемого приложения, который может изменяться со временем. Пользователь не получает автоматически доступ ко всей виртуальной памяти в адресном пространстве. Запросы на использование диапазона виртуальной памяти проверяются на ограничения размера, после чего происходит построение требуемых записей таблицы страниц для создания требуемой виртуальной памяти.

Программы, запускаемые в z/OS и на мэйнфреймах zSeries, могут использовать 24-, 31- или 64-разрядную адресацию (и при необходимости могут выполнять переключение между этими режимами). Программы могут использовать набор инструкций с 16-, 32-или 64-разрядными операндами и при необходимости переключаться между ними.

Операционные системы мэйнфреймов редко обеспечивают полную операционную среду. Они используют лицензированные программы в качестве промежуточного программного обеспечения и для реализации других функций. Многие изготовители, включая IBM, предлагают промежуточное программное обеспечение и различные утилиты.

Промежуточное программное обеспечение является сравнительно новым термином, который может включать несколько понятий одновременно. Основное свойство

промежуточного программного обеспечения состоит в том, что оно обеспечивает интерфейс программирования, и создаваемые приложения разрабатываются (или частично разрабатываются) для этого интерфейса.

Основные термины в этой главе			
Адресное пространство	Адресуемость	Вспомогательная память	Основная память
Управляющий блок	Динамическая трансляция адреса (DAT)	Фрейм	Ввод-вывод (I/O)
Лицензионная программа	Промежуточное программное обеспечение	Мультипрограммирование	Мультипроцессирование
Страница/страничный обмен	Изъятие страницы	Соглашение об уровне сервиса (SLA)	Слот
Свопинг	Виртуальная память	Управление рабочей нагрузкой (WLM)	z/OS

3.13 Контрольные вопросы

Чтобы проверить понимание материала этой главы, ответьте на следующие вопросы:

1. В чем отличие z/OS от однопользовательской операционной системы? Приведите два примера.
2. Для какой архитектуры была разработана z/OS? В каком году эта архитектура появилась?
3. Перечислите три основных типа памяти, используемых в z/OS.
4. В чем «виртуальность» виртуальной памяти?
5. Свяжите следующие термины:
   а) страница __ вспомогательная память;
   б) фрейм __ виртуальная память;
   в) слот __ основная память.
6. Какую роль играет WLM в системе z/OS?
7. Перечислите несколько определяющих свойств операционной системы z/OS.
8. Перечислите три типа программных продуктов, которые могут быть добавлены в z/OS для реализации полнофункциональной системы.
9. Перечислите несколько различий и сходств между операционными системами z/OS и UNIX.
10. Что из нижеперечисленного не считается промежуточным программным обеспечением в системе z/OS:

    а) веб-серверы,

    б) менджеры транзакций,

    в) менеджеры баз данных,

    г) менеджер вспомогательной памяти?

3.14 Темы для дальнейшего обсуждения

Дальнейшее изучение понятий z/OS может включать следующие темы для обсуждения:

1. z/OS поддерживает 64-разрядную адресацию. Предположим, что вы решили использовать эту возможность для работы с большой областью виртуальной памяти. Вы используете соответствующий программный интерфейс для получения, допустим, 30-гигабайтовой области виртуальной памяти и создаете цикл для инициализации этой области для своего приложения. Каковы возможные побочные эффекты этих действий? Когда такая схема целесообразна? Какие внешние обстоятельства следует учитывать? В чем заключалось бы различие на другой платформе, например в UNIX?
2. В чем может заключаться сложность для владельцев приложений при перемещении программ и блоков данных из области ниже «линии» в область выше «линии»? Как можно выполнить такое перемещение без нарушения совместимости с существующими приложениями?
3. Приложение можно разработать таким образом, чтобы оно выполнялось в 24-, 31- или 64-разрядном режиме адресации. Как программисту выбрать режим (в высокоуровневом языке, в ассемблере)? Вы начали использовать ISPF. В каком режиме адресации он работает?
4. Ускорит ли больший объем основной памяти работу системы? Какие показатели указывают на необходимость большего объема основной памяти? Когда больший объем основной памяти не нужен? Что может изменить эту ситуацию?
5. Если современная система z/OS работает только в режиме z/Architecture, почему говорится о 24-, 31- и 64-разрядных операциях? Почему говорится о 32-разрядных операндах?
6. Зачем заниматься выделением виртуальной памяти? Почему бы не построить все нужные таблицы страниц для всей виртуальной памяти при первоначальном создании адресного пространства?
7. Зачем нужны лицензированные программы? Почему бы просто не включить все программное обеспечение в состав операционной системы?

4

TSO/E, ISPF и UNIX: интерактивные средства z/OS

Цель. При работе с операционной системой z/OS вам потребуется иметь знание интерфейсов конечного пользователя. Главным из них является TSO и его интерфейс ISPF на основе меню. Эти программы позволяют подключаться к системе, запускать программы и управлять файлами данных. Кроме того, вам потребуется знать интерактивные средства реализации UNIX-интерфейсов в z/OS, называемые системными службами z/OS UNIX (z/OS UNIX System Services). После завершения работы над этой главой вы сможете:

- подключаться к z/OS;
- запускать программы из приглашения TSO READY;
- переходить по опциям меню ISPF;
- использовать редактор ISPF для внесения изменений в набор данных;
- использовать UNIX-интерфейсы в z/OS, включая командную оболочку z/OS UNIX.

4.1 Как происходит взаимодействие с z/OS?

Мы уже говорили, что z/OS идеально подходит для обработки пакетных заданий, выполняющихся в фоновом режиме с минимальным вмешательством человека или вообще без него. Однако z/OS настолько же является интерактивной операционной системой, насколько и системой пакетной обработки. Под словом *интерактивная* подразумевается, что конечные пользователи (в z/OS одновременно могут работать десятки тысяч пользователей) могут использовать систему посредством прямого взаимодействия, например посредством команд и пользовательских интерфейсов на основе меню.

z/OS содержит множество средств, позволяющих пользователям напрямую взаимодействовать с операционной системой. Эта глава содержит обзор каждого из перечисленных средств.

- В разделе «Обзор TSO» описывается, как осуществляется подключение к z/OS, а также рассматривается использование ограниченного набора базовых команд TSO, входящих в основную операционную систему. Взаимодействие с z/OS таким способом вызывается с использованием TSO в его *собственном режиме (native mode)*.
- В разделе «Обзор ISPF» представлена система ISPF на основе меню, которую многие используют исключительно для выполнения работы в z/OS. Меню ISPF содержат функции, наиболее часто используемые оперативными пользователями.
- В разделе «Интерактивные интерфейсы z/OS UNIX» описывается оболочка и утилиты z/OS UNIX. Это средство позволяет пользователям создавать и вызывать скрипты оболочки и утилиты и применять язык программирования оболочки.

В конце главы приведены практические упражнения, призванные помочь студентам разобраться в этих важных средствах.

4.2 Обзор TSO

Time Sharing Option/Extensions (TSO/E) позволяет пользователям создать интерактивный сеанс подключения к системе z/OS. TSO[1] содержит возможность однопользовательского входа и базовый интерфейс командной строки для работы с z/OS.

Подключение – процедура запуска терминального сеанса пользователем

Большинство пользователей работают с TSO через его интерфейс на основе меню ISPF (Interactive System Productivity Facility). Этот набор меню и панелей предлагает широкий диапазон функций, помогающих пользователям работать с файлами данных в системе. К пользователям ISPF относятся системные программисты, программисты приложений, администраторы и остальные пользователи, осуществляющие доступ к z/OS. В целом TSO и ISPF упрощают людям с различным уровнем компетентности взаимодействие с системой z/OS.

В системе z/OS каждому пользователю назначается идентификатор пользователя и пароль, авторизованный для подключения к TSO. Подключение к TSO требует ис-

[1] Большинство пользователей z/OS вместо «TSO/E» говорят «TSO», и именно такое название используется в этой книге. Кроме того, слово «пользователь» является тождественным словосочетанию «конечный пользователь».

пользования дисплейного устройства 3270 или, что чаще, эмулятора TN3270, запущенного на персональном компьютере.

Эмуляция 3270 – использование программного обеспечения, позволяющего эмулировать дисплейную станцию IBM 3270 или принтер на клиенте и применять функции главной системы

При подключении к TSO система выводит экран подключения TSO на пользовательское дисплейное устройство 3270 или в эмуляторе TN3270. Экран подключения имеет то же назначение, что и панель входа Windows.

Системные программисты z/OS часто изменяют расположение текста и информацию на панели входа TSO для более полного соответствия требованиям пользователей системы. Поэтому образы экранов, представленные в этой книге, скорее всего, отличаются от того, что вы увидите в реальной рабочей системе.

На рис. 4.1 представлен пример экрана подключения TSO.

```
------------------------------ TSO/E LOGON ----------------------------------

   Enter LOGON parameters below:                RACF LOGON parameters:

  Userid    ===> ZPROF

   Password  ===>                               New Password ===>

   Procedure ===> IKJACCNT                      Group Ident  ===>

   Acct Nmbr ===> ACCNT#

   Size      ===> 860000

   Perform   ===>

   Command   ===>

   Enter an 'S' before each option desired below:
           -Nomail         -Nonotice        -Reconnect        -OIDcard

PF1/PF13 ==> Help    PF3/PF15 ==> Logoff    PA1 ==> Attention    PA2 ==> Reshow
You may request specific help information by entering a '?' in any entry field
```

Рис. 4.1. Пример экрана подключения TSO/E

Многие образы экранов, используемые в этой книге, содержат настройки клавиш программных функций (PF). В организациях, использующих z/OS, часто выполняется собственное назначение клавиш программных функций для своих нужд, поэтому назначения клавиш, представленные в этой книге, могут не соответствовать назначениям клавиш, используемым в вашей организации. Список назначений клавиш программных функций, употребляемых в этой книге, представлен в разделе 4.3.1, «Назначения клавиш, используемые в этой книге».

4.2.1 Терминология файлов данных

Файлы z/OS называются *наборами данных (data sets)*. Прежде чем осуществлять запись данных в них, место для наборов данных должно быть зарезервировано на диске. Выбор количества дискового пространства, равно как и его форматирование, осуществляется пользователем.

Создание файла на мэйнфрейме является несколько более сложным процессом, чем на персональном компьютере. Дело не в использовании старой технологии; существуют достаточные основания для различий. Одно из различий состоит в том, что z/OS традиционно использует файловую систему, ориентированную на записи. В то же время операционные системы персональных компьютеров (Microsoft Windows, Linux, Mac OS и т. д.) используют байт-ориентированные файловые системы.

Запись – набор связанных данных, слов или полей, воспринимаемых как единое целое

В чем различие? В байт-ориентированной файловой системе файлы представляют собой наборы последовательных битов и используется специальный символ, показывающий компьютеру, где заканчивается одна строка (или *запись*) и начинается другая. В файловой системе, ориентированной на записи, файлы организуются на диске в виде отдельных записей. При использовании файлов, ориентированных на записи, можно явным образом определять размеры и атрибуты своих записей, поэтому не требуется использовать специальный символ конца строки, что позволяет экономить системные ресурсы. Тем не менее z/OS поддерживает особые байт-ориентированные файловые системы HFS и zFS; более подробно они рассматриваются в разделе 5.13 «Файловые системы z/OS UNIX».

Ниже приведены некоторые термины, используемые при *распределении (allocating)* набора данных.

Номер тома	Шестисимвольное имя дискового тома или тома на магнитной ленте, например TEST01
Тип устройства	Модель или тип дискового устройства, например 3390
Организация	Метод обработки набора данных, например последовательный
Формат записи	Данные хранятся в виде фрагментов, называемых записями, имеющих постоянную или переменную длину
Длина записи	Длина (количество символов) в каждой записи
Размер блока	При объединении записей в целях экономии пространства определяет длину блока в символах
Экстент	Область пространства для хранения данных. При заполнении первичного экстента операционная система автоматически выделяет экстенты, называемые вторичными
Пространство	Выделение дискового пространства происходит в единицах, называемых блоками, дорожками или цилиндрами

4.2.2 Использование команд TSO в собственном режиме

Большинство организаций, использующих z/OS, предпочитают, чтобы пользовательский сеанс TSO автоматически переключался в интерфейс ISPF после подключения к TSO. Однако в этом разделе мы кратко рассмотрим ограниченный набор базовых команд TSO, доступных без использования дополнительных программ, таких, как ISPF.

Такой режим использования TSO называется *собственным режимом (native mode)*.

Собственный режим – использование TSO без дополнительных программ, таких, как ISPF

Когда пользователь подключается к TSO, система z/OS отвечает отображением приглашения READY и ожидает ввода, как показано на рис. 4.2.

```
ICH70001I ZPROF  LAST ACCESS AT 17:12:12 ON THURSDAY, OCTOBER 7, 2004
ZPROF LOGON IN PROGRESS AT 17:12:45 ON OCTOBER 7, 2004
You have no messages or data sets to receive.
READY
```

Рис. 4.2. Приглашение READY при подключении к TSO

Приглашение READY принимает простые строчные команды, такие, как HELP, RENAME, ALLOCATE и CALL. На рис. 4.3 показан пример команды ALLOCATE, создающей *набор данных* (файл) на диске.

```
READY
 alloc dataset(zschol.test.cntl) volume(test01) unit(3390) tracks space(2,1)
recfm(f) lrecl(80) dsorg(ps)
READY
listds
 ENTER DATA SET NAME -
zschol.test.cntl
 ZSCHOL.TEST.CNTL
  --RECFM-LRECL-BLKSIZE-DSORG
    F     80    80      PS
 --VOLUMES--
   TEST01
 READY
```

Рис. 4.3. Распределение набора данных из командной строки TSO

Собственный режим TSO подобен интерфейсу приглашения DOS. TSO также содержит очень простой строчный редактор, в отличие от ISPF, содержащего полноэкранный редактор.

Рис. 4.4 содержит еще один пример строчных команд, которые пользователь может ввести в приглашении READY. Здесь пользователь вводит команды для сортировки данных.

В этом примере пользователь ввел несколько команд TSO ALLOCATE для назначения входных и выходных файлов на рабочей станции для программы сортировки. Затем пользователь ввел одну команду CALL для запуска программы сортировки DFSORT™, являющейся дополнительным программным продуктом компании IBM.

В командах ALLOCATE используется следующее содержимое (задаваемое операндом DATASET):

- SORTIN – в данном случае AREA.CODES;
- SORTOUT – в данном случае *, что означает экран терминала;
- SYSOUT;

- SYSPRINT;
- SYSIN.

```
 READY
ALLOCATE DATASET(AREA.CODES) FILE(SORTIN)    SHR
 READY
ALLOCATE DATASET(*)          FILE(SORTOUT)   SHR
 READY
ALLOCATE DATASET(*)          FILE(SYSOUT)    SHR
 READY
ALLOCATE DATASET(*)          FILE(SYSPRINT)  SHR
 READY
ALLOCATE DATASET(SORT.CNTL)  FILE(SYSIN)     SHR
 READY
CALL 'SYS1.SICELINK(SORT)'

 ICE143I 0 BLOCKSET     SORT  TECHNIQUE SELECTED
 ICE000I 1 - CONTROL STATEMENTS FOR Z/OS DFSORT V1R5
             SORT FIELDS=(1,3,CH,A)
 201  NJ
 202  DC
 203  CT
 204  Manitoba
 205  AL
 206  WA
 207  ME
 208  ID
 ***
```

Рис. 4.4. Использование собственных команд TSO для сортировки данных

После назначения входных и выходных файлов и введенной пользователем команды CALL программа сортировки выводит результаты на экран пользователя. Как показано на рис. 4.4, управляющий оператор SORT FIELDS осуществляет сортировку результатов по телефонному коду. Например, NJ (New Jersey) имеет наименьший телефонный код – 201.

Управление экраном в собственном режиме TSO выполняется очень просто. Например, когда экран заполняется данными, выводятся три звездочки (***), обозначающие полный экран. Это означает, что нужно нажать клавишу Enter, чтобы очистить экран и вывести остальные данные.

4.2.3 Использование CLIST и REXX в TSO

В собственном режиме TSO можно поместить список команд, называемый *командным списком* (*command list*, CLIST; произносится «си-лист»), в файл и выполнять список как одну команду. При вызове CLIST происходит последовательный вызов команд TSO/E. Командные списки используются для выполнения стандартных задач; они позволяют повысить эффективность работы пользователей в TSO.

CLIST – список команд, выполняемых как одна команда

Например, предположим, что команды, представленные в примере 4.4, были объединены

в файле под названием AREA.COMMND. Пользователь мог бы достичь тех же результатов, используя одну команду для выполнения CLIST:

```
EXEC 'CLIST AREA.COMMND'
```

REXX – интерпретируемый командный язык, используемый в TSO

Для создания командных списков пользователи TSO применяют командный язык CLIST. Другой командный язык, используемый в TSO, называется REXX (Restructured Extended Executor). Как CLIST, так и REXX осуществляют обработку, подобную использованию скриптов оболочки. Эти языки являются *интерпретируемыми*, а не *компилируемыми* (хотя в REXX допускается компиляция). Более подробно CLIST и REXX рассматриваются в главе 9 «Использование языков программирования в z/OS».

Некоторые пользователи TSO создают функции непосредственно как CLIST- или REXX-программы, но они чаще реализуются как ISPF-функции или различными программными продуктами. Программирование в CLIST уникально для z/OS, тогда как язык REXX используется на многих платформах.

4.3 Обзор ISPF

После подключения к TSO пользователи обычно осуществляют доступ к меню ISPF. В действительности многие используют исключительно ISPF для выполнения работы в z/OS. ISPF представляет собой полнофункциональное панельное приложение, управляемое с клавиатуры. ISPF включает текстовый редактор и просмотрщик, а также функции поиска и работы со списками файлов и выполнения других служебных функций. Меню ISPF содержат функции, чаще всего применяемые оперативными пользователями.

На рис. 4.5 представлена процедура распределения для создания набора данных с использованием ISPF.

На рис. 4.6 представлены результаты распределения набора данных с использованием панелей ISPF.

На рис. 4.7 представлена структура меню ISPF.

Для доступа к ISPF в TSO пользователю следует ввести команду наподобие ISPPDF в приглашении READY для вывода главного меню ISPF (ISPF Primary Option Menu).

На рис. 4.8 представлен пример главного меню ISPF.

Локальный системный программист может настраивать панель ISPF, используя дополнительные опции. Таким образом, она может иметь различные возможности и содержимое в разных организациях.

Для входа в меню ISPF, представленное на рис. 4.9, следует нажать M в строке опций.

На рис. 4.9 представлена панель, на которой программа SORT идет под номером 9. Мы выбираем ее как полезный пример панельных приложений ISPF.

На рис. 4.10 представлена панель, отображаемая при выборе опции 9 в ISPF.

В разделе 4.2.2, «Использование команд TSO в собственном режиме», был представлен пример того, как пользователь TSO может выполнить простую операцию сортировки посредством ввода команд TSO в собственном режиме TSO. Здесь та же

```
 Menu  RefList  Utilities  Help
 --------------------------------------------------------------------------------
Allocate New Data Set
 Command ===>
Data Set Name  . . . : ZCHOL.TEST.CNTL
Management class . . .                (Blank for default management class)
 Storage class  . . . .                (Blank for default storage class)
  Volume serial . . . . TEST01         (Blank for system default volume) **
  Device type . . . . .                (Generic unit or device address) **
 Data class . . . . . .                (Blank for default data class)
  Space units . . . . . TRACK          (BLKS, TRKS, CYLS, KB, MB, BYTES
                                        or RECORDS)
  Average record unit                  (M, K, or U)
  Primary quantity  . . 2              (In above units)
  Secondary quantity    1              (In above units)
  Directory blocks  . . 0              (Zero for sequential data set) *
  Record format . . . . F
  Record length . . . . 80
  Block size  . . . . .
  Data set name type  :                (LIBRARY, HFS, PDS, or blank)  *
                                       (YY/MM/DD, YYYY/MM/DD
  Expiration date . . .                 YY.DDD, YYYY.DDD in Julian form
 Enter "/" to select option             DDDD for retention period in days
    Allocate Multiple Volumes           or blank)

 ( * Specifying LIBRARY may override zero directory block)

 ( ** Only one of these fields may be specified)
  F1=Help F2=Split F3=Exit F7=Backward F8=Forward F9=Swap F10=Actions  F12=Cancel
```

Рис. 4.5. Распределение набора данных, используя панели ISPF

```
Data Set Information
Command ===>

Data Set Name  . . . : ZCHOL.TEST.CNTL

General Data                          Current Allocation
 Volume serial . . . : TEST01          Allocated tracks  . : 2
 Device type . . . . : 3390            Allocated extents . : 1
 Organization  . . . : PS
 Record format . . . : F
 Record length . . . : 80
 Block size  . . . . : 80             Current Utilization
 1st extent tracks . : 2               Used tracks . . . . : 0
 Secondary tracks  . : 1               Used extents  . . . : 0

 Creation date . . . : 2005/01/31
 Referenced date . . : 2005/01/31
 Expiration date . . : ***None***

 F1=Help F2=Split F3=Exit F7=Backward  F8=Forward   F9=Swap F12=Cancel
```

Рис. 4.6. Результат распределения набора данных с использованием ISPF

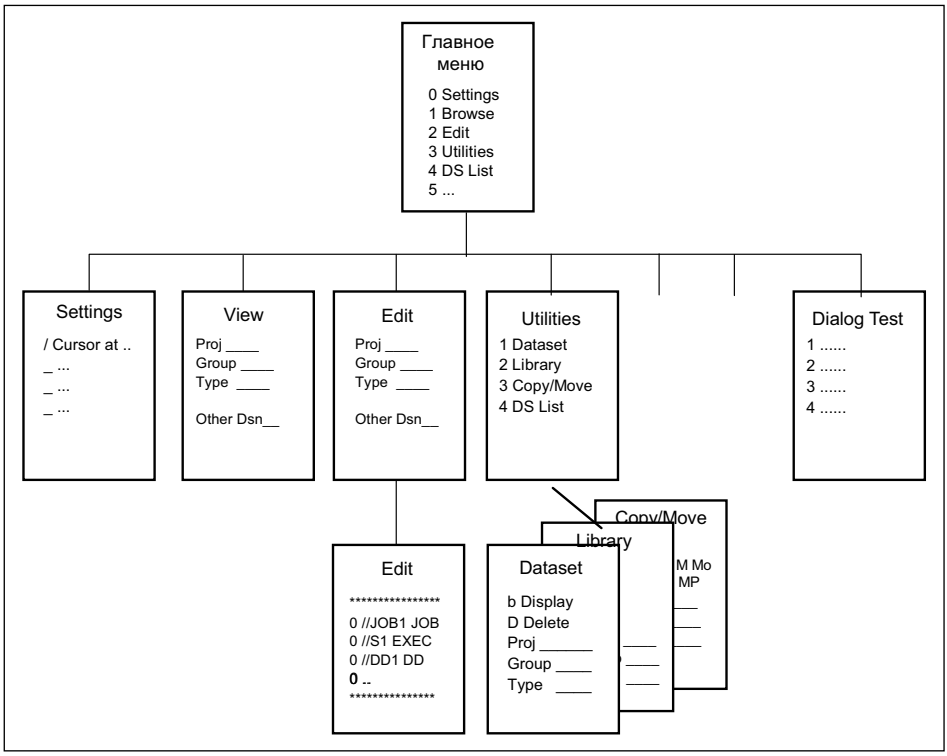


Рис. 4.7. Структура меню ISPF

```
Menu  Utilities  Compilers  Options  Status  Help
--------------------------------------------------------------------------------
                           ISPF Primary Option Menu
Option ===>

0  Settings      Terminal and user parameters           User ID . : ZPROF
1  View          Display source data or listings        Time. . . : 17:29
2  Edit          Create or change source data           Terminal. : 3278
3  Utilities     Perform utility functions              Screen. . : 1
4  Foreground    Interactive language processing        Language. : ENGLISH
5  Batch         Submit job for language processing     Appl ID . : PDF
6  Command       Enter TSO or Workstation commands      TSO logon : IKJACCT
7  Dialog Test   Perform dialog testing                 TSO prefix: ZPROF
8  LM Facility   Library administrator functions        System ID : SC04
9  IBM Products  IBM program development products       MVS acct. : ACCNT#
10 SCLM          SW Configuration Library Manager       Release . : ISPF 5.2
11 Workplace     ISPF Object/Action Workplace
M  More          Additional IBM Products

Enter X to Terminate using log/list defaults

 F1=Help F2=Split F3=Exit F7=Backward F8=Forward F9=Swap F10=Actions  F12=Cancel
```

Рис. 4.8. Главное меню ISPF

```
  Menu  Help
------------------------------------------------------------------------------
                           IBM Products Panel
                                                                More:      +
1  SMP/E        System Modification Program/Extended
2  ISMF         Integrated Storage Management Facility
3  RACF         Resource Access Control Facility
4  HCD          Hardware Configuration Dialogs
5  SDSF         Spool Search and Display Facility
6  IPCS         Interactive Problem Control System
7  DITTO        DITTO/ESA for MVS Version 1
8  RMF          Resource Measurement Facility
9  DFSORT       Data Facility Sort
10 OMVS         MVS OpenEdition
11 DB2          Data Base Products
12 RRS          Resource Recovery Services
13 DB2ADM       Data Base Admin Tool
14 QMF          Query Management Facility
15 MQ           WMQ Series Operations and Control
16 FMN          File Manager 3.1.0perations and Control
17 WLM          Workload Manager
18 PE           Performance Expert




Option ===> 9
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Рис. 4.9. Вывод дополнительных опций ISPF

```
DFSORT PRIMARY OPTION MENU
ENTER SELECTION OR COMMAND ===>

SELECT ONE OF THE FOLLOWING:

  0  DFSORT PROFILE              - Change DFSORT user profile
  1  SORT                        - Perform Sort Application
  2  COPY                        - Perform Copy Application
  3  MERGE                       - Perform Merge Application
  X  EXIT                        - Terminate DFSORT

           \-------------------------------------------------/
           | \-----------------------------------------------/ |
           | |      Licensed Materials - Property of IBM     | |
           | |                                               | |
           | |  5740-SM1 (C) Copyright IBM Corp. 1988, 1992. | |
           | |   All rights reserved.  US Government Users   | |
           | |    Restricted Rights - Use, duplication or    | |
           | |   disclosure restricted by GSA ADP Schedule   | |
           | |           Contract with IBM Corp.             | |
           | /-----------------------------------------------\ |
           /-------------------------------------------------\

USE HELP COMMAND FOR HELP; USE END COMMAND TO EXIT.

 F1=HELP     F2=SPLIT    F3=END      F4=RETURN   F5=RFIND    F6=RCHANGE
 F7=UP       F8=DOWN     F9=SWAP     F10=LEFT    F11=RIGHT   F12=CURSOR
```

Рис. 4.10. Панель программы SORT

функция сортировки доступна посредством ISPF как опция меню. Посредством опции SORT пользователь может разрешить ISPF выполнить распределение наборов в TSO, создать управляющий оператор SORT и вызвать программу SORT для генерирования результатов сортировки.

Обратите внимание на настройки клавиш программных функций (PF) в нижней части каждой панели; клавиша PF3 (END) возвращает пользователя в предыдущую панель.

4.3.1 Назначения клавиш, используемые в этой книге

Многие образы экранов, используемые в этой книге, содержат настройки клавиш программных функций (PF) в ISPF, выводимые в нижней части панели. Как говорилось выше, так как пользователи z/OS часто выполняют назначение клавиш программных функций под свои потребности, назначения клавиш, представленные в этой книге, могут не соответствовать назначениям клавиш, используемым в вашей системе. Действительные настройки клавиш у разных пользователей могут варьироваться.

```
Tutorial ---------------------- Table of Contents -------------------- Tutorial

                  ISPF Program Development Facility Tutorial

The following topics are presented in sequence, or may be selected by entering
a selection code in the option field:
     G  General      - General information about ISPF
     0  Settings     - Specify terminal and user parameters
     1  View         - Display source data or output listings
     2  Edit         - Create or change source data
     3  Utilities    - Perform utility functions
     4  Foreground   - Invoke language processors in foreground
     5  Batch        - Submit job for language processing
     6  Command      - Enter TSO command, CLIST, or REXX exec
     7  Dialog Test  - Perform dialog testing
     9  IBM Products - Use additional IBM program development products
     10 SCLM         - Software Configuration and Library Manager
     11 Workplace    - ISPF Object/Action Workplace
     X  Exit         - Terminate ISPF using log and list defaults
     The following topics will be presented only if selected by number:
     A  Appendices   - Dynamic allocation errors and ISPF listing formats
     I  Index        - Alphabetical index of tutorial topics

F1=Help      F2=Split     F3=Exit      F4=Resize    F5=Exhelp    F6=Keyshelp
  F7=PrvTopic  F8=NxtTopic  F9=Swap     F10=PrvPage  F11=NxtPage  F12=Cancel
```

Рис. 4.11. Главное меню руководства по ISPF

В табл. 4.1 перечисляются некоторые наиболее часто используемые клавиши PF, а также другие функции клавиатуры и соответствующие им клавиши.

Таблица 4.1. Назначения клавиш

Функция	Клавиша
Ввод (Enter)	Ctrl (справа)
Выход, завершение или возврат	PF3
Справка (Help)	PF1
PA1 или «Внимание»	Alt-Ins или Esc
PA2	Alt-Home
Перемещение курсора	Tab или Enter
Очистка	Pause
Предыдущая страница	PF7
Следующая страница	PF8
Перемещение влево	PF10
Перемещение вправо	PF11
Сброс блокировки клавиатуры	Ctrl (слева)

В примерах, приведенных в этой книге, используются эти назначения клавиш. Например, когда говорится нажать Enter, это означает, что следует нажать клавишу Ctrl справа внизу. Если клавиатура заблокируется, нажмите клавишу Ctrl слева внизу.

4.3.2 Использование PF1-HELP и руководство по ISPF

В главном меню ISPF нажмите клавишу PF1 HELP для вывода руководства по ISPF. Новые пользователи ISPF должны ознакомиться с руководством (рис. 4.11) и со справочными средствами ISPF.

Вы, скорее всего, будете использовать лишь часть содержимого руководства по ISPF.

Помимо руководства, можно вызвать электронную справку из любой панели ISPF. При вызове справки можно пролистывать информацию. Используйте клавишу PF1-Help также для вызова описания основных ошибок при наборе в ISPF и примеров правильного набора. ISPF Help также содержит справку по различным функциям, перечисленным в главном меню.

4.3.3 Использование клавиши PA1

Прерываем чтение учебника на короткую рекламу клавиши PA1. Это очень важная клавиша для пользователей TSO, и каждый пользователь должен знать, где она находится.

В старые времена «настоящие» терминалы 3270 имели клавиши с подписями PA1, PA2 и PA3. Эти клавиши назывались клавишами программных действий (Program Action, PA). В действительности только клавиша PA1 все еще широко используется и работает как клавиша прерывания в TSO. В терминологии TSO это называется прерыванием для привлечения внимания (attention interrupt). Другими словами, нажатие клавиши PA1 завершит выполнение текущей задачи.

Поиск клавиши PA1 на клавиатуре эмулятора терминала 3270, такого, как TN3270, может стать проблемой. Эмулятор 3270 может быть настроен на использование различных сочетаний клавиш. В неизмененном сеансе x3270 в качестве клавиши PA1 используется левый Alt-1.

Давайте попробуем воспользоваться клавишей PA1 (она пригодится вам в будущем). Если у вас уже открыт сеанс TSO, попробуйте следующее:

1. Выберите в ISPF опцию 6. Эта панель ввода команд TSO.
2. Введите LISTC LEVEL(SYS1) ALL в командной строке и нажмите Enter. Должен появиться экран выходных данных с тремя звездочками (***) в последней строке. В TSO три звездочки указывают, что есть неотображенные выходные данные и что нужно нажать Enter для их просмотра (это имеет место почти везде в TSO).
3. Нажмите Enter для вывода следующего экрана, затем еще раз Enter и т. д.
4. Нажмите клавишу PA1, используя сочетание клавиш, настроенное в вашем эмуляторе TN3270. Это должно прекратить вывод.

4.3.4 Перемещение по меню ISPF

ISPF включает текстовый редактор и просмотрщик, а также функции поиска и работы со списками наборов данных и выполнения других служебных функций. Мы пока не рассматривали *наборы данных* в этой книге, но вам потребуется по меньшей мере понимание того, как работать с наборами данных, чтобы начать выполнять практические упражнения в этой главе.

Для начала можно рассматривать набор данных как файл, используемый в z/OS для хранения данных или исполняемого кода. Набор данных может иметь имя длиной до 44 символов, например ZSCHOLAR.TEST.DATA. Наборы данных более подробно рассматриваются в главе 5, «Работа с наборами данных».

Имя набора данных обычно сегментируется с использованием одной или нескольких точек для создания отдельных *квалификаторов* набора данных длиной от 1 до 8 символов. Первый квалификатор набора данных представляет собой старший квалификатор (high level qualifier, HLQ). В данном примере HLQ-квалификатором является элемент ZSCHOLAR имени набора данных.

Пользователи z/OS для работы с наборами данных обычно применяют утилиту ISPF Data Set List. Для доступа к этой утилите из главного меню ISPF нужно выбрать **Utilities**, после чего выбрать **Dslist** на панели выбора утилиты (Utility Selection Panel), чтобы отобразить панель, представленную на рис. 4.12.

На этой панели можно использовать поле ввода данных Dsname Level для поиска и вывода списка наборов данных. В частности, для того чтобы найти один набор данных, следует ввести *полное (fully qualified)* имя набора данных. Для поиска диапазонов наборов данных, например всех наборов данных с общим HLQ, нужно ввести в поле Dsname Level только HLQ.

Квалификаторы могут задаваться полностью, частично или подразумеваться по умолчанию. По меньшей мере один квалификатор должен быть частично задан. Для поиска по элементу имени следует ввести звездочку (*) перед или после элемента имени набора данных. В этом случае утилита возвратит все наборы данных, соответствующие критериям поиска. Не следует выполнять поиск по одной лишь звездочке (*), так как в z/OS есть множество мест, в которых TSO будет осуществлять поиск, так что это может занять довольного много времени.

```
Menu  RefList  RefMode  Utilities  Help
--------------------------------------------------------------------------------
                          Data Set List Utility
Option ===> ____________________________________________________________

   blank Display data set list              P Print data set list
       V Display VTOC information           PV Print VTOC information

Enter one or both of the parameters below:
  Dsname Level . . . ZPROF_____________________
   Volume serial  . .  ______
Data set list options
   Initial View . . . 1  1. Volume       Enter "/" to select option
                         2. Space        /  Confirm Data Set Delete
                         3. Attrib       /  Confirm Member Delete
                         4. Total        /  Include Additional Qualifiers

When the data set list is displayed, enter either:
  "/" on the data set list command field for the command prompt pop-up,
  an ISPF line command, the name of a TSO command, CLIST, or REXX exec, or
  "=" to execute the previous command.

 F1=Help F2=Split F3=Exit F7=Backward  F8=Forward   F9=Swap F10=Actions  F12=Cancel
```

Рис. 4.12. Использование утилиты Data Set List

В большинстве панелей ISPF полное имя набора данных следует заключать в одинарные кавычки. Имена наборов данных, не заключенные в одинарные кавычки, по умолчанию предваряются старшим квалификатором, заданным в TSO PROFILE. Это значение можно изменить командой PROFILE PREFIX. Кроме того, исключением является ISPF-опция 3.4 DSLIST; на этой панели не следует заключать значение поля Dsname Level в кавычки.

```
 Menu  Options  View  Utilities  Compilers  Help
 ------------------------------------------------------------------------------
DSLIST - Data Sets Matching ZPROF                          Row 1 of 4
Command ===>                                             Scroll ===> PAGE

Command - Enter "/" to select action                Message          Volume
--------------------------------------------------------------------------------
         ZPROF                                                      *ALIAS
         ZPROF.JCL.CNTL                                             EBBER1
         ZPROF.LIB.SOURCE                                           EBBER1
         ZPROF.PROGRAM.CNTL                                         EBBER1
         ZPROF.PROGRAM.LOAD                                         EBBER1
         ZPROF.PROGRAM.SRC                                          EBBER1
***************************** End of Data Set list ****************************


F1=Help F2=Split F3=Exit F5=Rfind F7=Up F8=Down F9=Swap F10=Left F11=Right F12=Cancel
```

Рис. 4.13. Результирующий список имен наборов данных для значения ZPROF

Например, если в поле Dsname ввести *ZPROF*, утилита выведет все наборы данных, в которых *ZPROF* является старшим квалификатором. Результирующий список имен наборов данных (рис. 4.13) позволяет пользователям редактировать или просматривать содержимое любого набора данных в списке.

```
 Menu  Options  View  Utilities  Compilers  Help
 - +-------------------------------------------------------------+ ----------
 D !                   Data Set List Actions                       ! Row 1 of 4
 C !                                                               !  ===> PAGE
   ! Data Set: ZPROF.PROGRAM.CNTL                                !
 C !                                                               !     Volume
 - ! DSLIST Action                                                 ! -----------
   ! __  1.  Edit                    12. Compress                  !     *ALIAS
 / !     2.  View                    13. Free                      !     EBBER1
   !     3.  Browse                  14. Print Index               !     EBBER1
   !     4.  Member List             15. Reset                     !     EBBER1
 * !     5.  Delete                  16. Move                      ! ***********
   !     6.  Rename                  17. Copy                      !
   !     7.  Info                    18. Refadd                    !
   !     8.  Short Info              19. Exclude                   !
   !     9.  Print                   20. Unexclude 'NX'            !
   !     10. Catalog                 21. Unexclude first 'NXF'     !
   !     11. Uncatalog               22. Unexclude last 'NXL'      !
   !                                                              !
   ! Select a choice and press ENTER to process data set action.  !
   !  F1=Help        F2=Split       F3=Exit        F7=Backward     !
   !  F8=Forward     F9=Swap       F12=Cancel                      !
   +-------------------------------------------------------------+

F1=Help F2=Split F3=Exit F5=Rfind F7=Up F8=Down F9=Swap F10=Left F11=Right F12=Cancel
```

Рис. 4.14. Вывод допустимых действий в списке наборов данных

Для того чтобы просмотреть все возможные действия, которые можно выполнить для заданного набора данных, следует ввести слэш (/) в столбце команды слева от имени набора данных. ISPF выведет список допустимых действий, как показано на рис. 4.14.

4.3.5 Использование редактора ISPF

Для редактирования содержимого набора данных введите `e` (edit) слева от имени набора данных. В наборе данных каждая строка текста называется записью *(record)*.

Можно выполнять следующие задачи:

- для просмотра содержимого набора данных введите строчную команду `v` (view) в столбец;
- для редактирования содержимого набора данных введите строчную команду `e` (edit) в столбец;
- для редактирования содержимого набора данных переместите курсор в область записи, которую требуется изменить, и вводите новый текст поверх существующего;

- для поиска и изменения текста можно вводить команды в командную строку редактора;
- для вставки, копирования, удаления или перемещения текста вводите эти команды непосредственно в номера строк, где следует выполнить действие.

Для фиксации изменений используйте PF3 или команду save. Для выхода из набора данных без сохранения изменений введите `Cancel` в командную строку редактирования.

На рис. 4.15 представлено содержимое набора данных *ZPROF*.PROGRAM.CNTL(SORTCNTL), открытого в режиме редактирования.

```
File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------

EDIT       ZPROF.PROGRAM.CNTL(SORTCNTL) - 01.00       Columns 00001 00072

Command ===>                                          Scroll ===> CSR

****** ***************************** Top of Data ******************************

000010 SORT FIELDS=(1,3,CH,A)

****** **************************** Bottom of Data ****************************
```

Рис. 4.15. Редактирование набора данных

Обратите внимание на номера строк, текстовую область и командную строку редактора. Командная строка, строчные команды, вводимые в номера строк, и ввод поверх существующего текста представляют собой три различных способа изменения содержимого набора данных. Нумерация строк в редакторе TSO выполняется с приращением 10, что позволяет программисту вставить девять дополнительных строк без изменения нумерации в программе.

4.3.6 Использование электронной справки

При редактировании наборов данных вашим личным наставником является F1=Help. PF1 в режиме редактирования выводит полное руководство по редактору (рис. 4.16).

Во время практического упражнения вам потребуется редактировать набор данных и использовать F1=Help для изучения строчных команд редактирования (Edit Line Commands) и основных команд редактора (Edit Primary Commands). В функции Help выберите и просмотрите команды FIND, CHANGE и EXCLUDE. Это упражнение является важным для дальнейшего развития навыков в этом курсе.

Подмножество строчных команд включает:

i	Вставка строки
Клавиша Enter	Нажмите Enter, ничего не вводя, чтобы выйти из режима вставки
i5	Добавление пяти строк ввода
d	Удаление строки
d5	Удаление пяти строк

dd/dd	Удаление блока строк
r	Повторение строки
rr/rr	Повторение блока строк
c, затем a или b	Копирование строки до или после текущей строки
c5, затем a или b	Копирование пяти строк до или после текущей строки
cc/cc, затем a или b	Копирование блока строк до или после текущей строки
m, m5, mm/mm	Перемещение строк
x	Исключение строки

```
TUTORIAL ------------------------- EDIT ---------------------------- TUTORIAL
OPTION  ===>

                    ----------------------------------
                    |             EDIT               |
                    ----------------------------------

  Edit allows you to create or change source data.

The following topics are presented in sequence, or may be selected by number:
  0 - General introduction              8 - Display modes (CAPS/HEX/NULLS)
  1 - Types of data sets                9 - Tabbing (hardware/software/logical)
  2 - Edit entry panel                 10 - Automatic recovery
  3 - SCLM edit entry panel            11 - Edit profiles
  4 - Member selection list            12 - Edit line commands
  5 - Display screen format            13 - Edit primary commands
  6 - Scrolling data                   14 - Labels and line ranges
  7 - Sequence numbering               15 - Ending an edit session

The following topics will be presented only if selected by number:
 16 - Edit models
 17 - Miscellaneous notes about edit


 F1=Help      F2=Split     F3=Exit      F4=Resize    F5=Exhelp    F6=Keyshelp
 F7=PrvTopic  F8=NxtTopic  F9=Swap     F10=PrvPage  F11=NxtPage  F12=Cancel
```

Рис. 4.16. Справочная панель и руководство по редактору

4.3.7 Настройка параметров ISPF

Командная строка сеанса ISPF может выводиться в нижней части экрана, тогда как командная строка ISPF у вашего преподавателя может выводиться вверху. Это является персональным предпочтением, но при традиционном использовании она находится в верхней части панели.

Если требуется, чтобы ваша командная строка выводилась вверху панели, сделайте следующее:

1. Перейдите в главное меню ISPF.
2. Выберите опцию 0 для вывода меню Settings (Настройки), представленного на рис. 4.17.
3. В списке опций (Options) удалите «/» в строке «Command line at bottom» («Командная строка внизу»). Для перемещения курсора следует использовать клавишу табуляции или новой строки.

```
 Log/List  Function keys  Colors  Environ  Workstation  Identifier  Help
-------------------------------------------------------------------------------
                              ISPF Settings
 Command ===>

 Options                                  Print Graphics
   Enter "/" to select option               Family printer type 2
   _  Command line at bottom                Device name . . . .
   /  Panel display CUA mode                Aspect ratio  . . . 0
   /  Long message in pop-up
   _  Tab to action bar choices
   _  Tab to point-and-shoot fields       General
   /  Restore TEST/TRACE options            Input field pad . . B
   _  Session Manager mode                  Command delimiter . ;
   /  Jump from leader dots
   _  Edit PRINTDS Command
   /  Always show split line
   _  Enable EURO sign

 Terminal Characteristics
   Screen format   2  1. Data    2. Std     3. Max     4. Part

   Terminal Type   3    1. 3277       2. 3277A      3. 3278       4. 3278A
                        5. 3290A      6. 3278T      7. 3278CF     8. 3277KN
                        9. 3278KN    10. 3278AR    11. 3278CY    12. 3278HN
                       13. 3278HO    14. 3278IS    15. 3278L2    16. BE163
                       17. BE190     18. 3278TH    19. 3278CU    20. DEU78
                       21. DEU78A    22. DEU90A    23. SW116     24. SW131
                       25. SW500
```

Рис. 4.17. Настройки ISPF

Через это меню можно изменять некоторые параметры, которые вам потребуются в дальнейшем:

- Удалите «/» в строке **Panel display CUA mode**.
- Измените значение в строке **Terminal Type** на 4. Это обеспечивает поддержку символов, используемых в языке C, на терминалах 3270.
- Переместите курсор на опцию **Log/List** в верхней строке и нажмите Enter:
  - выберите **1** (Log Data set defaults);
  - введите опцию **2** (чтобы удалить набор данных без вывода на печать);
  - нажмите PF3 для выхода.

- Переместите курсор еще раз на опцию **Log/List**:
  - выберите **2** (List Data set defaults);
  - введите опцию **2** для удаления набора данных без вывода на печать;
  - нажмите PF3 для выхода.
- Нажмите еще раз PF3 для выхода в главное меню.

Набор действий в верхней строке обычно отличается в разных инсталляциях.

Другой способ настройки панелей ISPF заключается в использовании команды **`hilite`**, как показано на рис. 4.18. Эта команда позволяет настраивать различные опции ISPF для соответствия требованиям вашей среды.

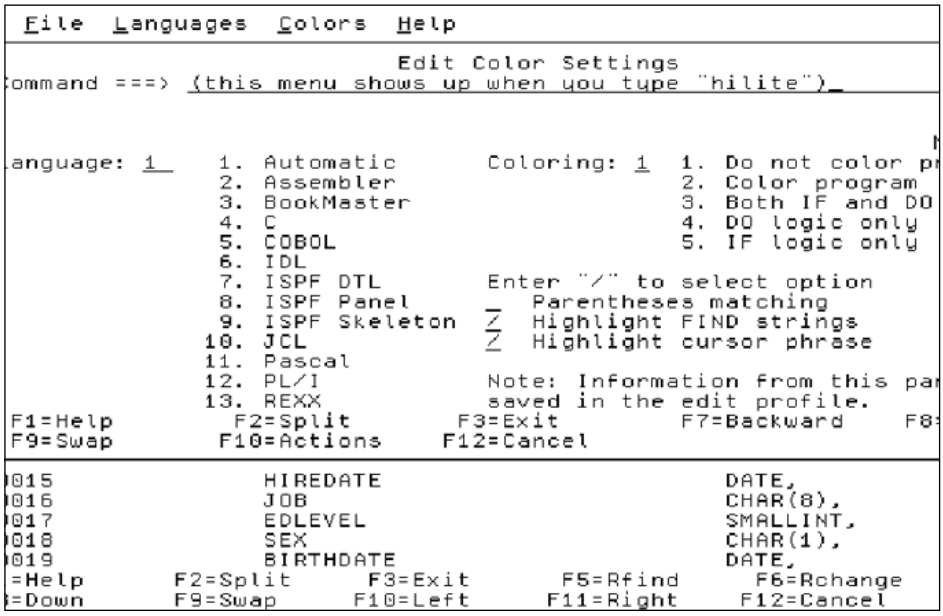


Рис. 4.18. Использование команды HILITE

4.3.8 Добавление графического интерфейса в ISPF

ISPF является полнофункциональным панельным приложением, управляемым с клавиатуры. Тем не менее можно загружать и устанавливать в системе z/OS различные клиенты графического интерфейса ISPF. После установки клиента графического интерфейса ISPF можно пользоваться мышью.

На рис. 4.19 представлен пример графического интерфейса ISPF.

Для использования выпадающих меню в верхней части панелей ISPF требуется навести курсор на выбранное поле и нажать Enter. Проведите указатель мыши графического клиента ISPF по выпадающим пунктам для отображения соответствующих подпунктов. В графическом интерфейсе также доступны кнопки, соответствующие клавишам Enter и PF.

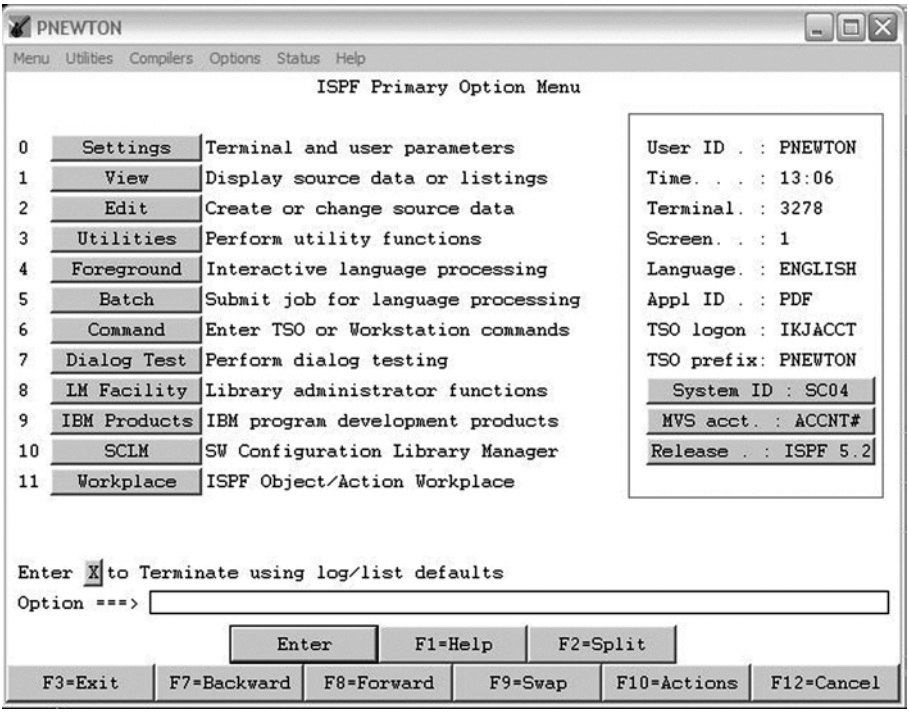


Рис. 4.19. Графический интерфейс ISPF

4.4 Интерактивные интерфейсы z/OS UNIX

Оболочка и утилиты z/OS UNIX обеспечивают интерактивный интерфейс z/OS. Оболочку и утилиты можно сравнить с функциями TSO в z/OS.

Оболочка – интерпретатор команд UNIX и операторов языка оболочки

Для выполнения некоторых командных запросов оболочка вызывает другие программы, называемые *утилитами (utilities)*. Оболочку можно использовать для следующих целей:

- вызова скриптов оболочки и утилит;
- создания скриптов оболочки (именованного списка команд оболочки с использованием языка программирования оболочки).
- запуска скриптов оболочки и программ на языке C в интерактивном режиме, в фоновом режиме TSO или в пакетном режиме.

Пользователь может вызвать оболочку z/OS UNIX следующими способами:

- с дисплея 3270 или с рабочей станции с запущенным эмулятором 3270;
- с терминала, подключенного к TCP/IP-сети, используя команды **`rlogin`** и **`telnet`**;
- через TSO-сеанс, используя команду OMVS.

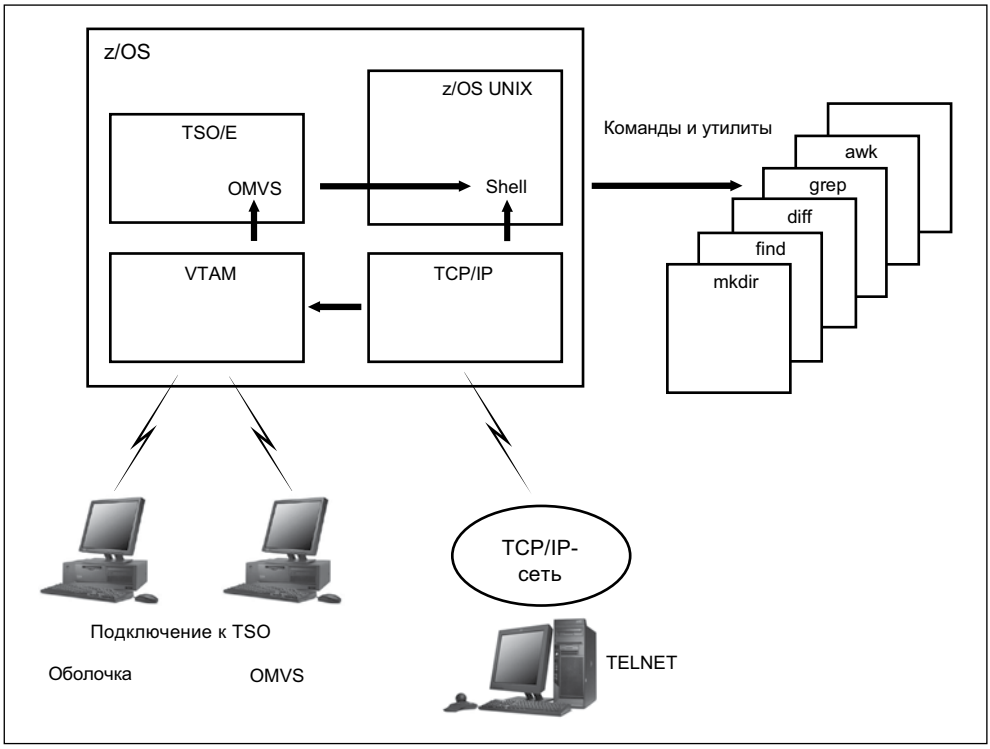


Рис. 4.20. Оболочка и утилиты

Вместо того чтобы вызывать оболочку непосредственно, пользователь может применить ISHELL, введя команду ISHELL из TSO. ISHELL обеспечивает панельный интерфейс ISPF для выполнения множества действий с z/OS UNIX.

На рис. 4.21 представлен обзор этих интерактивных интерфейсов: оболочки z/OS UNIX и ISHELL. Кроме того, существуют команды TSO/E, поддерживающие z/OS UNIX, но они ограничены такими функциями, как копирование файлов и создание директорий.

ISHELL – команда TSO, вызывающая панельный интерфейс ISPF для выполнения множества действий с z/OS UNIX

Оболочка z/OS UNIX основана на оболочке UNIX System V и имеет некоторые возможности оболочки Korn в UNIX. Стандарт POSIX проводит различие между *командой (command)*, представляющей собой указание оболочке выполнить определенную задачу, и *утилитой (utility)*, представляющей имя программы, вызываемой из оболочки. С точки зрения пользователя нет различий между командой и утилитой.

Оболочка z/OS UNIX обеспечивает среду с большим количеством функций и возможностей. Она поддерживает множество возможностей обычного языка программирования.

Последовательность команд оболочки можно хранить в исполняемом текстовом файле. Такой файл называется *скриптом оболочки (shell script)*.

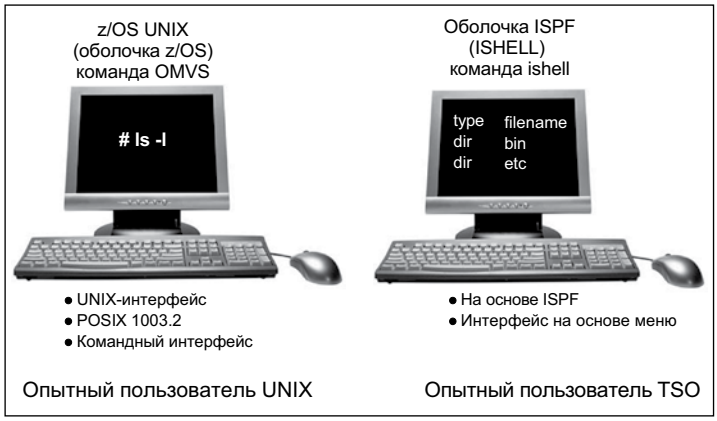


Рис. 4.21. Интерактивные интерфейсы z/OS UNIX

Команды TSO, применяемые в z/OS UNIX:

ISHELL Команда ISHELL вызывает панельный интерфейс ISPF для z/OS UNIX System Services. ISHELL является хорошей отправной точкой для пользователей, знакомых с TSO и ISPF, желающих применять z/OS UNIX. Эти пользователи могут выполнять значительную часть своей работы с применением команды ISHELL, которая предоставляет панели для работы с файловой системой z/OS UNIX, включая панели для подключения и отключения файловых систем и для выполнения некоторых задач администрирования z/OS UNIX. ISHELL часто бывает полезна для системных программистов, знакомых с z/OS, которым требуется выполнить настройку ресурсов UNIX для пользователей.

OMVS Команда OMVS употребляется для вызова оболочки z/OS UNIX. Пользователям, в основном употребляющим в качестве интерактивной среды систему UNIX, оболочка z/OS UNIX будет знакома.

4.4.1 Команда ISHELL (ish)

На рис. 4.22 представлена панель ISHELL (ISPF Shell), отображаемая в результате ввода команды ISHELL или ISH из опции 6 в ISPF.

```
  File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
------------------------------------------------------------------------------
                      UNIX System Services ISPF Shell

Enter a pathname and do one of these:

    - Press Enter.
    - Select an action bar choice.
    - Specify an action code or command on the command line.

Return to this panel to work with a different pathname.
                                                                More:     +
   /u/rogers_____________________________________________________________
   ______________________________________________________________________
   ______________________________________________________________________
   ______________________________________________________________________
```

Рис. 4.22. Панель, отображаемая после ввода команды ISH

4.4.2 ISHELL – пользовательские файлы и директории

Для поиска файлов и директорий пользователя введите следующее:

```
/u/userid
```

и нажмите Enter:

На рис. 4.23 показаны файлы и директории пользователя `rogers`.

```
                          Directory List

Select one or more files with / or action codes.  If / is used also select an
action from the action bar otherwise your default action will be used.  Select
with S to use your default action.  Cursor select can also be used for quick
navigation.  See help for details.
EUID=0   /u/rogers/
  Type  Perm  Changed-EST5EDT   Owner      ------Size  Filename    Row 1 of 9
_ Dir    700  2002-08-01 10:51  ADMIN            8192  .
_ Dir    555  2003-02-13 11:14  AAAAAAA             0  ..
_ File   755  1996-02-29 18:02  ADMIN             979  .profile
_ File   600  1996-03-01 10:29  ADMIN              29  .sh_history
_ Dir    755  2001-06-25 17:43  AAAAAAA          8192  data
_ File   644  2001-06-26 11:27  AAAAAAA         47848  inventory.export
_ File   700  2002-08-01 10:51  AAAAAAA            16  myfile
_ File   644  2001-06-22 17:53  AAAAAAA         43387  print.export
_ File   644  2001-02-22 18:03  AAAAAAA         84543  Sc.pdf
```

Рис. 4.23. Вывод файлов и директорий пользователя

Здесь надо использовать коды действий для выполнения следующих функций:

b Просмотр файла или директории.
e Редактирование файла или директории.
d Удаление файла или директории.
r Переименование файла или директории.
a Вывод атрибутов файла или директории.
c Копирование файла или директории.

4.4.3 Сеанс оболочки команды OMVS

Команда OMVS используется для вызова оболочки z/OS UNIX.

Оболочка представляет собой командный процессор, используемый для выполнения следующих функций:

- вызова команд или утилит оболочки, запрашивающих обслуживание в системе.
- создания скриптов оболочки с использованием языка программирования оболочки.
- запуска скриптов оболочки и программ на языке C в интерактивном режиме, в фоновом режиме TSO или в пакетном режиме.

Команды оболочки часто имеют опции (называемые также *флагами*), которые можно указать и которые обычно принимают аргумент (например, имя файла или директории). Формат ввода команды начинается с имени команды, после которого идет опция или опции, и затем идет аргумент (если есть).

Например, на рис. 4.24 представлена следующая команда:

```
ls -al /u/rogers
```

где `ls` – имя команды, а `-al` – опции.

```
ROGERS @ SC43:/>ls -al /u/rogers
total 408
drwx------   3 ADMIN    SYS1        8192 Aug  1  2005 .
dr-xr-xr-x  93 AAAAAAA  TTY            0 Feb 13 11:14 ..
-rwxr-xr-x   1 ADMIN    SYS1         979 Feb 29  1996 .profile
-rw-------   1 ADMIN    SYS1          29 Mar  1  1996 .sh_history
-rw-r--r--   1 AAAAAAA  SYS1       84543 Feb 22  2001 Sc.pdf
drwxr-xr-x   2 AAAAAAA  SYS1        8192 Jun 25  2001 data
-rw-r--r--   1 AAAAAAA  SYS1       47848 Jun 26  2001 inventory.export
-rwx------   1 AAAAAAA  SYS1          16 Aug  1  2005 myfile
-rw-r--r--   1 AAAAAAA  SYS1       43387 Jun 22  2001 print.export
```

Рис. 4.24. Экран сеанса оболочки OMVS после ввода команды OMVS

Путь/имя пути – маршрут к определенному файлу в файловой системе

Эта команда выводит файлы и директории пользователя. Если имя пути указывает на файл, команда `ls` выводит информацию по этому файлу в соответствии с заданными опциями. Если оно указывает на директорию, `ls` выводит информацию по файлам и поддиректориям. Для получения информации по самой директории, следует использовать опцию `-d`.

Если опции не указаны, `ls` выводит только имена файлов. Когда `ls` направляет выходные данные в программный канал или в файл, он выводит по одному имени в строке; когда вывод направляется на терминал, используется формат `-C` (многоколоночный формат).

Терминология. Пользователи z/OS склонны применять термины *набор данных* и *файл* как синонимы, однако не в тех случаях, когда речь идет о z/OS UNIX System Services. В контексте поддержки UNIX в z/OS файловая система представляет собой набор данных, содержащий директории и файлы. Поэтому файл имеет специальное определение. Файлы z/OS UNIX отличны от других наборов данных z/OS, так как они являются байт-ориентированными, а не ориентированными на записи.

4.4.4 Прямое подключение к оболочке

Можно напрямую подключиться к оболочке z/OS UNIX из системы, подключенной к z/OS через TCP/IP. Для этого следует использовать один из следующих методов:

rlogin Можно выполнить удаленное подключение (remote log in, rlogin) к оболочке из системы, осуществляющей поддержку rlogin-клиентов. Для подключения следует использовать командный синтаксис rlogin, поддерживаемый в вашей инсталляции.

telnet Можно выполнить telnet-подключение к оболочке. Для подключения следует ввести команду telnet со своей рабочей станции или с другой системы с поддержкой telnet-клиента.

Как показано на рис. 4.25, каждый из этих методов требует, чтобы в системе z/OS был установлен и активирован демон inetd.

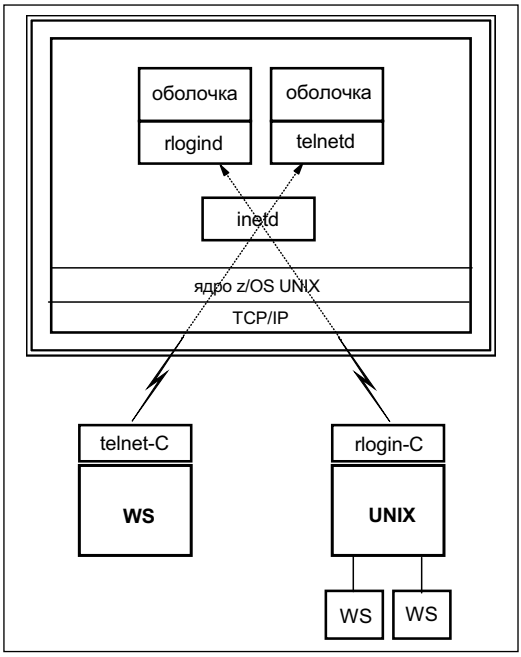


Рис. 4.25. Диаграмма подключения к оболочке с терминала

На рис. 4.26 представлена оболочка z/OS после подключения через telnet.

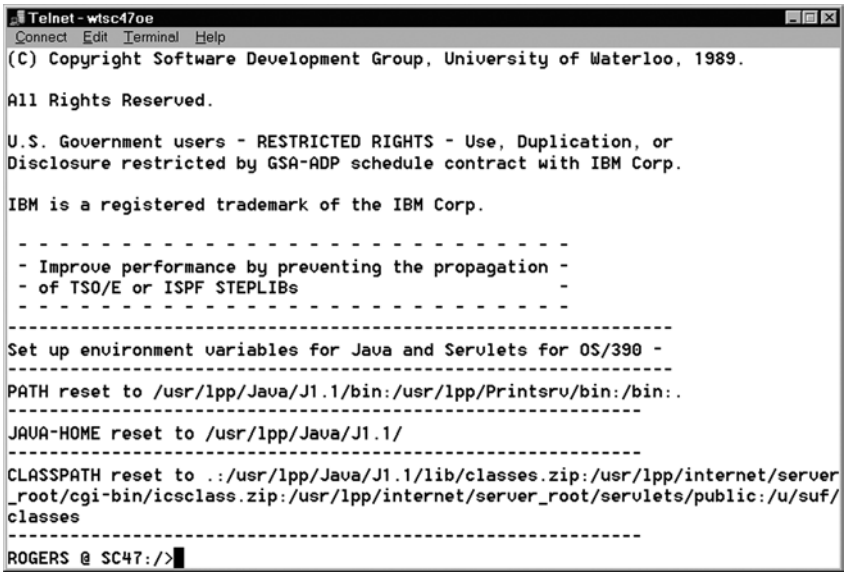


Рис. 4.26. Экран telnet-подключения к оболочке

Существует несколько различий между поддержкой асинхронного терминала (прямого подключения к оболочке) и поддержкой терминала 3270 (команда OMVS):

- Нельзя переключиться в TSO/E. Однако можно использовать команду TSO для выполнения команды TSO/E из своего сеанса оболочки.
- Нельзя использовать редактор ISPF (это относится и к команде **`oedit`**, вызывающей редактор ISPF).
- Можно использовать UNIX-редактор vi и другие интерактивные утилиты, обрабатывающие каждое нажатие клавиш без нажатия Enter.
- Можно использовать редактирование командной строки в стиле UNIX.

4.5 Заключение

TSO дает возможность пользователям подключаться к z/OS и применять ограниченный набор базовых команд. Это иногда называется использованием TSO в собственном режиме.

ISPF представляет собой интерфейс на основе меню для взаимодействия пользователя с системой z/OS. Среда ISPF выполняется из собственного режима TSO.

ISPF предоставляет пользователю доступ к утилитам, редактору и ISPF-приложениям. В пределах, разрешенных различными средствами управления безопасностью, ISPF-пользователь имеет полный доступ к большинству системных функций z/OS.

TSO/ISPF следует рассматривать как интерфейс управления системой и как интерфейс разработки для традиционного программирования z/OS.

Оболочка и утилиты z/OS UNIX представляют собой командный интерфейс для среды z/OS UNIX. Доступ к оболочке можно осуществлять либо посредством подключения к TSO/E, либо с использованием средств удаленного подключения через TCP/IP (rlogin).

При использовании TSO/E, оболочка запускается командой OMVS. Работа в среде оболочки продолжается до выхода или временного переключения в среду TSO/E.

Основные термины в этой главе		
Эмуляция 3270	CLIST	ISHELL
ISPF	Подключение	Собственный режим
Команда OMVS	Путь/имя пути	Запись
REXX (Restructured Extended Executor)	оболочка	TSO/E (Time Sharing Option/ Extensions)

4.6 Контрольные вопросы

Чтобы проверить понимание материала этой главы, ответьте на следующие вопросы:

1. Если требуется дополнительная информация об определенной панели ISPF или справка по пользовательской ошибке, что следует сделать в первую очередь?
2. Чем полезна ISPF-команда PFSHOW OFF?
3. ISPF представляет собой полноэкранный интерфейс с полноэкранным редактором; TSO представляет собой интерфейс командной строки, содержащий только

построчный редактор. Построчный редактор TSO используется редко. Можете ли вы представить ситуацию, в которой потребовалось бы использовать построчный редактор TSO?

4. Можно ли настроить панели ISPF, поставляемые IBM?
5. Назовите два интерактивных интерфейса z/OS UNIX и объясните некоторые различия между ними.

4.7 Упражнения

Практические упражнения в этой главе помогут вам развить навыки использования TSO, ISPF и командной оболочки z/OS UNIX. Эти навыки необходимы для выполнения практических упражнений в остальной части книги. Для выполнения практических упражнений каждому студенту или группе нужен идентификатор и пароль пользователя TSO (для этого следует обратиться к преподавателю).

Упражнения обучают следующим навыкам:

- подключению к z/OS и вводу команд TSO;
- переходу по опциям меню ISPF;
- использованию редактора ISPF;
- использованию SDSF;
- открытию оболочки z/OS UNIX и ввод команд;
- использованию команд OEDIT и OBROWSE.

Наиболее часто используемые функции и соответствующие им клавиши представлены в табл. 4.1.

4.7.1 Подключение к z/OS и ввод команд TSO

Установите 3270-подключение к z/OS, используя эмулятор 3270 на рабочей станции, и подключитесь под своим идентификатором пользователя (мы будем использовать идентификатор *yourid*). Из приглашения TSO READY (после ввода **=x** для выхода из ISPF в собственный режим TSO) введите следующие команды:

1. PROFILE.
   Обратите внимание на значение префикса. Запишите его: это ваш идентификатор пользователя в системе.
2. PROFILE NOPREFIX.
   Эта команда изменяет ваш профиль таким образом, чтобы TSO не выводил префикс в начале ваших команд. Ввод PROFILE PREFIX (со значением) или NOPREFIX (без значения) сообщает системе о том, следует ли использовать значение (например, ваш идентификатор пользователя) для поиска файлов в системе. NOPREFIX сообщает системе о том, что не следует ограничивать результаты только файлами, начинающимися с вашего идентификатора пользователя (например), что она в противном случае делала бы по умолчанию.
3. LISTC.
   Команда LISTCAT (или LISTC для краткости) выводит наборы данных в определенном каталоге (catalog); каталоги будут обсуждаться в следующей главе. Ваш эмуля-

тор 3270 имеет клавишу PA1 (attention). Можно использовать клавишу PA1 для прекращения вывода команд.

> **Примечание.** Вывод трех звездочек (***) указывает на то, что ваш экран заполнен. Нажмите Enter или PA для продолжения.

4. PROFILE PREFIX(userid).
   Эта команда указывает, что ваш идентификатор пользователя будет подставляться в качестве префикса во все неполные имена наборов данных. Это позволяет профильтровать результаты следующей команды (см. п. 5).
5. LISTC.
   Обратите внимание на результат выполнения команды.
6. ISPF (или ISPPDF).

   Вход в ISPF-интерфейс TSO.

> **Примечание**. В некоторых системах требуется также выбрать опцию P для доступа к основному экрану ISPF.

4.7.2 Переход по опциям меню ISPF

Из главного меню ISPF выполните следующие действия:

1. Выберите **Utilities**, после чего выберите **Dslist** на панели выбора утилиты (Utility Selection Panel).
2. Введите `SYS1` в поле Dsname Level, после чего нажмите Enter. Что появилось на экране?
3. Нажмите F8 для перехода к следующей странице или вперед, F7 для перехода к предыдущей странице или назад, F10 для сдвига влево и F11 для сдвига вправо. Выполните выход клавишей F3.
4. Введите `SYS1.PROCLIB` в поле Dsname Level, после чего нажмите Enter. Что появилось на экране?
5. Введите `v` в столбец команды слева от SYS1.PROCLIB. Это библиотечный набор данных со множеством разделов. Введите `s` слева от любого раздела, чтобы выбрать данный раздел для просмотра. Нажмите F1. Какая именно справка появилась?
6. Введите `=0` в командную строку или строку опций ISPF. Какая первая опция выводится на панели Settings в ISPF? Измените настройки таким образом, чтобы перенести командную строку в нижнюю часть панели. Изменения применяются при выходе из панели Settings.
7. Введите `PFSHOW OFF` и затем `PFSHOW ON`. В чем различие? В чем может быть польза?
8. Выйдите в главное меню ISPF. Какое значение используется для выбора Utilities?
9. Выберите **Utilities**.

10. Какое значение на панели выбора утилит (Utilities Selection Panel) используется для выбора Dslist? Выйдите обратно в главное меню ISPF. В строку опций введите значение выбора Utilities, затем введите точку и значение выбора Dslist. Какая панель появилась?
11. Выйдите обратно в главное меню ISPF. Переведите курсор на элемент Status в верхней части панели и нажмите Enter. Выберите значение Calendar и нажмите Enter, после чего выберите значение Session. Что изменилось?
12. Теперь установите первоначальную конфигурацию экрана, используя выпадающее меню Status и выбрав **Session**.

4.7.3 Использование редактора ISPF

Из главного меню ISPF выполните следующие действия:

1. Перейдите в панель утилиты DSLIST и введите *yourid*.JCL в поле Dsname Level. Нажмите Enter.
2. Введите `e` (edit) слева от *yourid*.JCL. Введите `s` (select) слева от раздела EDITTEST. Введите `PROFILE` в командную строку редактирования; обратите внимание на данные, выводимые после строк профиля и сообщений. Просмотрите настройки профиля и сообщения, после чего введите в командную строку `RESET`. Каковы результаты?
3. Введите любую последовательность символов в конец первой строки данных, затем нажмите Enter. В командную строку введите `CAN` (cancel). Нажмите Enter для подтверждения запроса отмены. Повторно запустите редактирование EDITTEST в наборе данных. Были ли сохранены ваши изменения?

> **Замечание.** В процессе ознакомления с ISPF вы запомните буквы и числа, соответствующие некоторым наиболее используемым опциям. Ввод ключа = перед опцией выполняет переход непосредственно к этой опции в обход промежуточных меню.

   Используя знак =, можно также переходить напрямую к вложенным опциям. Например, =3.4 выполняет переход непосредственно в наиболее используемое меню утилит набора данных.
4. Переместите курсор на одну из верхних строк на вашем дисплее. Нажмите F2. Появится вторая панель ISPF. Что произойдет, если после этого несколько раз нажать F9?
5. Используя F9, переключитесь в главное меню ISPF, после чего нажмите F1 для вывода панели ISPF Tutorial.
6. Из панели ISPF Tutorial выберите **Edit**, затем **Edit Line Commands**, затем **Basic Commands**. Нажмите Enter для просмотра руководства по основным командам. Во время просмотра несколько раз переключитесь (F9) в сеанс редактирования и выполните команды в EDITTEST. Повторите эту последовательность для команд перемещения/копирования (Move/Copy) и команд сдвига.
7. Из панели ISPF Tutorial выберите **Edit**, затем **Edit Primary Commands**, затем **FIND/CHANGE/EXCLUDE commands**. Нажмите Enter для просмотра руководс-

тва по командам FIND/CHANGE/EXCLUDE. Во время просмотра несколько раз переключитесь (F9) в сеанс редактирования и выполните команды в EDITTEST.

8. Введите `=X` на панели справки ISPF для завершения сеанса второй панели ISPF. Сохраните и выйдите из панели редактирования (F3) для возврата в главное меню ISPF.

4.7.4 Использование SDSF

В главном меню ISPF найдите и выберите **System Display and Search Facility (SDSF)** – утилиту, позволяющую просматривать выходные наборы данных. Выберите **More**, чтобы вывести SDSF-опцию (5), или просто введите `=M.5`. Главное меню ISPF обычно включает больше пунктов, чем показано на первой панели, а также содержит инструкции по выводу дополнительных пунктов.

1. Введите `LOG`, после чего выполните сдвиг влево (F10), сдвиг вправо (F11), переход на предыдущую страницу (F7) и переход на следующую страницу (F8). Введите в командную строку ввода `TOP` и затем `BOTTOM`. Введите в командную строку ввода `DOWN 500` и `UP 500`. Описание этого системного журнала будет представлено ниже.
2. Обратите внимание на значение SCROLL слева в командной строке ввода.

   ```
   Scroll ===> PAGE
   ```

   Перейдите к значению SCROLL, которое может принимать следующие значения:

C или CSR	Прокрутка до курсора.
P или PAGE	Полная страница или экран.
H или HALF	Половина страницы или половина экрана.

3. Значение SCROLL можно найти на многих панелях ISPF, включая редактор. Это значение можно изменить путем ввода первой буквы режима прокрутки поверх первой буквы текущего значения. Измените значение на CSR, переведите курсор на другую строку в системном журнале и нажмите F7. Оказалась ли строка с курсором вверху экрана?
4. Введите `ST` (status) в командную строку ввода SDSF, затем введите `SET DISPLAY ON`. Просмотрите значения Prefix, Dest, Owner и Sysname. Для вывода всех текущих значений для каждого из них введите * в качестве фильтра, например:

   ```
   PREFIX *
   OWNER *
   DEST
   ```

   Результат должен иметь следующий вид:

   ```
   PREFIX=* DEST=(ALL) OWNER=*
   ```

5. Введите `DA`, чтобы вывести все активные задания. Введите `ST` для вывода состояния всех заданий во входной, активной и выходной очередях. И снова нажмите F7 (предыдущая страница), F8 (следующая страница), F10 (сдвиг влево) и F11 (сдвиг вправо).

4.7.5 Открытие оболочки z/OS UNIX и ввод команд

В главном меню ISPF выберите опцию 6, после чего введите команду OMVS. Из своего домашнего каталога введите следующие команды оболочки:

id	Выводит текущий идентификатор пользователя.
date	Выводит время и дату.
man date	Справка по команде **date**. Прокрутка панелей выполняется нажатием Enter. Для выхода из панелей следует ввести quit.
man man	Справка по пользованию электронной справкой.
env	Переменные окружения для текущего сеанса.
type read	Определяет, является ли read командой, утилитой, синонимом и т. д.
ls	Вывод директории.
ls -l	Вывод текущей директории.
ls -l /etc	Вывод директории /etc.
cal	Вывод календаря текущего месяца.
cal 2005	Вывод календаря 2005 года.
cal 1752	Вывод календаря 1752 года. Пропущено ли 13 дней в сентябре? (Ответ: да в UNIX во всех календарях пропущено 13 дней в сентябре 1752 года). Дополнительное задание: чтобы узнать почему, спросите преподавателя по истории!
exit	Завершение сеанса OMVS.

4.7.6 Использование команд OEDIT и OBROWSE

По-другому запуск оболочки OMVS осуществляется путем ввода команды TSO OMVS на любой панели ISPF. Из своей домашней директории введите следующие команды оболочки:

cd /tmp	Выполняет переход в директорию, в которой вы можете делать изменения.
\	Открывает панель редактирования ISPF и создает новый текстовый файл в текущей директории. Введите немного текста в редактор. Сохраните и выйдите (F3).
ls	Выводит текущую директорию в кратком режиме.
ls -l	Выводит текущую директорию в подробном режиме.
myfile	*myfile* может быть любым файлом, который следует создать.
obrowse myfile	Выводит только что созданный файл.
exit	Завершает сеанс OMVS.

5

Работа с наборами данных

Цель. При работе с операционной системой z/OS необходимо иметь понимание наборов данных – файлов, содержащих программы и данные. Свойства традиционных наборов данных в z/OS значительно отличаются от файловых систем, используемых в системах UNIX и персональных компьютерах. Чтобы еще больше вас заинтересовать, скажем, что в z/OS можно создавать файловые системы UNIX с общими свойствами систем UNIX.

После завершения работы над этой главой вы сможете:

- объяснить, что такое набор данных;
- описать соглашения именования и форматы записи наборов данных;
- перечислить некоторые методы доступа для управления данными и программами;
- описать, для чего нужны каталоги и VTOC;
- создавать, удалять и изменять наборы данных;
- перечислить различия между файловыми системами UNIX и наборами данных z/OS;
- описать использование наборов данных в файловых системах z/OS UNIX.

5.1 Что такое набор данных?

Практически любая работа в системе предполагает ввод или вывод данных. В мэйнфрейм-системе управление использованием устройств ввода-вывода, в частности дисков, накопителей на магнитной ленте и принтеров, осуществляет канальная подсистема, тогда как z/OS связывает данные той или иной задачи с требуемым устройством.

z/OS осуществляет управление данными посредством использования *наборов данных (data sets)*. Термин «набор данных» относится к файлу, содержащему одну или несколько записей. Любая именованная группа записей называется набором данных. Наборы данных могут содержать такую информацию, как медицинские записи или страховые записи, используемые программой, выполняющейся в системе. Наборы данных используются также для хранения информации, нужной приложениям или самой операционной системе, например исходных программ, библиотек макросов или системных переменных или параметров. Что касается наборов данных, содержащих читаемый текст, их можно распечатывать или выводить на консоль (многие наборы данных содержат загрузочные модули или другие непечатаемые двоичные данные). Наборы данных можно *каталогизировать*, что позволяет обращаться к набору данных по имени, не указывая, где он сохранен.

Набор данных – набор логически связанных записей, например библиотека макросов или исходная программа

Попросту говоря, *запись (record)* представляет собой фиксированное количество байтов, содержащих данные. Часто запись содержит связанную информацию, воспринимаемую как единое целое, например как один элемент в базе данных или как данные об одном сотруднике отдела. Термин *поле (field)* относится к определенному фрагменту записи, используемому для хранения определенной категории данных, например имени или отдела сотрудника.

Запись представляет собой базовую единицу информации, используемую программой, запущенной в z/OS[1]. Записи в наборе данных могут быть организованы различным способом, в зависимости от того, каким образом планируется осуществлять доступ к информации. При разработке приложения, осуществляющего, например, обработку данных о сотрудниках, программа может определять формат записи для данных по каждому человеку.

В z/OS существует много разных типов наборов данных и различных методов доступа к ним. В этой главе рассматривается три типа наборов данных: последовательные, секционированные и VSAM.

В *последовательном наборе данных (sequential data set)* записи представляют собой элементы данных, хранящиеся последовательно. Для того чтобы извлечь, например, десятый элемент в наборе данных, система должна сначала передать предыдущие девять элементов. Элементы данных, которые должны использоваться последовательно, например алфавитный список имен в списке класса, лучше всего хранить в последовательном наборе данных.

[1] Файлы z/OS UNIX отличаются от стандартных наборов данных в z/OS, так как они являются байт-ориентированными, а не ориентированными на записи.

Секционированный набор данных (partitioned data set, PDS) состоит из оглавления *(directory)* и *разделов (members)*. Оглавление содержит адрес каждого раздела и, таким образом, позволяет программам или операционной системе осуществлять прямой доступ к каждому разделу. Каждый раздел, однако, состоит из последовательно хранимых записей. Секционированные наборы данных часто называются *библиотеками (libraries)*. Программы хранятся как разделы секционированных наборов данных. Обычно операционная система загружает разделы PDS в память последовательно, однако она может осуществлять прямой доступ к разделам при выборе программы для выполнения.

В наборе данных VSAM KSDS (Virtual Storage Access Method Key Sequenced Data Set) записи представляют собой элементы данных, хранящиеся вместе с управляющей информацией (ключами), вследствие чего система может извлечь элемент без выполнения поиска всех предыдущих элементов в наборе данных. Наборы данных VSAM KSDS идеально подходят для элементов данных, используемых часто и в непредсказуемом порядке. Различные типы наборов данных и использование каталогов обсуждается далее в этой главе.

Дополнительные сведения. Стандартным справочником по информации о наборах данных является публикация *z/OS DFSMS Using Data Sets*. Эту и другие публикации IBM можно найти на веб-сайте z/OS Internet Library

```
http://www.ibm.com/servers/eserver/zseries/zos/bkserv/
```

5.2 Где хранятся наборы данных?

z/OS поддерживает множество различных устройств хранения данных. Наиболее часто используемыми устройствами для долгосрочного хранения наборов данных являются диски и магнитная лента. Дисковые приводы называются *устройствами хранения с прямым доступом (direct access storage devices, DASD)*, так как, несмотря на то что некоторые наборы данных на них могут быть сохранены последовательно, эти устройства могут осуществлять прямой доступ. Приводы для магнитных лент называются устройствами последовательного доступа, так как доступ к наборам данных на ленте должен осуществляться последовательно.

Термин *DASD* применяется к дискам или эмулируемым аналогам дисков. DASD-тома используются для хранения данных и исполняемых программ, включая и саму операционную систему, а также для временного хранения рабочих данных. Один DASD-том можно использовать для множества различных наборов данных, повторно распределяя и используя пространство тома.

Для того чтобы система могла быстро найти требуемый набор данных, z/OS включает набор данных, называемый главным каталогом (master catalog), который позволяет осуществлять доступ к любому набору данных в компьютерной системе или к другим каталогам наборов данных. В z/OS требуется, чтобы главный каталог находился на DASD-томе, установленном в приводе, постоянно включенном в системе. Более подробно каталоги обсуждаются в разделе 5.11, «Каталоги и VTOC».

5.3 Что такое методы доступа?

Метод доступа определяет способ хранения и извлечения данных. Методы доступа предоставляют собственные структуры наборов данных для организации данных, системные программы (или *макросы*) для определения наборов данных и служебные программы для обработки наборов данных.

Методы доступа определяются главным образом организацией наборов данных. Пользователи z/OS, например, применяют базисный последовательный метод доступа (basic sequential access method, BSAM) или последовательный метод доступа с очередями (queued sequential access method, QSAM) для последовательных наборов данных.

Иногда метод доступа, предназначенный для определенной организации наборов данных, можно использовать для обработки набора данных, организованного по-другому. Например, последовательный набор данных (не являющийся набором данных расширенного формата), созданный с использованием BSAM, допускает обработку с использованием базисного прямого метода доступа (basic direct access method, BDAM) и наоборот. Другим примером являются файлы UNIX, которые можно обрабатывать, используя BSAM, QSAM, базисный библиотечный метод доступа (basic partitioned access method, BPAM) или виртуальный метод доступа (virtual storage access method, VSAM).

В этой книге не рассматриваются все методы доступа, существующие в z/OS. К наиболее используемым методам доступа относятся следующие:

QSAM	Последовательный метод доступа с очередями (широко используется).
BSAM	Базисный последовательный метод доступа (используется в особых ситуациях).
BDAM	Базисный прямой метод доступа (становится устаревшим).
BPAM	Базисный библиотечный метод доступа (для библиотек).
VSAM	Виртуальный метод доступа (используется в более сложных приложениях).

5.4 Как используются DASD-тома?

DASD-тома используются для хранения данных и исполняемых программ (включая и саму операционную систему), а также для временного хранения рабочих данных. Один DASD-том можно применять для множества различных наборов данных, повторно распределяя и используя пространство тома.

В пределах тома имя набора данных должно быть уникальным. Набор данных можно найти по типу устройства, серийному номеру тома и имени набора данных. В этом набор данных отличается от файла в системе UNIX. Базовая файловая структура в z/OS является неиерархической. В наборах данных z/OS нет аналогов путям.

Несмотря на то что DASD-тома отличаются по физическому представлению, размеру и скорости работы, они похожи с точки зрения записи данных, проверки данных, формата данных и программирования. Рабочая поверхность каждого тома разделена на множество концентрических *дорожек (tracks)*. Количество дорожек и их размер варьируется для разных устройств. Каждое устройство имеет механизм доступа, содержащий головки чтения-записи для передачи данных при прохождении рабочей поверхности под ними.

5.4.1 Терминология DASD для пользователей UNIX и PC

Работа аппаратного и программного обеспечения мэйнфрейма с дисками и наборами данных значительно отличается от систем UNIX и PC, и при этом используется специальная терминология. В этой книге для описания различных аспектов управления хранением в z/OS применяются следующие термины:

- *DASD (Direct Access Storage Device)* является другим названием дискового привода.
- Дисковый привод также называется *дисковым томом, дисковым пакетом или блоком дисков с головками (Head Disk Assembly, HDA)*. В этой книге используется термин *том*, за исключением случаев обсуждения физических свойств устройств.
- Дисковый привод содержит *цилиндры*.
- Цилиндры содержат *дорожки*.
- Дорожки содержат *записи данных* и имеют формат CKD (Count Key Data)[1].
- *Блоки данных* представляют собой единицы записи на диске.

5.4.2 Что такое DASD-метки?

Операционная система использует группы меток для обозначения DASD-томов и хранящихся на них наборов данных. Приложения клиентов обычно не используют эти метки напрямую. DASD-тома должны использовать стандартные метки. Стандартные метки включают метку тома, метку набора данных для каждого набора данных и дополнительные пользовательские метки. Метка тома, хранящаяся в нулевой дорожке нулевого цилиндра, идентифицирует каждый DASD-том.

Системные программисты z/OS и администраторы внешней памяти применяют служебную программу ICKDSF для инициализации каждого DASD-тома до его использования в системе. ICKDSF генерирует метку тома и создает оглавление тома (volume table of contents, VTOC) – структуру, содержащую метки наборов данных (более подробно VTOC обсуждаются в разделе «Что такое VTOC?»). Системные программисты могут также использовать ICKDSF для сканирования тома, чтобы убедиться в его целостности и переформатировать все дорожки.

5.5 Распределение набора данных

Для того чтобы использовать набор данных, нужно сначала его распределить (установить связь с ним) и затем осуществлять доступ к данным, используя макросы для выбранного метода доступа.

Понятие *распределения* набора данных может иметь одно из двух или оба следующих значения:

- выделение (создание) пространства для нового набора данных на диске;
- установление логической связи между шагом задания и любым набором данных.

[1] Современные устройства на самом деле используют протоколы Extended Count Key Data (ECKD™), но мы будем использовать CKD в качестве собирательного названия в этой книге.

В конце этой главы рассматривается распределение набора данных с использованием опции 3.2 панели ISPF. Другие способы распределения набора данных включают следующие методы:

Службы методов доступа	Распределение наборов данных можно выполнять с использованием многофункциональных служб, называемых службами методов доступа. К службам методов доступа относятся часто используемые команды для работы с наборами данных, такие, как ALLOCATE, ALTER, DELETE и PRINT.
ALLOCATE	Можно использовать команду TSO ALLOCATE для создания наборов данных. Эта команда управляет вводом требуемых параметров распределения.
Меню ISPF	Можно использовать набор меню TSO, называемый Interactive System Productivity Facility. Одно из меню управляет распределением набора данных пользователем.
Использование JCL	Можно использовать набор команд, называемый языком управления заданиями (job control language), для распределения наборов данных.

5.6 Как назначаются имена наборов данных

При распределении нового набора данных необходимо дать этому набору данных уникальное имя.

Имя набора данных может состоять из одного сегмента имени или из набора соединенных сегментов имени. Каждый сегмент имени представляет квалификатор определенного уровня. Например, имя набора данных VERA.LUZ.DATA состоит из трех сегментов. Первый левый сегмент имени называется старшим квалификатором (high-level qualifier, HLQ), последний правый сегмент имени называется младшим квалификатором (lowest-level qualifier, LLQ).

HLQ – первый сегмент многосегментного имени

Сегменты или *квалификаторы (qualifiers)* могут содержать не больше восьми символов, первый из которых должен быть алфавитным (A–Z) или *специальным* (# @ $). В качестве остальных семи символов можно использовать алфавитные, цифровые (0–9), специальные символы или дефис (-). Сегменты имени разделяются точкой (.).

Имя набора данных, включая все сегменты имени и точки, не должно содержать больше 44 символов. Таким образом, имя набора данных может содержать не больше 22 сегментов.

Например, следующие имена наборов данных являются недопустимыми:

- имя с квалификатором длиной больше восьми символов (HLQ.ABCDEFGHI.XYZ);
- имя, содержащее две точки подряд (HLQ..ABC);
- имя, завершающееся точкой (HLQ.ABC.);
- имя, содержащее квалификатор, начинающийся не с алфавитного или специального символа (HLQ.123.XYZ).

HLQ в именах наборов данных пользователей обычно контролируется системой безопасности. Относительно остальной части имени существует ряд соглашений. Это именно *соглашения*, а не правила, однако они довольно широко используются. Имеют место следующие соглашения:

- Буквы LIB в любой части имени указывают на то, что набор данных является библиотекой. Буквы PDS означают то же самое, но используются реже.
- Буквы CNTL, JCL или JOB в любой части имени обычно указывают на то, что набор данных содержит JCL (но может содержать не только JCL).
- Буквы LOAD, LOADLIB или LINKLIB в имени указывают на то, что набор данных содержит исполняемый код (библиотека с исполняемыми модулями z/OS должна содержать только исполняемые модули).
- Буквы PROC, PRC или PROCLIB обозначают библиотеку процедур JCL.
- Используются различные сочетания, обозначающие исходный код на определенном языке, например COBOL, Assembler, FORTRAN, PL/I, JAVA, C или C++.
- Часть имени набора данных может обозначать определенный проект, например PAYROLL.
- Использование чрезмерного числа квалификаторов считается плохим стилем. Например,

  `P390A.A.B.C.D.E.F.G.H.I.J.K.L.M.N.O.P.Q.R.S`

  является допустимым именем набора данных (в верхнем регистре, не превышает 44 байта, отсутствуют специальные символы), но не очень осмысленным. Хорошим стилем считается, когда имя набора данных содержит три-четыре квалификатора.
- Опять же повторимся, что точки учитываются при расчете 44-символьного предела.

5.7 Распределение пространства на DASD-томах посредством JCL

В этом разделе описывается распределение набора данных с использованием языка управления заданиями (job control language, JCL). Использование JCL описывается далее в этой книге; в этом разделе рассматриваются некоторые параметры распределения пространства для наборов данных, которые будут использоваться дальше в этой книге. Помимо JCL, другими распространенными методами распределения наборов данных являются утилита IDCAMS или автоматическое распределение наборов данных с использованием DFSMS.

В JCL можно задать размер требуемого пространства в блоках, записях, дорожках или цилиндрах. При создании набора данных на DASD следует явно указать размер требуемого пространства с использованием параметра SPACE или неявно с использованием информации, доступной в классе данных[1].

При активной SMS система может использовать класс данных, даже если набор данных не является SMS-управляемым. Для наборов данных, управляемых системой,

[1] При распределении набора данных с использованием DFSMS или утилиты IDCAMS, можно задать распределение пространства в килобайтах или мегабайтах, а не в блоках, записях, дорожках или цилиндрах.

система выбирает тома, устраняя необходимость задавать том при распределении набора данных.

Если запрос на распределение пространства задается по средней длине записи, выделение пространства не зависит от типа устройства. Независимость от устройства особенно важна для памяти, управляемой системой.

5.7.1 Логические записи и блоки

Длина логической записи (logical record length, LRECL) является единицей информации о единице обработки (например, о клиенте, счете, сотруднике в платежной ведомости и т. д.). Она представляет собой наименьший объем обрабатываемых данных и состоит из полей, содержащих информацию, распознаваемую обрабатывающим приложением.

Длина логической записи (LRECL) – максимальная длина логической записи – атрибут DCB набора данных

При выделении пространства на DASD, магнитной ленте или оптических носителях логические записи группируются в физические записи, называемые блоками. BLKSIZE указывает длину этих блоков. Каждый блок данных на DASD-томе имеет определенное расположение и уникальный адрес, что позволяет найти любой блок без интенсивного поиска. Логические записи можно сохранять и извлекать либо напрямую, либо последовательно.

Максимальная длина логической записи (LRECL) ограничена физическим объемом используемого носителя.

Если требуемый объем пространства выражается в блоках, необходимо задать количество и среднюю длину блоков в наборе данных.

Приведем пример запроса дискового пространства:

- средняя длина блока в байтах = 300;
- первичное количество (число) блоков = 5000;
- вторичное количество блоков, выделяемых, если первичное количество будет заполнено данными, = 100.

На основании этой информации операционная система оценивает и выделяет требуемое количество дискового пространства.

5.7.2 Экстенты наборов данных

Пространство для набора данных на диске выделяется в *экстентах (extents)*. Экстент представляет собой *непрерывный* набор дорожек, цилиндров или блоков дискового привода. Количество экстентов, занимаемых наборами данных в процессе своего роста, может увеличиваться. Старые типы наборов данных могут иметь до 16 экстентов на том. Более новые типы наборов данных могут иметь до 128 экстентов на том или до 255 экстентов на нескольких томах.

Экстенты важны, когда не используются PDSE и приходится осуществлять управление пространством самостоятельно, а не через DFSMS. В данном случае лучше, чтобы набор данных поместился в один экстент для максимизации быстродействия

дисков. Чтение или запись непрерывного набора дорожек выполняется быстрее, чем чтение или запись дорожек, разбросанных по диску, что могло бы иметь место в случае динамического выделения дорожек. Однако при недостаточности непрерывного пространства набор данных записывается в несколько экстентов.

5.8 Форматы записи наборов данных

Традиционные наборы данных z/OS являются *ориентированными на записи (record oriented)*. При нормальном использовании отсутствуют байт-ориентированные файлы, подобные используемым в PC и UNIX-системах. (z/OS UNIX использует байт-ориентированные файлы; байт-ориентированные функции существуют также в других специализированных областях. Такие наборы данных не считаются традиционными).

В z/OS не используются символы новой строки (new line, NL) или возврата каретки и перевода строки (carriage return + line feed, CR+LF) для обозначения конца записи. Записи любого набора данных имеют либо фиксированную, либо переменную длину. Например, при редактировании набора данных через ISPF каждая строка является записью.

Традиционные наборы данных z/OS имеют один из пяти форматов записи:

F – Fixed (Фиксированный)	Это означает, что один физический блок на диске соответствует одной логической записи и что все блоки/записи имеют одинаковый размер. Этот формат используется редко.
FB - Fixed Blocked (Фиксированный блочный)	Это означает, что несколько логических записей объединяются в один физический блок. Это может обеспечить эффективное использование пространства и работу. Этот формат широко используется для записей фиксированной длины.
V – Variable (Переменный)	В этом формате используется одна логическая запись как один физический блок. Логическая запись переменной длины состоит из дескриптора записи (record descriptor word, RDW), за которым следуют данные. Дескриптор записи представляет собой 4-байтовое поле, описывающее запись. Первые 2 байта содержат длину логической записи (включая 4-байтовый RDW). Длина может иметь значение от 4 до 32 760 байт. Все биты третьего и четвертого байтов должны быть равны 0, так как другие значения используются для сцепленных записей. Этот формат используется редко.
VB – Variable Blocked (Переменный блочный)	В этом формате несколько логических записей переменной длины (каждая из которых содержит RDW) помещаются в один физический блок. Программа должна поместить в начало блока дополнительный дескриптор блока (Block Descriptor Word, BDW), содержащий общую длину блока.

U – Undefined (Неопределенный)	Этот формат содержит физические записи/блоки переменной длины без предопределенной структуры. Хотя этот формат может выглядеть привлекательным для различных необычных вариантов применения, обычно он используется только для исполняемых модулей.

Необходимо обратить внимание на различие между блоком и записью. Блок – это то, что записывается на диск, тогда как запись представляет собой логическую сущность.

Используемая здесь терминология широко применяется в литературе по z/OS. Приведем основные термины.

- Размер блока (Block Size, BLKSIZE) – размер физического блока, записанного на диск для F- и FB-записей. Для V-, VB- и U-записей он указывает максимальный размер физического блока, который можно использовать для набора данных.
- Размер логической записи (Logical Record Size, LRECL) – представляет либо размер логической записи (F, FB) либо максимальный допустимый размер логической записи (V, VB) для набора данных. Записи формата U не используют показатель LRECL.
- Формат записи (Record Format, RECFM) – F, FB, V, VB или U (см. выше).

Размер блока – размер физического блока, записанного на диск для F- и FB-записей

Эти понятия называются свойствами блока управления данными (data control block, DCB), так как в программе на ассемблере они задаются именно в этом управляющем блоке. От пользователя часто требуется задать эти параметры при создании нового набора данных. Тип и длина набора данных определяется его форматом записей (RECFM) и длиной логической записи (LRECL). Наборы данных фиксированной длины имеют формат записей F, FB, FBS и т. д. Наборы данных переменной длины имеют формат записей V, VB, VBS и т. д.

Формат записи (RECFM) – формат записи; одно из свойств блока управления данными

Набор данных с параметрами RECFM=FB и LRECL=25 представляет собой набор данных фиксированной длины (FB) с длиной записи 25 байт (B – «блочный»). В наборе данных формата FB длина логической записи (LRECL) указывает длину каждой записи в наборе данных; все записи имеют одинаковую длину. Первый байт данных в FB-записи находится в позиции 1. Запись в наборе данных формата FB с LRECL=25 может иметь следующий вид:

Позиции 1–3: Код страны = ‘USA’

Позиции 4–5: Код штата = ‘CA’

Позиции 6–25: Город = ‘San Jose’ + 12 пробелов справа

Набор данных с параметрами RECFM=VB и LRECL=25 представляет собой набор данных переменной длины (VB) с максимальной длиной записи 25 байт. В наборе данных формата VB записи могут иметь разную длину. Первые 4 байта каждой записи содержат RDW, и первые 2 байта RDW содержат длину записи (в двоичном представ-

лении). Первый байт данных VB-записи находится в позиции 5, после 4-байтового RDW, расположенного в позициях 1–4. Запись в наборе данных формата VB с LRECL=25 может иметь следующий вид:

Позиции 1–2: Длина в RDW = шестнадцатеричное 0011 = десятичное 17

Позиции 3–4: Нули в RDW = шестнадцатеричное 0000 = десятичное 0

Позиции 5–7: Код страны = 'USA'

Позиции 8–9: Код штата = 'CA'

Позиции 10–17: Город = 'San Jose'

Рис. 5.1 иллюстрирует взаимосвязь между записями и блоками для каждого из пяти форматов записей.

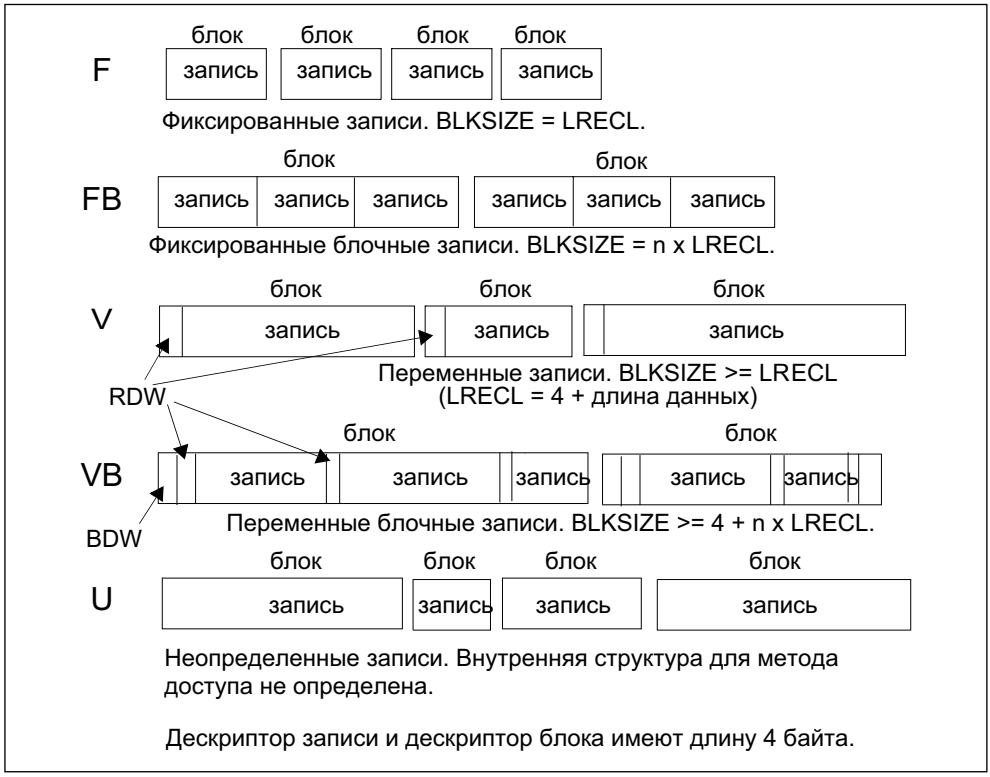


Рис. 5.1. Основные форматы записей

5.9 Типы наборов данных

В z/OS существует множество различных типов наборов данных и различных методов управления ими. В этой главе рассматривается три типа: последовательный, секционированный и VSAM. Все они используются для хранения на дисках; мы также рассмотрим наборы данных на магнитной ленте.

5.9.1 Что такое последовательный набор данных?

Простейшей структурой данных в системе z/OS является последовательный набор данных. Он состоит из одной или нескольких записей, записываемых в физическом порядке и обрабатываемых последовательно. Новые записи добавляются в конец набора данных.

Примерами последовательного набора данных являются выходной набор данных для построчного принтера и файл журнала.

Пользователь z/OS определяет последовательные наборы данных посредством языка управления заданиями (job control language, JCL), указав организацию набора данных PS (DSORG=PS), что означает physical sequential (физически последовательная). Другими словами, записи в наборе данных физически располагаются одна за другой.

В этой главе рассматриваются, главным образом, наборы данных на дисках, однако мэйнфрейм-приложения могут также использовать наборы данных на магнитной ленте для различных целей. Приводы для носителей на магнитной ленте для мэйнфреймов используют записи (блоки) переменной длины. Максимальный размер блока для стандартных методов программирования составляет 65 Кб. Специализированные программы могут создавать блоки большей длины. Существует множество моделей приводов для носителей на магнитной ленте с различными характеристиками.

5.9.2 Что такое PDS?

Секционированный набор данных (partitioned data set, PDS) добавляет уровень организации в простую структуру последовательных наборов данных. PDS представляет собой собрание последовательных наборов данных, называемых *разделами (members)*. Каждый раздел подобен последовательному набору данных и имеет простое имя длиной до восьми символов.

PDS, кроме того, содержит оглавление. Оглавление содержит записи для каждого раздела в PDS с ссылкой (или указателем) на раздел. Имена разделов перечисляются в оглавлении в алфавитном порядке, но сами разделы могут находиться в библиотеке в любом порядке. Оглавление дает возможность системе извлечь определенный раздел в наборе данных.

Раздел – секция секционированного набора данных (partitioned data set, PDS) или расширенного секционированного набора данных (partitioned data set extended, PDSE)

Библиотека – секционированный набор данных, используемый для хранения исходных программ, параметров и исполняемых модулей

Секционированные наборы данных часто называют *библиотеками (library)*. В z/OS библиотеки используются для хранения исходных программ, параметров управления системами и приложениями, JCL и исполняемых модулей. Существует очень мало системных наборов данных, не являющихся библиотеками.

PDS расходует пространство при каждом изменении или добавлении раздела. В результате пользователям z/OS требуется регулярно сжимать PDS для восстановления потерянного пространства.

Пользователь z/OS определяет PDS посредством JCL с организацией наборов данных PO (DSORG=PO), что означает partitioned organization (секционированная организация).

Почему PDS имеет такую структуру?

Структура PDS была разработана таким образом, чтобы обеспечивать эффективный доступ к библиотекам связанных разделов, будь то загрузочные модули, модули исходного кода программ, JCL или многие другие типы содержимого.

Многие системные наборы данных также хранятся в наборах данных PDS, особенно когда они состоят из множества небольших связанных файлов. Например, определения панелей ISPF хранятся в наборах данных PDS.

Основное назначение ISPF состоит в том, чтобы создавать и управлять наборами данных PDS. Эти наборы данных обычно содержат исходный код программ, текст руководств или экранов справки либо JCL для распределения наборов данных и запуска программ.

Преимущества PDS

Набор данных PDS обеспечивает простой и эффективный способ организации связанных групп последовательных файлов. С точки зрения пользователей z/OS PDS имеет следующие преимущества:

- Группирование связанных наборов данных под одним именем упрощает управление данными в z/OS. Можно либо отдельно обрабатывать файлы, хранящиеся как разделы PDS, либо обрабатывать все разделы как единое целое.
- Так как пространство, распределяемое для наборов данных z/OS, всегда выделяется с начала дорожки на диске, использование PDS позволяет сохранять больше одного небольшого набора данных в дорожке. В ситуациях, когда есть много наборов данных гораздо меньшего размера, чем дорожка, это позволяет сэкономить дисковое пространство. На дисковых устройствах 3 390 дорожка составляет 56 664 байта.
- Разделы PDS можно использовать в качестве последовательных наборов данных, которые можно добавлять (или *конкатенировать*) к последовательным наборам данных.
- Несколько наборов данных PDS можно конкатенировать, создавая, таким образом большие библиотеки.
- Наборы данных PDS легко создаются с помощью JCL или ISPF; они просты в управлении с использованием утилит ISPF или команд TSO.

Недостатки PDS

Наборы данных PDS просты, гибки и широко распространены. Однако некоторые аспекты построения PDS неблагоприятно влияют как на производительность, так и на эффективность использования дисковой памяти, в частности имеют место следующие недостатки:

- Потери пространства.
  При замене раздела в PDS новая область данных записывается в новый раздел области памяти, выделенной для PDS. При удалении раздела удаляется и указатель на него, т. е. отсутствует механизм повторного использования пространства. Это потерянное пространство часто называется газом (gas), который требуется периодически удалять путем реорганизации PDS, например с использованием утилиты IEBCOPY для его сжатия.
- Ограниченный размер оглавления.
  Размер оглавления PDS задается во время распределения. С увеличением размера набора данных он может занимать больше пространства в единицах размера, заданного в качестве дополнительного пространства. Эти дополнительные единицы называются *вторичными экстентами (secondary extents)*.
  Однако в оглавлении PDS можно сохранить лишь фиксированное количество записей разделов, так как размер фиксируется при распределении набора данных. Если требуется сохранить больше записей, чем имеется пространства, необходимо распределить новый PDS с дополнительными блоками огравления и скопировать в него разделы из старого набора данных. Это означает, что при распределении PDS необходимо посчитать количество требуемого пространства оглавления.
- Длительный поиск в оглавлении.
  Как говорилось выше, запись в оглавлении PDS состоит из имени и указателя на расположение раздела. Записи хранятся в алфавитном порядке имен разделов. Вставка записи в начале большого оглавления может вызвать значительное количество операций ввода-вывода, так как все записи после новой перемещаются, чтобы освободить место для нее.
  Кроме того, поиск в записях осуществляется последовательно в алфавитном порядке. Если оглавление очень велико и разделы малы, поиск в оглавлении может занять больше времени, чем извлечение раздела, если известно его расположение.

5.9.3 Что такое PDSE?

Аббревиатура PDSE означает «partitioned data set extended» (расширенный секционированный набор данных). Этот набор данных состоит из оглавления и из одного или нескольких разделов, как и PDS. Как и PDS, его можно создать с использованием JCL, TSO/E и ISPF и обрабатывать, используя такие же методы доступа.

PDS / PDSE – библиотека z/OS, содержащая разделы, в частности исходные программы

Оглавление может по мере необходимости автоматически расширяться до предела адресации 522 236 разделов. Он также содержит индекс, обеспечивающий быстрый поиск имен разделов. Пространство удаленных или перемещенных разделов автоматически повторно используется для новых разделов, поэтому не требуется выполнять сжатие PDSE для удаления потерянного пространства. Каждый раздел PDSE может иметь до 15 728 639 записей. PDSE может иметь не более 123 экстентов, но не может занимать больше одного тома. При использовании оглавления PDSE оно хранится в процессорной памяти для быстрого доступа.

Наборы данных PDSE можно использовать почти вместо всех наборов данных PDS, используемых для хранения данных. Но формат PDSE не предназначен для использования в качестве замены PDS. Когда PDSE применяется для хранения загрузочных модулей, он хранит их в структурах, называемых *программными объектами (program objects)*.

Сравнение PDS с PDSE

Во многом PDSE подобен PDS. Имя каждого раздела может иметь длину до 8 байт. С точки зрения доступа к оглавлению или разделу PDS большинство интерфейсов PDSE неразличимы от интерфейсов PDS. Наборы данных PDS и PDSE обрабатываются с использованием тех же методов доступа (BSAM, QSAM, BPAM). И если вам интересно, в заданном PDS или PDSE разделы должны использовать одинаковый метод доступа.

Однако наборы данных PDSE имеют другой внутренний формат, что увеличивает их полезность. Можно использовать PDSE вместо PDS для хранения данных или программ. В PDS программы можно сохранять как *загрузочные модули (load modules)*. В PDSE программы хранятся как программные объекты. Если требуется сохранить загрузочный модуль в PDSE, его сначала необходимо преобразовать в программный объект (используя утилиту IEBCOPY).

Наборы данных PDSE имеют некоторые свойства, позволяющие увеличить продуктивность пользователей и производительность системы. Основное преимущество использования PDSE в сравнении с PDS состоит в том, что PDSE автоматически повторно использует пространство в наборе данных, не требуя периодически запускать утилиту для его реорганизации.

Кроме того, размер оглавления PDS является фиксированным, независимо от количества разделов в нем, тогда как размер оглавления PDSE является гибким и расширяемым для включения хранящихся в нем разделов.

Кроме того, система автоматически восстанавливает пространство при удалении или замене раздела и возвращает его в пул пространства, доступного для других разделов того же PDSE. Пространство можно повторно использовать без использования утилиты IEBCOPY для сжатия.

К прочим преимуществам наборов данных PDSE относятся:

- Разделы PDSE могут быть разделяемыми. Это упрощает обеспечение целостности PDSE при одновременном изменении отдельных разделов PDSE.
- Сокращенное время поиска в оглавлении. Оглавление PDSE содержит индекс, и поиск в нем осуществляется по этому индексу. Оглавление PDS организовано в алфавитном порядке, поэтому поиск в нем выполняется последовательно. Система может кешировать в памяти оглавления часто используемых наборов данных PDSE.
- Создание нескольких разделов одновременно. Например, можно открыть два DCB для одного PDSE и записать два раздела одновременно.
- Наборы данных PDSE могут содержать до 123 экстентов. Экстент представляет собой непрерывную область пространства на DASD-томе, занятую или зарезервированную для определенного набора данных.

- После записи на DASD логические записи извлекаются с пользовательских блоков и повторно разбиваются на блоки. При чтении, записи в PDSE повторно разбиваются на блоки размером, заданным в DCB. Размер блока, используемый для повторного разбиения на блоки, может отличаться от первоначального размера блока.

5.9.4 Когда набору данных недостаточно пространства

Как говорилось выше, при распределении набора данных резервируется некоторое количество пространства в блоках, дорожках или цилиндрах на диске хранения. Если полностью израсходовать это пространство, система выводит сообщение SYSTEM ABEND '0D37' (либо B37 или E37).

В этой книге не рассматриваются аварийные завершения (abnormal end, *abend*), однако при возникновении такой проблемы вам придется ее каким-то образом разрешать. При работе в сеансе редактирования вы не сможете выйти из сеанса до тех пор, пока проблема не будет разрешена.

При аварийном завершении в связи с недостатком пространства можно использовать следующие варианты разрешения:

- Если набор данных имеет формат PDS, его можно сжать следующим образом:
  а) разделите (PF 2) экран и выберите **UTILITIES** (опция 3);
  б) выберите **LIBRARIES** (опция 1) в меню выбора утилиты;
  в) укажите имя набора данных и введите C в строку опций;
  г) после завершения сжатия набора данных должно появиться сообщение COMPRESS SUCCESSFUL;
  д) теперь можно переключиться (PF 9) в сеанс редактирования и сохранить новый материал.
- Распределите набор данных большего размера и выполните копирование в него следующим образом:
  а) Разделите (PF 2) экран и выберите **UTILITIES** (опция 3), затем **DATASET** (опция 2) с другой стороны разделения.
  б) Укажите имя набора данных, получившего сообщение аварийного завершения, чтобы вывести его свойства.
  в) Распределите другой набор данных с пространством большего размера.
  г) Выберите **MOVE/COPY** (опция 3) в меню выбора утилиты, чтобы скопировать разделы из старого набора данных в новый набор данных большего размера.
  д) Просмотрите (опция 1) новый набор данных, чтобы убедиться в том, что все было скопировано правильно.
  е) Переключитесь (PF 9) обратно в сеанс редактирования, в котором произошло аварийное завершение, введите `CC` в верхнюю строку ввода и в нижнюю строку ввода, введите `CREATE` в командную строку и нажмите Enter.

ж) Введите имя нового набора данных большего размера и имя раздела для получения скопированной информации.

з) Вы снова видите сеанс редактирования, в котором произошло аварийное завершение. Введите `CAN` в командную строку. Нажмите клавишу RETURN (PF 4) для выхода из сеанса редактирования.

и) Выберите **DATASET** (опция 2) из меню выбора утилиты, чтобы удалить старый набор данных.

й) Переименуйте новый набор данных, назначив ему старое имя.

- Отмените новый материал, введенный в сеансе редактирования; для этого введите `CAN` в командную строку. Теперь вы должны иметь возможность выполнить выход без аварийного завершения; однако вся несохраненная информация будет потеряна.

5.10 Что такое VSAM?

Термин *Virtual Storage Access Method (VSAM, виртуальный метод доступа)* применяется как к типу набора данных, так и к методу доступа, используемому для управления различными типами пользовательских данных. Как метод доступа VSAM обеспечивает гораздо более сложные функции, чем другие методы доступа к дискам. VSAM хранит дисковые записи в уникальном формате, непонятном для других методов доступа.

VSAM – метод доступа для прямой и последовательной обработки записей фиксированной и переменной длины

VSAM используется главным образом для приложений. Он не используется для исходных программ, JCL или исполняемых модулей. VSAM-файлы нельзя отобразить или редактировать через ISPF.

Можно использовать VSAM, чтобы организовать записи по четырем типам наборов данных: упорядоченный по ключам, упорядоченный по вводу, линейный или с записями с относительными номерами. Основное различие между этими типами наборов данных состоит в способе хранения и доступа к их записям.

Приведем краткое описание наборов данных VSAM.

- Key Sequence Data Set (KSDS; набор данных, упорядоченный по ключам). Представляет собой наиболее используемый вариант VSAM. Каждая запись имеет одно или несколько ключевых полей, и запись можно извлечь (или вставить) по значению ключа. Это обеспечивает произвольный доступ к данным. Записи имеют переменную длину.
- Entry Sequence Data Set (ESDS; набор данных, упорядоченный по вводу). Этот вариант VSAM хранит записи в последовательном порядке. Доступ к записям осуществляется последовательно. Этот вариант используется в IMS, DB2 и z/OS UNIX.
- Relative Record Data Set (RRDS; набор данных с записями с относительными номерами).

Этот формат VSAM позволяет извлекать записи по номеру: запись 1, запись 2 и т. д. Это обеспечивает произвольный доступ и предполагает, что приложение имеет способ получения требуемых номеров записей.

- Linear Data Set (LDS; линейный набор данных).
  Представляет собой байт-ориентированный набор данных и является единственным форматом байт-ориентированных наборов данных в традиционных файлах z/OS (в отличие от файлов z/OS UNIX). Этот формат интенсивно используется многими системными функциями z/OS, однако пользовательские приложения его используют редко.
  Существует несколько дополнительных методов доступа к данным в VSAM, не перечисленных здесь. Большинство приложений использует VSAM для работы с данными, содержащими ключи.
  VSAM работает с логической областью данных, называемой управляющим интервалом (control interval, CI) и представленной на рис. 5.2. По умолчанию CI имеет размер 4 Кб, однако она может занимать до 32 Кб. CI содержит записи данных, неиспользуемое пространство, поля дескрипторов записей (record descriptor fields, RDF) и поле дескриптора CI.

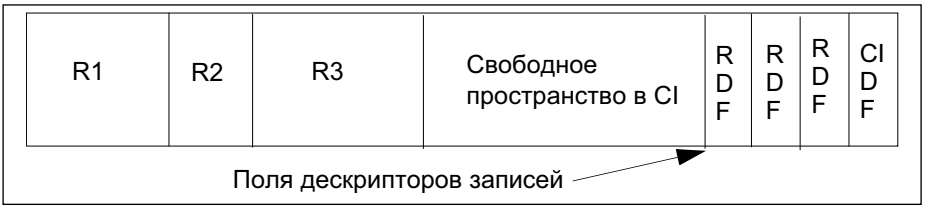


Рис. 5.2. Простой управляющий интервал VSAM

Несколько CI размещаются в управляющей области (control area, CA). Набор данных VSAM состоит из управляющих областей и индексных записей. Одной из разновидностей индексных записей является набор последовательности (sequence set), который представляет собой индекс низшего уровня, указывающий на управляющий интервал.

Данные VSAM всегда имеют переменную длину, и записи автоматически блокируются в управляющих интервалах. Ни атрибуты RECFM (F, FB, V, VB, U), ни атрибут BLKSIZE не применяются к VSAM. Для определения и удаления структур VSAM, таких, как файлы и индексы, можно использовать утилиту AMS (Access Method Services). См. пример 5.1.

Пример 5.1. Определение VSAM KSDS через AMS

```
DEFINE CLUSTER -
(NAME(VWX.MYDATA) -
VOLUMES(VSER02) -
RECORDS(1000 500)) -
DATA -
(NAME(VWX.KSDATA) -
```

```
KEYS(15 0) -
RECORDSIZE(250 250) -
BUFFERSPACE(25000) ) -
INDEX -
(NAME(VWX.KSINDEX) -
   CATALOG (UCAT1)
```

Существует множество подробностей обработки VSAM, не включенных в это краткое описание. Основной объем обработки VSAM осуществляет невидимым образом; приложение просто извлекает, изменяет, удаляет или добавляет записи на основе значений ключей.

5.11 Каталоги и VTOC

z/OS использует каталог (catalog) и оглавление тома (volume table of contents, VTOC) на каждом DASD для управления хранением и размещением наборов данных; эти понятия рассматриваются в следующих разделах:

- «Что такое VTOC?»,
- «Что такое каталог?».

z/OS также позволяет группировать наборы данных на основе исторически связанных данных, что описывается в разделе «Что такое группа поколений данных?».

5.11.1 Что такое VTOC?

z/OS требует использования определенного формата дисков, представленного на рис. 5.3. Запись 1 на первой дорожке первого цилиндра содержит метку диска. Она содержит 6-символьный серийный номер (volser) и указатель на *оглавление тома (volume table of contents, VTOC)*, которое может находиться где угодно на диске.

VTOC перечисляет наборы данных, расположенные на томе, вместе с информацией о расположении и размере каждого набора данных, а также прочими атрибутами набора данных. Для создания метки и VTOC в z/OS используется стандартная утилита ICKDSF.

Оглавление тома (VTOC) – структура, содержащая метки наборов данных

При инициализации дискового тома через ICKDSF владелец может определять расположение и размер VTOC. Размер может быть переменным и варьироваться от нескольких треков до, например, 100 треков, в зависимости от предполагаемого назначения тома. Большое число наборов данных на дисковом томе требует больше места во VTOC.

VTOC также содержит записи для всего свободного пространства на томе. При выделении пространства для набора данных системные подпрограммы анализируют записи свободного пространства, изменяют их и создают новую запись VTOC. Наборы данных всегда занимают целое число дорожек (или цилиндров) и начинаются с начала дорожки (или цилиндра).

Можно также создать VTOC с индексом. Индекс VTOC в действительности представляет собой набор данных с именем SYS1.VTOCIX.*volser*, содержащий записи, ор-

ганизованные в алфавитном порядке имен наборов данных с указателями на записи VTOC. Он также содержит битовые карты свободного пространства тома. Индекс VTOC позволяет пользователю гораздо быстрее находить наборы данных.

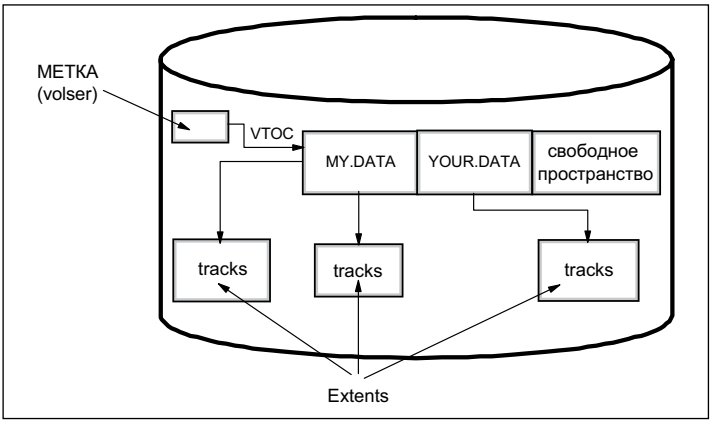


Рис. 5.3. Метка диска, VTOC и экстенты

5.11.2 Что такое каталог?

Каталог описывает атрибуты наборов данных и указывает тома, на которых расположены наборы данных. Если набор данных каталогизирован, к нему можно обращаться по имени, не указывая, где хранится набор данных. Наборы данных можно каталогизировать, раскаталогизировать или рекаталогизировать. Все управляемые системой наборы данных, расположенные на DASD, автоматически заносятся в каталог. Каталогизация наборов данных на магнитной ленте необязательна, однако она может упростить работу пользователей.

Каталог – описывает атрибуты набора данных, включая место расположения набора данных

В z/OS главный каталог и пользовательские каталоги содержат расположения наборов данных. Можно осуществлять каталогизацию как наборов данных на дисках, так и наборов данных на ленте.

Для того чтобы найти запрашиваемый набор данных, z/OS должна знать следующую информацию:

- имя набора данных;
- имя тома;
- устройство (тип тома, например диск 3390 или привод для лент 3590).

Все три значения можно задать через панели ISPF или в JCL. Однако тип устройства и имя тома часто несущественны для конечного пользователям или приложения. Системный каталог используется для хранения и извлечения значений UNIT или VOLUME для набора данных. В своей самой простой форме каталог содержит тип устройства и имя тома для любого каталогизированного набора данных. Системный каталог предоставляет простую функцию поиска. При употреблении этого средства, пользователю требуется только лишь задать имя набора данных.

Главные каталоги и пользовательские каталоги

Система z/OS всегда содержит по меньшей мере один главный каталог (master catalog). Если она содержит только один каталог, этот каталог является главным каталогом и записи о расположении всех наборов данных будут храниться в нем. Однако один каталог не обеспечит ни эффективность, ни гибкость, поэтому стандартная система z/OS использует главный каталог и множество подключенных к нему пользовательских каталогов (*user catalogs*), как показано на рис.5.4.

В пользовательском каталоге хранится имя и расположение набора данных (dsn/volume/unit). В главном каталоге обычно хранится только HLQ набора данных с именем пользовательского каталога, содержащего информацию о расположении всех наборов данных с заданным HLQ. HLQ называется алиасом (alias).

На рис. 5.4 имя набора данных главного каталога – `SYSTEM.MASTER.CATALOG`. В этом главном каталоге хранится полное имя набора данных и расположение всех наборов данных с префиксом SYS1, например SYS1.A1. В главном каталоге определено две записи HLQ (алиаса): IBMUSER и USER. Команда, определившая IBMUSER, содержала имя набора данных пользовательского каталога, содержащего все полные имена наборов данных IBMUSER вместе с их расположением. Это же относится и к HLQ (алиасу) USER.

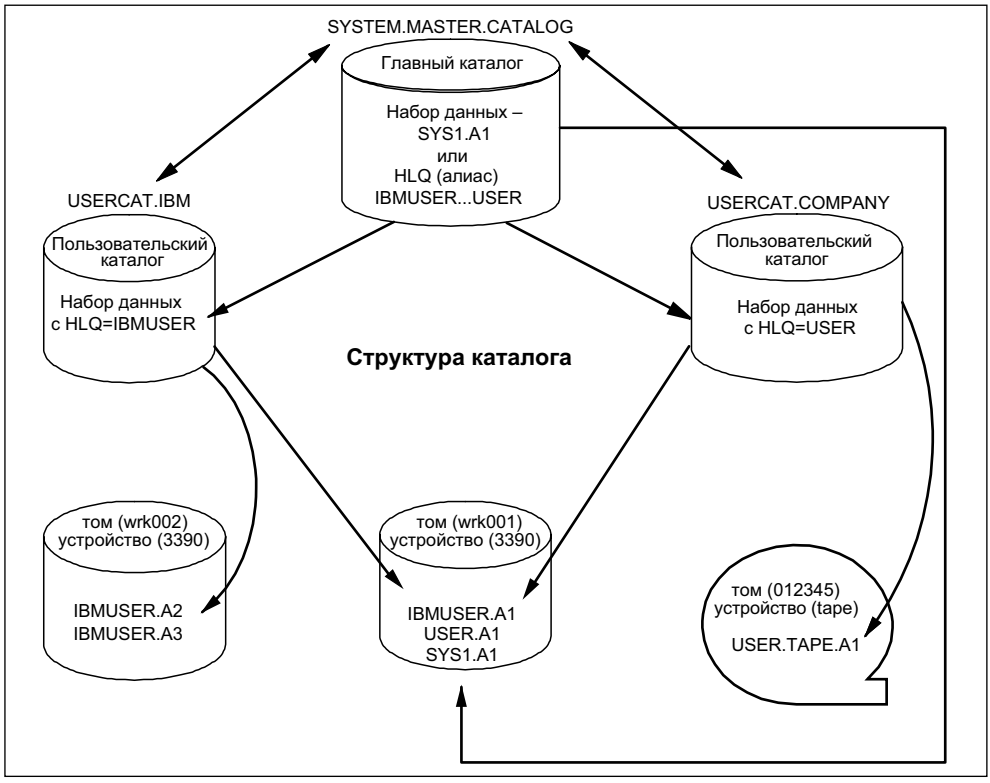


Рис. 5.4. Схема каталога

Если запрашивается SYS1.A1, главный каталог возвращает информацию о расположении, имени тома (WRK001) и типе устройства (3390). Если запрашивается IBMUSER.A1, главный каталог перенаправляет запрос в USERCAT.IBM, после чего USERCAT.IBM возвращает информацию о расположении инициатору запроса.

В качестве дальнейшего примера рассмотрим следующие операторы DEFINE:

```
DEFINE ALIAS ( NAME ( IBMUSER ) RELATE ( USERCAT.IBM ) )
DEFINE ALIAS ( NAME ( USER ) RELATE ( USERCAT.COMPANY ) )
```

Эти операторы используются для записи имен алиасов IBMUSER и USER в главный каталог с именем пользовательского каталога, хранящего полные имена наборов данных и информацию о расположении. Если IBMUSER.A1 каталогизирован, для его распределения используется JCL-оператор:

```
//INPUT DD DSN=IBMUSER.A1,DISP=SHR
```

Если IBMUSER.A1 не каталогизирован, для его распределения используется JCL-оператор:

```
//INPUT DD DSN=IBMUSER.A1,DISP=SHR,VOL=SER=WRK001,UNIT=3390
```

Как правило, все пользовательские наборы данных в инсталляции z/OS каталогизируются. Некаталогизированные наборы данных редко нужны, и их использование часто связано с проблемами восстановления или установкой нового программного обеспечения. Наборы данных, создаваемые через ISPF, каталогизируются автоматически.

Использование резервного главного каталога

Итак, что произойдет, если инсталляция утратит свой главный каталог или если главный каталог будет каким-то образом поврежден? Это вызовет серьезные проблемы и потребует быстрых действий по восстановлению.

Чтобы избежать потенциальных проблем, большинство программистов определяют резервный главный каталог. Этот резервный каталог задается при запуске системы. В этом случае системному программисту рекомендуется хранить резервный каталог на отдельном томе, отличном от тома, содержащего главный каталог (для защиты от ситуаций, когда том становится недоступным).

5.11.3 Что такое группа поколений данных?

В z/OS возможна каталогизация последовательных изменений или поколений связанных данных. Они называются группами поколений данных (generation data group, GDG).

Каждый набор данных в GDG называется поколением (generation) или набором данных поколения (generation data set, GDS). Группа поколений данных (GDG) представляет собой набор исторически связанных наборов данных, отличных от VSAM, организованных в хронологическом порядке. Другими словами, каждый набор данных исторически связан с другими наборами данных в группе.

В GDG поколения могут иметь похожие или непохожие DCB-атрибуты и организацию наборов данных. Если атрибуты и организации всех поколений в группе идентичны, поколения могут извлекаться как единый набор данных.

Группирование связанных наборов данных имеет ряд преимуществ. Например:

- ко всем наборам данных в группе можно обращаться, используя общее имя;
- операционная система может хранить поколения в хронологическом порядке;
- устаревшие поколения могут автоматически удаляться операционной системой.

Наборы данных поколения имеют последовательно упорядоченные абсолютные и относительные имена, представляющие их возраст. Подпрограммы управления каталогом операционной системы используют абсолютные имена поколений. Старые наборы данных имеют меньшие абсолютные номера. Относительное имя представляет собой целое число со знаком, используемое для обращения к последнему (0), предпоследнему (-1) и так далее поколениям.

Например, имя набора данных LAB.PAYROLL(0) относится к последнему набору данных группы; LAB.PAYROLL(-1) относится к предпоследнему набору данных и т. д. Относительный номер можно также использовать для каталогизации нового поколения (+1). База группы поколений данных (GDG) размещается в каталоге до каталогизации наборов данных поколения. Каждая GDG представлена записью базы GDG.

Для новых наборов данных, не управляемых системой, если не задан том и не открыт набор данных, система не выполняет каталогизацию набора данных. Новые управляемые системой наборы данных всегда каталогизируются при распределении с назначением тома из группы памяти.

5.12 Роль DFSMS в управлении пространством

В системе z/OS управление пространством включает в себя выделение пространства, размещение, мониторинг, миграцию, резервное копирование, возврат, восстановление и удаление наборов данных. Эти действия можно выполнять вручную либо с использованием автоматизированных процессов. Если управление данными автоматизировано, операционная система определяет расположение объектов и автоматически управляет резервным копированием наборов данных, их перемещением, выделением пространства и безопасностью. Стандартная рабочая система z/OS включает как ручные, так и автоматизированные процессы управления наборами данных.

В зависимости от настройки системы z/OS и ее устройств хранения пользователь или программа может напрямую контролировать многие аспекты управления данными, и на заре развития операционных систем пользователи были обязаны это делать. Однако в настоящее время инсталляции z/OS все больше полагаются на параметры управления данными и ресурсами, характерные для конкретной инсталляции, а также на средства управления пространством, такие, как DFSMS, позволяющие автоматизировать использование пространства для наборов данных.

Управление данными включает следующие основные задачи:

- выделение пространства на DASD-томах;
- автоматическое извлечение каталогизированных наборов данных по имени;
- подключение томов на магнитной ленте в приводе;
- установление логического соединения между приложением и носителем;
- управление доступом к данным;
- передача данных между приложением и носителем.

Основным средством управления пространством в z/OS является использование компонента DFSMS операционной системы. DFSMS выполняет основные функции управления данными, хранением, программами и устройствами в системе. DFSMS представляет собой набор продуктов, и один из этих продуктов (DSFMSdfp) необходим для работы z/OS. DFSMS, вместе с аппаратными продуктами и параметрами инсталляции для управления данными и ресурсами, обеспечивает память, управляемую системой, в среде z/OS.

Основным элементом DFSMS является подсистема SMS (Storage Management Subsystem, подсистема управления памятью). Используя SMS, системный программист или администратор памяти определяет политики, автоматизирующие управление внешней памятью и аппаратными устройствами. Эти политики описывают свойства распределения данных, цели производительности и доступности, требования к резервному копированию и сроку хранения, а также требования к памяти в системе. SMS управляет этими политиками в системе, тогда как ISMF (Interactive Storage Management Facility) обеспечивает пользовательский интерфейс для определения и управления политиками.

SMS – Storage Management Subsystem (Подсистема управления памятью)

Наборы данных, распределяемые посредством SMS, называются наборами данных, управляемыми системой, или наборами данных, управляемыми SMS. При распределении или определении набора данных, использующего SMS, требуется задать требования к набору данных через класс данных, класс памяти и класс управления. Обычно не требуется задавать эти классы, так как администратор памяти устанавливает программы автоматического назначения классов (automatic class selection, ACS), определяющих, какие классы следует использовать для того или иного набора данных.

DFSMS содержит набор конструкций, пользовательских интерфейсов и программ (использующих продукты DFSMS), помогающих администратору памяти. Основная логика DFSMS, в частности программы ACS, код ISMF и конструкции, находится в DFSMSdfp™. DFSMShsm™ и DFSMSdss™ реализуют конструкцию класса управления. Используя DFSMS, системный программист z/OS или администратор памяти может определять цели производительности и требования к доступности данных, создавать модельные определения данных для стандартных наборов данных и автоматизировать резервное копирование данных. DFSMS может автоматически предоставлять исходя из политики инсталляции эти службы и атрибуты определения данных наборам данных при их создании. Продукты управления памятью IBM определяют расположение данных, управляют резервным копированием данных, управляют использованием пространства и обеспечивают безопасность данных.

5.13 Файловые системы z/OS UNIX

Файловую систему UNIX можно рассматривать как контейнер, содержащий части всего дерева директорий UNIX. В отличие от традиционной библиотеки z/OS файловая система UNIX является иерархической и байт-ориентированной. Чтобы найти файл в файловой системе UNIX, нужно выполнить поиск в одной или нескольких директориях (directories); см. рис. 5.5. В UNIX отсутствует понятие каталога (catalog), указывающего непосредственно на файл, используемое в z/OS.

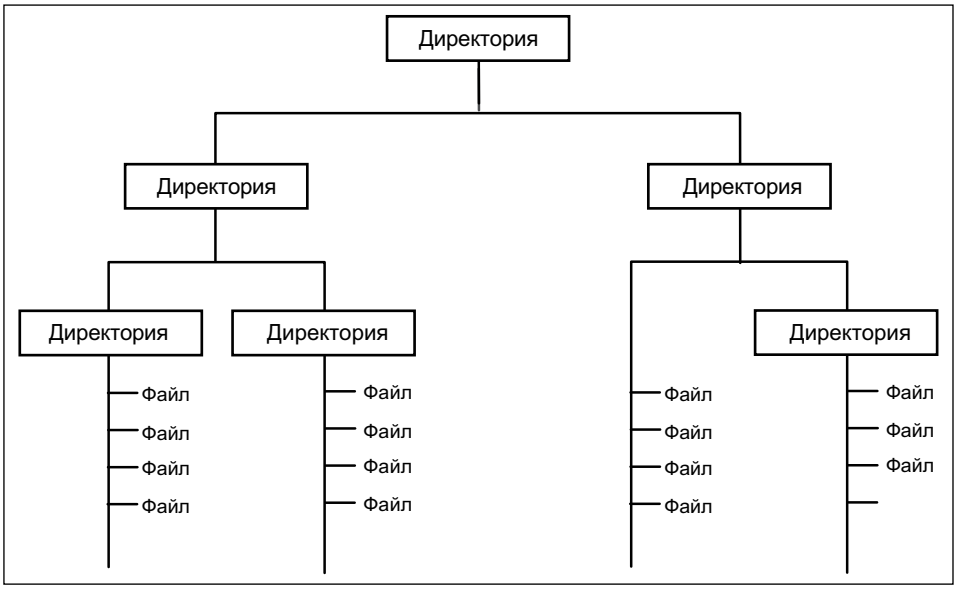


Рис. 5.5. Иерархическая структура файловой системы

z/OS UNIX System Services (z/OS UNIX) позволяет пользователям z/OS создавать файловые системы UNIX и деревья директорий файловой системы в z/OS, а также осуществлять доступ к файлам UNIX в z/OS и других системах. В z/OS файловая система UNIX подключается (монтируется) к пустой директории системным программистом (или пользователю с полномочиями монтирования).

В z/OS UNIX допускается использование следующих типов файловых систем:

- zSeries File System (zFS) – файловая система, хранящая файлы в линейных наборах данных VSAM;
- Hierarchical File System (HFS) – монтируемая файловая система, в настоящее время вытесняемая zFS;
- z/OS Network File System (z/OS NFS) – файловая система, позволяющая системе z/OS осуществлять доступ к удаленной файловой системе UNIX (z/OS UNIX или другой) через TCP/IP, как если бы она была частью локального дерева директорий z/OS;
- Temporary File System (TFS) – временная физическая файловая система, поддерживающая монтируемые файловые системы, создаваемые в памяти.

Как и в других файловых системах UNIX, путь указывает на файл и состоит из имен директорий и имени файла. Полное имя файла, состоящее из имен всех директорий в пути к файлу и имени самого файла, может иметь длину до 1 023 байт.

Путь состоит из имен отдельных директорий и имени файла, разделенных символом прямого слэша, например:

/dir1/dir2/dir3/MyFile

Подобно UNIX, в z/OS UNIX имена файлов и директорий являются чувствительными к регистру. Например, в одной директории могут существовать два разных файла с именами MYFILE и MyFile.

Файлы в файловой системе HFS представляют собой последовательные файлы, доступ к которым осуществляется как к байтовым потокам. Для этих файлов неприменимо понятие записи в смысле, отличном от структуры, определенной приложением.

Набор данных zFS, содержащий файловую систему UNIX, представляет собой набор данных z/OS (линейный набор данных VSAM). Наборы данных zFS и наборы данных z/OS могут располагаться на одном DASD-томе; z/OS содержит команды управления использованием пространства в zFS.

Интеграция файловой системы zFS с существующими службами управления файловой системой z/OS обеспечивает автоматизированные возможности управления файловой системой, которые могут быть недоступны на других платформах UNIX. Такая интеграция позволяет владельцам файлов тратить меньше времени на такие задачи, как резервное копирование и восстановление файловых систем целиком.

5.13.1 Сравнение наборов данных z/OS и файлов файловой системы

Многие элементы UNIX имеют аналоги в операционной системе z/OS. В частности, организация пользовательского каталога (user catalog) аналогична пользовательской директории (user directory; /u/ibmuser) в файловой системе.

В z/OS префикс пользователя, назначаемый набору данных z/OS, указывает на пользовательский каталог (user catalog). Обычно один пользователь владеет всеми наборами данных, имя которых начинается с его пользовательского префикса. Например, наборы данных, принадлежащие TSO/E-пользователю IBMUSER, начинаются со старшего квалификатора (префикса) IBMUSER. Могут существовать наборы данных IBMUSER.C, IBMUSER.C.OTHER и IBMUSER.TEST.

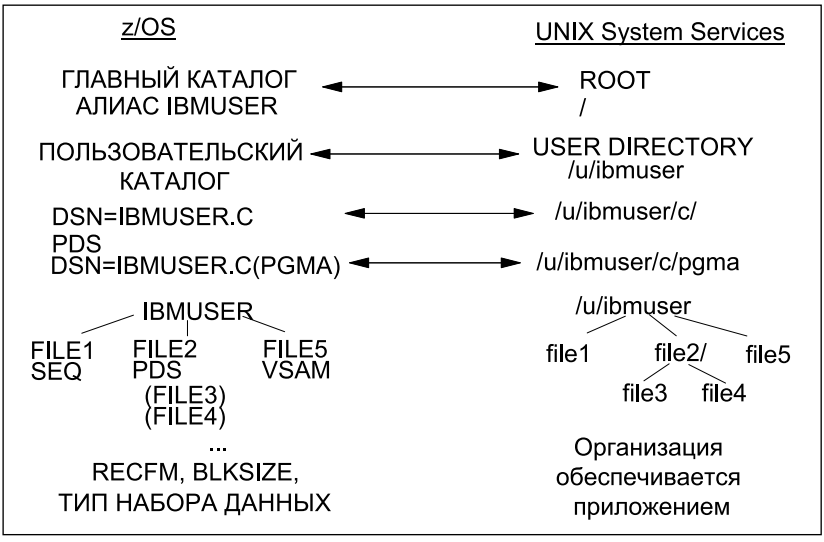


Рис. 5.6. Сравнение наборов данных z/OS и файлов файловой системы

В файловой системе UNIX ibmuser имеет пользовательскую директорию (user directory) с названием /u/ibmuser. В этой директории может находиться поддиректория /u/ibmuser/c. Тогда /u/ibmuser/c/pgma указывает на файл pgma (рис. 5.6).

Среди существующих типов наборов данных z/OS секционированный набор данных (PDS) больше всего похож на пользовательскую директорию в файловой системе. Секционированный набор данных, например IBMUSER.C, может содержать разделы (файлы) PGMA, PGMB, обозначаемые IBMUSER.C(PGMA), IBMUSER.C(PGMB) и т. д. Аналогично поддиректория /u/ibmuser/c может содержать много файлов, таких, как pgma, pgmb и т. д.

Все данные, записываемые в файловую систему HFS, могут считываться любыми программами сразу же после их записи. Запись данных на диск выполняется, когда программа выдает **`fsync()`**.

5.14 Работа с файловой системой zFS

z/OS Distributed File Service (DFS™) zSeries File System (zFS) является файловой системой в z/OS UNIX System Services (z/OS UNIX), которую можно использовать вместе с файловой системой HFS (Hierarchical File System). Файловая система zFS содержит файлы и директории, к которым можно осуществлять доступ через интерфейсы программирования приложений z/OS UNIX. Эти файловые системы могут поддерживать списки управления доступом (ACL). Файловые системы zFS могут монтироваться к иерархии z/OS UNIX вместе с другими локальными (или удаленными) файловыми системами (например, HFS, TFS, AUTOMNT и NFS).

Блок серверных сообщений (SMB) службы Distributed File Service обеспечивает работу сервера, который делает файлы и наборы данных z/OS UNIX доступными для SMB-клиентов. К поддерживаемым наборам данных относятся последовательные наборы данных (на DASD), PDS и PDSE, а также наборы данных VSAM. Поддержка наборов данных обычно называется поддержкой файловой системы ориентированной на записи RFS (Record File System). Протокол SMB поддерживается посредством использования TCP/IP в z/OS. Этот коммуникационный протокол позволяет клиентам осуществлять доступ к разделяемым директориям и разделяемым принтерам. Клиенты PC в сети могут использовать функции совместного доступа к файлам и принтерам, включенные в их операционные системы.

К поддерживаемым SMB-клиентам относятся Windows XP Professional, Windows Terminal Server в Windows 2000, Windows Terminal Server в Windows 2003 и LINUX. В то же время эти файлы могут использоваться совместно с локальными приложениями z/OS UNIX и клиентами DCE DFS.

Дополнительные сведения. Использование DFS описывается в публикации *z/OS DFS Administration*. Эту и другие публикации IBM можно найти на веб-сайте z/OS Internet Library

```
http://www.ibm.com/servers/eserver/zseries/zos/bkserv/
```

5.15 Заключение

Набор данных представляет собой набор логически связанных данных; он может содержать исходную программу, библиотеку программ или файл записей данных, используемых обрабатывающей программой. *Записи* набора данных являются основной единицей информации, используемой обрабатывающей программой.

Пользователи должны определить количество пространства, выделяемого для набора данных (до его использования), или же эти операции размещения необходимо автоматизировать с использованием DFSMS. Используя DFSMS, системный программист z/OS или администратор памяти может определять цели производительности и требования к доступности данных, создавать модельные определения данных для стандартных наборов данных и автоматизировать резервное копирование данных. DFSMS может автоматически предоставлять, исходя из политики инсталляции, эти службы и атрибуты определения данных наборам данных при их создании. Можно использовать и другие продукты управления памятью для определения расположения данных, управления резервным копированием данных, управления использованием пространства и обеспечения безопасности данных.

Почти вся обработка данных в z/OS является ориентированной на записи. Байт-ориентированные файлы не представлены в традиционной обработке, хотя они являются стандартным компонентом z/OS UNIX. Записи и физические блоки z/OS имеют один из нескольких определенных форматов. Большинство наборов данных имеют DCB-атрибуты, включающие формат записи (RECFM: F, FB, V, VB, U), максимальную длину логической записи (LRECL) и максимальный размер блока (BLKSIZE).

Библиотеки z/OS называются также секционированными наборами данных (PDS или PDSE) и содержат разделы. Исходные программы, параметры управления системой и приложениями, JCL и исполняемые модули почти всегда находятся в библиотеках.

Виртуальный метод доступа (Virtual Storage Access Method, VSAM) представляет метод доступа, обеспечивающий гораздо более сложные функции, чем другие дисковые методы доступа. VSAM используется главным образом для приложений и не допускает редактирования с использованием ISPF.

Наборы данных z/OS имеют имена длиной не более 44 символов в верхнем регистре, разделяемые точками на квалификаторы длиной не более 8 байт. Старший квалификатор (HLQ) может быть фиксированным и устанавливаться средствами управления безопасностью системы, однако остальная часть имени набора данных задается пользователем. Существуют определенные соглашения именования.

Существующий набор данных можно найти, если известно имя набора данных, том и тип устройства. Если набор данных каталогизирован, эти требования можно сократить до одного имени набора данных. Системный каталог представляет собой единую логическую функцию, хотя его данные могут быть распределены по главному каталогу и множеству пользовательских каталогов. В действительности почти все дисковые наборы данных являются каталогизированными. Побочный эффект использования каталогов состоит в том, что все (каталогизированные) наборы данных должны иметь уникальные имена.

Файл в файловой системе UNIX может быть либо текстовым, либо двоичным. В текстовом файле каждая строка текста отделена разделителем новой строки. Двоичный файл состоит из последовательностей двоичных слов (потока байтов), и для этих файлов неприменимо понятие записи в смысле, отличном от структуры, определенной приложением. Приложение, осуществляющее чтение файла, отвечает за интерпретацию формата данных; z/OS воспринимает всю иерархию файловой системы UNIX как собрание наборов данных. Каждый набор данных является монтируемой файловой системой.

Основные термины в этой главе		
Размер блока	Каталог	Набор данных
Старший квалификатор (HLQ)	Библиотека	Длина логической записи (LRECL)
Раздел	PDS/**PDSE**	Формат записи (RECFM)
Подсистема управления памятью (SMS)	Виртуальный метод доступа (VSAM)	VTOC

5.16 Контрольные вопросы

Чтобы проверить понимание материала этой главы, ответьте на следующие вопросы:

1. Что такое набор данных? Какие типы наборов данных используются в z/OS?
2. Почему в z/OS необходимо использовать уникальные имена наборов данных?
3. Почему используется PDS?
4. Используются ли библиотеки приложениями? Почему?
5. Что определяет максимальный размер файла, используемого в традиционной системе UNIX? Существует ли аналогичный предел в z/OS?
6. Существуют ли закономерности в именах временных наборов данных?
7. Какие специальные символы используются для определения временного набора данных в JCL-потоке?
8. Информация о наборе данных, предоставляемая ISPF 3.4, является полезной. Почему бы не выводить всю информацию на базовой панели списка наборов данных?
9. Мы создали исходную библиотеку в одном из упражнений и задали 80-байтовые записи фиксированной длины. Почему?
10. Для упражнений используется дисковый том *WORK02*. Можно ли распределить набор данных на других томах? На любом томе?
11. Какая информация о наборе данных хранится в каталоге? Какие операнды DD были бы нужны, если бы набор данных не находился в каталоге?
12. В чем состоит различие между главным каталогом и пользовательским каталогом?

5.17 Упражнения

Практические упражнения в этой главе помогают развить навыки работы с наборами данных через ISPF. Эти навыки необходимы для выполнения практических упражнений в остальной части книги.

Для выполнения практических упражнений вам или вашей подгруппе нужен идентификатор и пароль пользователя TSO (для этого следует обратиться к преподавателю).

Упражнения посвящены следующим темам:

- «Изучение опции ISPF 3.4»;
- «Распределение набора данных с использованием опции ISPF 3.2»;
- «Копирование исходной библиотеки»;
- «Работа с разделами набора данных»;
- «Вывод списка наборов данных (и другие опции ISPF 3.4)»;
- «Выполнение поиска по каталогу»;

Замечание. Не следует путать клавишу Enter в терминале 3270 с клавишей Enter в PC. Большинство эмуляторов 3270 позволяют пользователю назначить эти функции любой клавише на клавиатуре, и мы предполагаем, что функция Enter терминала 3270 назначена правой клавише CTRL. Однако некоторые пользователи z/OS предпочитают, чтобы PC-клавиша Enter выполняла функцию Enter терминала 3270 и чтобы Shift-Enter (или клавиша Enter на дополнительной клавиатуре) выполняла функцию New Line терминала 3270.

5.17.1 Изучение опции ISPF 3.4

Одной из наиболее полезных панелей ISPF является опция 3.4. Это означает, что в главном меню ISPF следует выбрать опцию 3 (Utilities), после чего выбрать опцию 4 (Dslist). Эту последовательность можно сократить путем ввода 3.4 в главное меню, или =3.4 из любой панели.

Многие пользователи ISPF работают практически исключительно с панелями 3.4. Здесь мы рассмотрим некоторые функции 3.4 и рассмотрим другие функции в последующих упражнениях в этой книге. Будьте внимательны при работе с опциями 3.4: они могут выполнять изменения как на уровне отдельных пользователей, так и на уровне всей системы.

Пользователи z/OS обычно применяют опцию ISPF 3.4 для проверки наборов данных на DASD-томе или для изучения свойств определенного набора данных. Пользователям может быть необходимо знать следующее:

- какие наборы данных находятся на этом томе?
- сколько различных типов наборов данных находится на томе?
- каковы DCB-свойства отдельного файла?

Ответим на эти вопросы, используя том WORK02 или другой том, определенный преподавателем.

1. В панели 3.4 введите `WORK02` в поле Volume Serial. Не следует что-либо вводить в строку Option==> или в поле Dsname Level.
2. Используйте PF8 и PF7 для прокрутки выведенного списка наборов данных.
3. Используйте PF11 и PF10 для боковой прокрутки, чтобы вывести больше информации. В данном случае выполняется не совсем прокрутка; дополнительная информация выдается только при использовании PF11 или PF10.

Первый экран PF11 выводит дорожки, процент использования, XT и тип устройства. Значение XT означает количество *экстентов*, применяемых для получения показанного количества дорожек. Функции утилиты ISPF могут определить количество пространства, фактически используемого некоторыми наборами данных, и если это возможно, выводится процентный показатель.

Следующий экран, PF11, выводит DCB-свойства DSORG, RECFM, LRECL и BLKSIZE.

PS Последовательный набор данных (QSAM, BSAM).
PO Секционированный набор данных.
VS Набор данных VSAM.
blank Неизвестная организация (или данные не существуют).

Параметры RECFM, LRECL и BLKSIZE должны быть вам знакомы. В некоторых случаях, обычно когда не используется стандартный метод доступа или когда данные не были записаны, эти параметры определить невозможно. В наборах данных VSAM нет прямого аналога для этих параметров и вместо них выводятся вопросительные знаки.

Выберите другой том, для которого доступен больший диапазон свойств. Преподаватель должен сообщить вам серийные номера томов. Другой способ найти такой том состоит в использовании опции 3.2 для поиска местонахождения SYS1.PARMLIB и последующем изучении этого тома.

5.17.2 Распределение набора данных с использованием опции ISPF 3.2

ISPF обеспечивает удобный метод распределения наборов данных. В этом приложении вам предстоит создать новую библиотеку, которую можно использовать позже в этом курсе для хранения исходных данных программы. Новые наборы данных должны размещаться на томе *WORK02* и иметь имя *yourid*.LIB.SOURCE (где *yourid* – ваш идентификатор пользователя).

В этом упражнении предположим, что достаточно использовать 10 дорожек для первичного пространства и 5 дорожек для вторичных экстентов и что достаточно использовать 10 блоков для оглавления. Кроме того, мы знаем, что в библиотеке требуется хранить 80-байтовые записи фиксированной длины. Для этого нужно выполнить следующие действия:

1. Войдите в главное меню ISPF.
2. Перейдите к опции 3.2 или перейдите к опции 3 (Utilities) и затем к опции 2 (Data Set).
3. Введите букву A в поле Option ==>, но пока не нажимайте Enter.
4. Введите имя нового набора данных в поле Data Set Name, но пока не нажимайте Enter. Имя может указываться в одинарных кавычках (например, '*yourid*.LIB.SOURCE') или без кавычек (LIB.SOURCE); во втором случае TSO/ISPF в качестве HLQ автоматически использует идентификатор текущего пользователя TSO.
5. Введите *WORK02* в поле Volume Serial и нажмите Enter.
6. Заполните следующие поля и нажмите Enter:
   - Space Units = TRKS;
   - Primary quantity = 10;

- Secondary quantity = 5;
- Directory blocks = 10;
- Record format = FB;
- Record length = 80;
- Block size = 0 (это сообщает z/OS, что нужно выбрать оптимальное значение);
- Data set type = PDS.

В результате после этих действий должен быть распределен новый PDS на томе *WORK02*. В правом верхнем углу должно появиться следующее сообщение:

```
Menu RefList Utilities Help
------------------------------------------------------------------
Data Set Utility Data set allocated
Option ===>
A Allocate new data set C Catalog data set
.....
```

5.17.3 Копирование исходной библиотеки

Для упражнений нужны некоторые исходные программы, находящиеся в *ZPROF*. ZSCHOLAR.LIB.SOURCE на томе WORK02. Существует несколько способов скопировать наборы данных (включая библиотеки). Можно выполнить следующие действия:

1. Перейдите к опции ISPF 3.3 (Utilities, Move/Copy).
2. На первой панели:
   а) Введите C в поле Option==>.
   б) Введите '*ZPROF*.ZSCHOLAR.LIB.SOURCE' в поле Data Set Name. В данном случае нужны одинарные кавычки.
   в) Значение Volume Serial вводить не требуется, так как набор данных каталогизирован.
   г) Нажмите Enter.
3. На второй панели:
   а) Введите '*yourid*.LIB.SOURCE' в поле Data Set Name и нажмите Enter. Если такой PDS не существует, введите `1` для наследования атрибутов исходной библиотеки. Это должно сгенерировать на панели список всех разделов входной библиотеки:
   б) Нажмите `S` перед именем каждого раздела, после чего нажмите Enter.

      Это копирует все указанные разделы из исходной библиотеки в целевую библиотеку. В качестве входного набора данных можно было задать '*ZPROF*.ZSCHOLAR.LIB.SOURCE(*)'; это вызвало бы автоматическое копирование всех разделов. Это один из немногих случаев, где в именах наборов данных z/OS используются *подстановочные знаки*.
4. Создайте еще другую библиотеку и переместите в нее несколько разделов из LIB.SOURCE. Назовите ее '*yourid*.MOVE.SOURCE'. Убедитесь в том, что перемещенные

разделы находятся в новой библиотеке и отсутствуют в старой. Скопируйте эти разделы обратно в библиотеку LIB. Убедитесь в том, что они существуют в обеих библиотеках.

5. Переименуйте раздел в библиотеке MOVE. Переименуйте библиотеку MOVE, задав ей имя '*yourid*.TEST.SOURCE'.

5.17.4 Работа с разделами набора данных

Существует несколько способов добавления нового раздела в библиотеку. Требуется создать новый раздел с именем TEST2 и добавить его в библиотеку, которую мы перед этим редактировали.

1. В главном меню ISPF выберите опцию 2.
2. Введите имя библиотеки, не указывая имя раздела, например *yourid*.JCL. Это выводит список имен разделов, уже существующих в библиотеке.
3. Убедитесь в том, что раздел EDITTEST имеет то же содержимое, какое использовалось ранее:
   а) Если необходимо, прокрутите до имени раздела EDITTEST;
   б) Переместите курсор влево от этой строки;
   в) Введите `S` и нажмите Enter;
   г) Просмотрите предыдущую работу, чтобы убедиться в том, что она не изменилась;
   д) Нажмите PF3 для выхода («back out of») из раздела EDITTEST. Снова появится список имен разделов библиотеки.
4. Введите `S TEST2` в командную строку вверху экрана и нажмите Enter. (S TEST2 читается как «select TEST2».) Эта команда создаст раздел TEST2 и переведет экран в режим ввода.
5. Введите несколько строк текста, используя обсуждавшиеся ранее команды и функции.
6. Нажмите PF3, чтобы сохранить TEST2, и выполните выход.
7. Нажмите PF3 повторно, чтобы выйти из функции ISPF Edit.

В дальнейшем, при редактировании чего-либо или использовании других функций ISPF, мы будем говорить просто «Введите xxx». Это означает, что нужно (1) ввести xxx и (2) нажать клавишу Enter. Клавиша New Line (на которой написано Enter) используется только для перемещения курсора на экране.

5.18 Вывод списка наборов данных и другие опции ISPF 3.4

Перейдите в панель ISPF 3.4. Введите *yourid* в поле Dsname Level и нажмите Enter. Это должно вывести все каталогизированные наборы данных в системе с заданным HLQ. Альтернативный вариант состоит в том, чтобы оставить поле Dsname Level пустым и ввести `WORK02` в поле Volume Serial; это выведет все наборы данных на указанном

томе. (Если используются оба поля, список будет содержать только каталогизированные наборы данных с заданным HLQ, находящиеся на заданном томе.)

Некоторые функции можно вызвать путем ввода соответствующей буквы перед именем набора данных. Например, можно поместить курсор перед именем одного из наборов данных и нажать PF1 (Help). В панели Help появится список всех строчных команд, которые можно использовать из списка имен наборов данных в панели 2.4. Не экспериментируйте с ними, не имея полного понимания их функций. Не все эти функции применимы к данному учебному классу. Применимые команды:

E Редактирование набора данных.

B Просмотр набора данных.

D Удаление набора данных.

R Переименование набора данных.

Z Сжатие библиотеки PDS для восстановления потерянного пространства.

C Каталогизация набора данных.

U Раскаталогизация набора данных.

При выводе списка разделов (как и при редактировании или просмотре библиотеки) доступно несколько строчных команд:

S Выбор данного раздела для редактирования или просмотра.

R Переименование раздела.

D Удаление раздела.

5.18.1 Выполнение поиска по каталогу

Опцию ISPF 3.4 можно использовать для поиска в каталоге по частям имен. Дополнительные сведения об этой важной функции см. в справке PF1 Help; для этого:

1. Выберите опцию 3.4.
2. Нажмите PF1 для вывода справки и выберите **Display a data set list**. Нажмите Enter для прокрутки информационных панелей.
3. Выберите **Specifying the DSNAME LEVEL**. Нажмите Enter для прокрутки информационных панелей.
4. Нажмите PF3 для выхода из справки.

Обратите внимание на то, что в опции 3.4 в поле DSNAME LEVEL не применяются кавычки и что идентификатор пользователя TSO/E не применяются автоматически в качестве префикса для имен в этом поле. Это одно из немногих исключений из общего правила назначения имен наборов данных в TSO.

6

Использование JCL и SDSF

Цель. Как техническому специалисту в сфере мэйнфрейм-вычислений, вам потребуется знание языка JCL, сообщающего z/OS, какие ресурсы нужны для обработки пакетного задания или запуска системной задачи.
После завершения работы над этой главой вы сможете:

- объяснить, каким образом JCL работает в системе, описать методы; кодирования в JCL и некоторые наиболее важные операторы и ключевые слова;
- создать простое задание и передать его на выполнение;
- просмотреть выходные данные своего задания через SDSF.

6.1 Что такое JCL?

Язык управления заданиями (Job Control Language, JCL) используется для того, чтобы сообщить системе, какую программу следует выполнить, и представить описание входных и выходных данных программы. Можно *передать на выполнение (submit)* JCL для пакетной обработки или *запустить (start)* JCL-процедуру (PROC), которая считается запускаемой задачей. Подробное изучение JCL может быть сложным, но общие понятия достаточно просты. Кроме того, по меньшей мере в 90% случаев используется небольшая часть JCL. В этой главе рассматриваются некоторые опции JCL.

JCL – сообщает системе, какую программу следует выполнить, и представляет описание входных и выходных данных программы

Если программистам приложений нужны некоторые знания JCL, то аналитики производственного контроля должны быть *очень* квалифицированными специалистами в JCL, чтобы создавать, осуществлять мониторинг, исправлять и перезапускать повседневные пакетные задачи компании.

Существует три основных оператора JCL:

JOB Передает в систему имя задания (jobname) для пакетной задачи. Может включать учетную информацию и несколько параметров задания.

EXEC Передает имя исполняемой программы. Задание может содержать множество операторов EXEC. Каждый оператор EXEC в одном задании является шагом задания *(job step)*.

DD Определение данных (Data Definition) передает входные и выходные данные исполняемой программы, заданной оператором EXEC. Этот оператор связывает набор данных или другое устройство ввода-вывода или функцию с DD-именем, заданным в программе. Операторы DD связаны с определенным шагом задания.

На рис. 6.1 представлен базовый синтаксис JCL-кода.

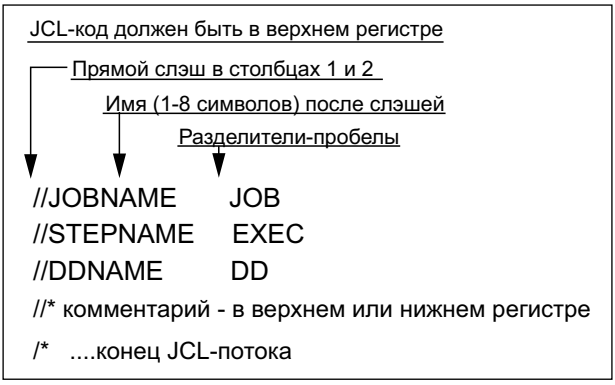


Рис. 6.1. Базовый синтаксис JCL-кода

В примере 6.1 представлен пример JCL-кода.

Пример 6.1. Пример JCL

```
//MYJOB    JOB 1
//MYSORT   EXEC PGM=SORT
//SORTIN   DD DISP=SHR,DSN=ZPROF.AREA.CODES
//SORTOUT  DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSIN    DD *
  SORT FIELDS=(1,3,CH,A)
/*
```

В главе 4, «TSO/E, ISPF и UNIX: интерактивные средства z/OS», мы выполнили эту же программу из приглашения TSO READY. Каждый JCL-оператор DD аналогичен TSO-команде ALLOCATE. Оба они используются для того, чтобы связать набор данных z/OS с DD-именем, распознаваемым программой как входные или выходные данные. Различие в методе выполнения состоит в том, что TSO выполняет сортировку в оперативном режиме, а JCL используется для выполнения сортировки в фоновом режиме.

При передаче для выполнения:

MYJOB Имя задания, которое система связывает с данной рабочей нагрузкой.

MYSORT Имя шага, сообщающее системе о необходимости выполнить программу SORT.

SORTIN В операторе DD представляет DD-имя. SORTIN в программе SORT представляет входные данные программы. Имя набора данных (DSN) для этого оператора DD: *ZPROF*.AREA.CODES. Набор данных может быть разделяемым (DISP=SHR) с другими системными процессами. Содержимое *ZPROF*.AREA.CODES представляет входные данные программы SORT.

SORTOUT Это DD-имя представляет выходные данные программы SORT.

SYSOUT SYSOUT=* определяет передачу выходных сообщений системы в область вывода на печать подсистемы JES (Job Entry Subsystem). Можно отправить выходные данные в набор данных.

SYSIN DD * – еще один оператор ввода. Он сообщает о том, что далее следуют данные или управляющие операторы. В данном случае имеет место инструкция сортировки, сообщающая программе SORT, какие поля записей данных SORTIN подлежат сортировке.

В этой книге мы используем понятие *JCL-операторы (JCL statements)*; иногда пользователи z/OS применяют старый термин *JCL-карта (JCL card)*, несмотря на то что JCL находится на диске, а не на перфокартах.

6.2 Параметры JOB, EXEC и DD

Операторы JOB, EXEC и DD имеют множество параметров, дающих возможность пользователю задавать инструкции и информацию. Описание всех этих параметров заняло бы целую книгу (такую, как публикация *z/OS JCL Reference*).

В этом разделе представлено лишь краткое описание некоторых часто используемых параметров операторов JOB, EXEC и DD.

6.2.1 Параметры JOB

Оператор JOB – JCL-оператор, идентифицирующий задание и пользователя, передавшего его на выполнение

Оператор `//MYJOB JOB 1` использует имя задания `MYJOB`. Значение `1` представляет учетное поле, которое может использоваться программами системных выходов (exits), применяемыми для выставления счетов пользователям системы.

К основным параметрам оператора JOB относятся:

REGION=	Запрашивает выделение определенных ресурсов памяти для задания.
NOTIFY=	Отправляет уведомление о выполнении задания определенному пользователю, например передавшему задание.
USER=	Определяет, что задание, выполняется с полномочиями заданного идентификатора пользователя.
TYPRUN=	Откладывает или приостанавливает выполнение задания, продолжаемого позже.
CLASS=	Направляет JCL-оператор для выполнения в заданную входную очередь.
MSGCLASS=	Направляет выходные данные задания в заданную выходную очередь.
MSGLEVEL=	Управляет количеством получаемых системных сообщений.

Пример:

```
//MYJOB JOB 1,NOTIFY=&SYSUID,REGION=6M
```

6.2.2 Параметры EXEC

Оператор EXEC – JCL-оператор, задающий имя программы, подлежащей выполнению

JCL-оператор `//MYSTEP EXEC` использует *имя шага (stepname)* `MYSTEP`. После EXEC вводится либо `PGM=`(имя исполняемой программы), либо имя JCL-процедуры. Если задана JCL-процедура, тогда в качестве параметров задаются подстановки значений переменных, требуемые JCL-процедурой. К основным параметрам оператора `EXEC PGM=` относятся:

PARM=	Параметры, известные программе и передаваемые программе.
COND=	Логический параметр, контролирующий выполнение других шагов EXEC данного задания. Существуют JCL-операторы IF, THEN, ELSE, использование которых предпочтительнее применения COND; однако в рабочих средах может существовать множество старых JCL, использующих этот оператор.
TIME=	Задает временное ограничение.

Пример:

```
//MYSTEP EXEC PGM=SORT
```

6.2.3 Параметры DD

Оператор DD – задает входные и выходные данные программы, заданной оператором EXEC

JCL-оператор `//MYDATA DD` использует DD-имя `MYDATA`. Оператор DD (Data Definition) имеет гораздо больше параметров, чем операторы JOB и EXEC. JCL-оператор DD может использоваться во многих аспектах определения или описания атрибутов входных или выходных данных программы. К основным параметрам оператора DD относятся:

DSN= Имя набора данных; может включать создание временных или новых наборов данных или ссылку на имя набора данных.

DISP= Диспозиция набора данных; определяет, нужно ли создавать набор данных, или он уже существует, и может ли набор данных разделяться между несколькими заданиями. В действительности параметр DISP= настолько важен, что мы посвятим ему следующий раздел: 6.3, «Диспозиция набора данных, параметр DISP».

SPACE= Объем дискового пространства, запрашиваемый для нового набора данных.

SYSOUT= Определяет целевое устройство печати (а также очередь вывода или набор данных).

VOL=SER= Имя тома, имя диска или имя ленты.

UNIT= Системный диск, лента, специальный тип устройства или групповое имя (локальное имя).

DEST= Направляет выходные данные в удаленную систему.

DCB= Блок управления данными; множество подпараметров.

Основные подпараметры:

- LRECL= Длина логической записи. Количество байтов/символов в каждой записи.
- RECFM= Формат записи: фиксированный, блочный, переменный и т. д.
- BLOCKSIZE= Задает размер блока для хранения записей; обычно является кратным LRECL. При значении 0 система автоматически выбирает оптимальное значение.
- DSORG= Организация набора данных: последовательная, секционированная, и т. д.

LABEL= Метка ленты (No Label или Standard Label, задается после расположения набора данных). На ленте может храниться множество наборов данных; каждый набор данных на ленте имеет определенное положение. Первый набор данных на ленте является файлом 1.

DUMMY Означает нулевой ввод или вывод в никуда данных, записываемых в это DD-имя.

* После звездочки вводятся входные данные или управляющие операторы; представляет метод передачи данных в программу из JCL-потока.

*,DLM= Все последующее представляет входные данные (даже //) до тех пор, пока в столбец 1 не будут введены два заданных алфавитно-цифровых или специальных символа.

6.3 Диспозиция набора данных, параметр DISP

Все параметры JCL важны, однако функция DISP является, возможно, наиболее важной для операторов DD. Среди различных вариантов применений параметр DISP настраивает в системе организацию очередей к наборам данных, необходимых для предотвращения конфликтов при использовании набора данных другими заданиями.

Полный параметр содержит следующие поля:

```
DISP=(status,normal end,abnormal end)
DISP=(status,normal end)
DISP=status
```

где status может принимать значения NEW, OLD, SHR или MOD.

NEW	Указывает, что требуется создать новый набор данных. Это задание во время выполнения имеет исключительный доступ к набору данных. Набор данных не должен ни уже существовать на одном томе с новым набором данных, ни находиться в системном или пользовательском каталоге (catalog).
OLD	Указывает, что набор данных уже существует и что это задание во время выполнения должно иметь исключительный доступ к нему.
SHR	Указывает, что набор данных уже существует и что несколько параллельных заданий во время выполнения могут осуществлять разделяемый доступ к нему. Все параллельные задания должны задавать значение SHR.
MOD	Указывает, что набор данных уже существует и что текущее задание во время выполнения должно иметь исключительный доступ к нему. Если текущее задание открывает набор данных для вывода, вывод будет добавлен в конец набора данных.

Шаг задания – JCL-операторы, запрашивающие и контролирующие выполнение программы и определяющие ресурсы, необходимые для выполнения программы

Параметр *normal end* указывает, что нужно делать с набором данных (*диспозиция*) при нормальном завершении текущего шага задания. Подобным образом параметр *abnormal end* указывает, что нужно сделать с набором данных при аварийном завершении.

Оба параметра имеют одинаковые опции:

DELETE	Удаление (и раскаталогизация) набора данных по завершении шага задания.
KEEP	Сохранение (без каталогизации) набора данных по завершении шага задания.
CATLG	Сохранение и каталогизация набора данных по завершении шага задания.
UNCATLG	Сохранение и раскаталогизация набора данных по завершении шага задания.
PASS	Передача функций определения конечной диспозиции последующему заданию.

По умолчанию параметры диспозиции (для нормального и аварийного завершения) заданы таким образом, чтобы оставить набор данных в том состоянии, каким он был до запуска шага задания (мы рассматривали каталоги в разделе 5.11.2, «Что такое каталог?»).

Вам может быть интересно, что произойдет, если задать DISP=NEW для уже существующего набора данных? На самом деле ничего особенного! Чтобы предотвратить неумышленное удаление файлов, z/OS отклоняет запрос DISP=NEW для существующего набора данных. Вместо нового набора данных появится сообщение об ошибке JCL.

6.3.1 Создание новых наборов данных

Если для набора данных задан DISP-параметр NEW, необходимо ввести дополнительную информацию, в частности:

- Имя набора данных.
- Тип устройства для набора данных.
- Серийный номер тома (volser), если используется диск или лента с меткой.
- Если используется диск, необходимо задать объем пространства, выделяемого для первичного экстента.
- Если набор данных является секционированным, необходимо задать размер оглавления (directory).
- Можно также указать параметры DCB. Альтернативный вариант состоит в том, чтобы эти параметры задавались программой, записывающей набор данных.

Мы уже описали параметр DISP и имена наборов данных. Опишем кратко остальные параметры.

Серийный номер тома — В операторе DD серийный номер тома имеет следующий формат: VOL=SER=xxxxxx, где xxxxxx – серийный номер тома. Параметр VOL позволяет задавать и другую информацию, поэтому используется именно такой формат.

Тип устройства — Существует множество способов задать тип устройства, однако чаще всего применяется UNIT=xxxx. Здесь xxxx может указывать тип устройства IBM (например, 3390), конкретный адрес устройства (например, 300) или групповое имя (*esoteric name*), заданное инсталляцией (например, SYSDA). Обычно SYSDA используется, чтобы сообщить системе, что следует выбрать любой доступный дисковый том из пула доступных устройств.

Имя раздела — Помните, что раздел библиотеки (или секционированного набора данных, PDS) многими приложениями и утилитами может восприниматься как набор данных. Для обращения к определенному разделу используется формат DSNAME=*ZPROF*.LIB.CNTL(TEST). Если приложение или утилита требует последовательного набора данных, то необходимо задать либо последовательный набор данных, либо раздел библиотеки. Имя библиотеки (без имени определенного раздела) можно использовать, только если программа/утилита ожидает имя библиотеки.

Space

Параметр SPACE оператора DD необходим для распределения наборов данных на DASD. Он определяет пространство, необходимое для вашего набора данных. Прежде чем создать набор данных на диске, система должна знать, сколько пространства нужно набору данных и в каких единицах следует измерять пространство.

Существует множество различных форматов и вариаций. Приведем несколько примеров.

SPACE=(TRK,10)	10 дорожек, вторичные экстенты отсутствуют
SPACE=(TRK,(10,5))	10 дорожек в первичном экстенте, 5 дорожек в каждом вторичном экстенте
SPACE=(CYL,5)	Используются цилиндры (CYL) вместо дорожек (TRK)
SPACE=(TRK,(10,5,8))	PDS с восемью блоками оглавления
SPACE=(1000,(50000,10000))	50 000 первичных записей по 1 000 байт в каждой

В простейшем случае SPACE имеет два параметра: единицу измерения и размер пространства. В качестве единицы измерения могут использоваться дорожки, цилиндры или средний размер блока (average block size)[1].

Количество пространства обычно имеет до трех подпараметров:

- Первым параметром является размер первичного экстента, выраженный в единицах измерения. Система попытается получить единый экстент (непрерывный участок пространства) с заданным количеством пространства. Если система не может получить такое количество пространства не более чем в пяти экстентах (на одном томе) перед запуском задания, происходит ошибка задания.
- Вторым параметром (если он используется) является размер каждого вторичного экстента. Система не получает это пространство перед запуском задания и не гарантирует, что это пространство доступно. Система получает вторичные экстенты динамически, во время выполнения задания. В приведенных здесь простых примерах вторичные экстенты находятся на том же томе, на котором находится и первичный экстент.
- Третий параметр (если он используется) указывает на создание секционированного набора данных (библиотеки). Он является единственным указанием на создание PDS, а не набора данных другого типа. Численное значение представляет количество блоков оглавления (по 255 байт в каждом), выделяемых для оглавления PDS (другой параметр JCL нужен для создания PDSE вместо PDS).

Если параметр SPACE содержит несколько подпараметров, весь параметр SPACE должен быть заключен в скобки.

6.4 Продолжение и сцепление

Вследствие ограничений количества символов, помещавшихся на 80-колоночных перфокартах, использовавшихся в ранних системах, в z/OS реализованы понятия продолжения (continuation) и сцепления (concatenation). Таким образом, z/OS сохранила эти соглашения с целью уменьшить неблагоприятное воздействие на предыдущие приложения и операции.

Для продолжения в синтаксисе JCL используется запятая в конце последнего полного параметра. Следующая строка JCL должна включать два слэша (//), за которыми следует по меньшей мере один пробел и затем дополнительные параметры. Синтак-

[1] В качестве единицы измерения могут применяться килобайты и мегабайты, однако они редко используются.

сис параметра JCL в строке продолжения должен начинаться с 16-го или до 16-го столбца и не должен выходить за столбец 72[1].

```
//JOBCARD JOB 1,REGION=8M,NOTIFY=ZPROF
```

Результат выполнения представленного выше JCL-оператора идентичен следующему JCL-коду с продолжениями:

```
//JOBCARD  JOB 1,
//         REGION=8M,
//         NOTIFY=ZPROF
```

Важная особенность операторов DD состоит в том, что одно DD-имя может иметь несколько операторов DD. Это называется *сцеплением (concatenation)*.

Следующий JCL-код содержит сцепленные наборы данных:

```
//DATAIN   DD DISP=OLD,DSN=MY.INPUT1
//         DD DISP=OLD,DSN=MY.INPUT2
//         DD DISP=SHR,DSN=YOUR.DATA
```

Сцепление – одно DD-имя может иметь несколько операторов DD (входных наборов данных)

Сцепление применимо только к входным наборам данных. Наборы данных автоматически обрабатываются один за другим. В этом примере, когда приложение доходит до конца набора данных `MY.INPUT1`, система автоматически открывает `MY.INPUT2` и начинает его считывать. Приложение не знает, что теперь идет считывание второго набора данных. Это продолжается до тех пор, пока не будут считаны последние данные в сцеплении; в это время приложение получает обозначение конца файла.

6.5 Почему z/OS использует символические имена файлов

z/OS обычно использует символические имена файлов (symbolic file names)[2], и это является еще одним определяющим свойством этой операционной системы. Она применяет перенаправление именования между используемым в программе именем, связанным с набором данных, и действительным именем набора данных, применяемого при выполнении этой программы. Это показано на рис. 6.2.

На рис. 6.2 показана программа на любом языке, которой требуется открыть и считать набор данных[3]. При написании программы имя XYZ выбирается произвольным образом для обращения к набору данных. Программу можно скомпилировать и сохранить как исполняемый модуль. Для выполнения исполняемой программы необходимо ввести JCL-оператор, связывающий имя XYZ с действительным именем набора данных. Этим JCL-оператором является оператор DD. Символическим именем, используемым в программе, является DDNAME, а настоящим именем набора данных является DSNAME.

[1] Столбцы 73–80 зарезервированы для так называемых порядковых номеров карт.

[2] Это относится к обычной традиционной обработке. В некоторых языках, например в C, определены интерфейсы, обходящие эту функцию.

[3] Псевдопрограмма использует термин *файл*, общий для большинства компьютерных языков.

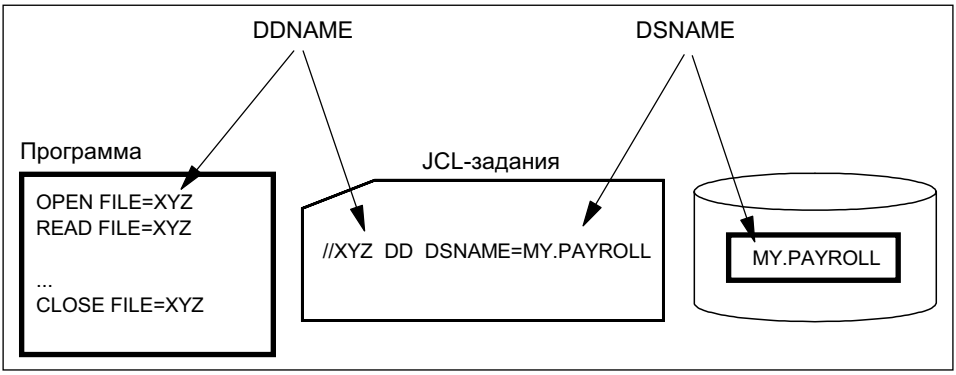


Рис. 6.2. DDNAME и DSNAME

Программу можно использовать для обработки различных входных наборов данных, просто изменив DSNAME в JCL. Это может быть полезным в больших коммерческих приложениях, использующих множество наборов данных в одном сеансе выполнения программы. Хорошим примером является программа ведения ведомостей в большой корпорации. Она может представлять собой чрезвычайно сложное приложение, которое может использовать сотни наборов данных. Одна и та же программа может применяться в различных подразделениях корпорации посредством запуска с использованием разных JCL. Подобным образом, ее можно тестировать для специальных тестовых наборов данных, используя другой набор JCL.

Символическое имя файла – перенаправление именования между используемым в программе именем, связанным с набором данных, и действительным именем набора данных, применяемого при выполнении этой программы

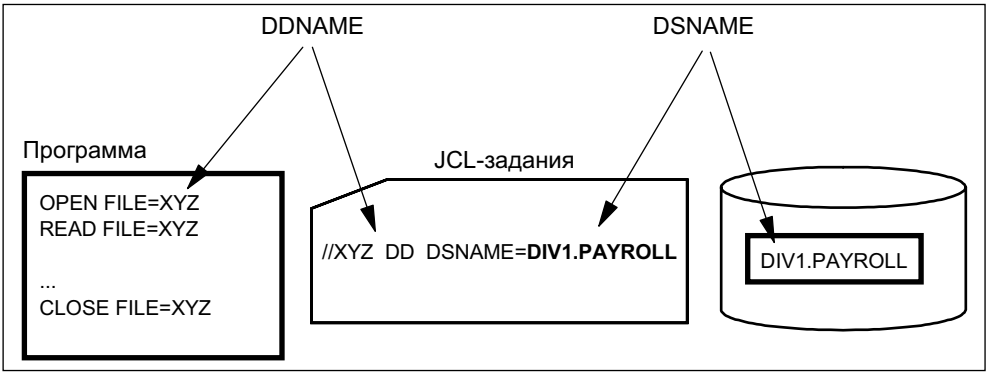


Рис. 6.3. Символическое имя файла – та же программа, но другой набор данных

Компания может использовать одну и ту же программу ведения ведомостей в различных подразделениях, изменяя только лишь один параметр в JCL-карте (`DD DSN=DIV1.PAYROLL`). При использовании значения DIV1.PAYROLL программа осуществляет доступ к набору данных для подразделения 1. Этот пример демонстрирует мощность и гибкость, обеспечиваемые использованием JCL и символических имен файлов.

Такое преобразование DDNAME--JCL--DSNAME распространяется на все традиционные задачи z/OS, хотя это может быть не всегда очевидным. Например, при использовании ISPF для редактирования набора данных ISPF создает внутренний аналог оператора DD, после чего открывает запрашиваемый набор данных с использованием оператора DD. Пользователь ISPF не видит этой обработки, она выполняется «прозрачно»[1].

6.6 Зарезервированные DD-имена

Программист может выбрать *почти* любое имя для использования в качестве DD-имени, однако рекомендуется применять осмысленное имя (длиной не больше восьми символов).

Существует несколько зарезервированных DD-имен, которые программист не может использовать (все они являются необязательными операторами DD):

```
//JOBLIB DD ...
//STEPLIB DD ...
//JOBCAT DD ...
//STEPCAT DD ...
//SYSABEND DD ...
//SYSUDUMP DD ...
//SYSMDUMP DD ...
//CEEDUMP DD ...
```

Оператор JOBLIB DD, идущий сразу за оператором JOB, определяет библиотеку, в которой следует в первую очередь выполнять поиск программ, выполняемых заданием. Оператор STEPLIB DD, идущий сразу за оператором EXEC, определяет библиотеку, в которой следует в первую очередь выполнять поиск программы, выполняемой оператором EXEC. При использовании обоих операторов STEPLIB замещает JOBLIB.

Операторы JOBCAT и STEPCAT используются для указания личных каталогов (private catalogs), но они применяются редко (последние версии z/OS не поддерживают личные каталоги). Тем не менее эти DD-имена следует считать зарезервированным именами.

Операторы SYSABEND, SYSUDUMP, SYSMDUMP и CEEDUMP используются для дампов памяти различных типов, генерируемых при аварийном завершении программы (*ABEND*).

6.7 JCL-процедуры (PROC)

Некоторые программы и задачи требуют большего количества JCL-кода, чем пользователь может быстро ввести. JCL-код для этих функций может храниться в библиотеках процедур. Раздел библиотеки процедур содержит *часть* JCL-кода рассматриваемой задачи; обычно это фиксированная, неизменяемая часть JCL-кода. Пользователь процедуры вводит переменную часть JCL-кода для конкретного задания. Другими словами, JCL-процедуры подобны макросам.

[1] Мы здесь временно игнорируем некоторые рабочие характеристики интерфейсов z/OS UNIX в z/OS; это обсуждение относится к традиционному использованию z/OS.

PROC – раздел библиотеки, содержит часть (обычно фиксированную) JCL-кода рассматриваемой задачи

Такие процедуры иногда называют *каталогизированными процедурами (cataloged procedure)*. Каталогизированные процедуры не имеют отношения к системному каталогу; это название позаимствовано из терминологии другой операционной системы.

В примере 6.2 представлен образец JCL-процедуры (PROC).

Пример 6.2. Пример JCL-процедуры

```
//MYPROC   PROC
//MYSORT   EXEC  PGM=SORT
//SORTIN   DD  DISP=SHR,DSN=&SORTDSN
//SORTOUT  DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//         PEND
```

Большая часть этого JCL-кода должна быть вам понятна. Представленные здесь JCL-функции включают:

- Операторы `PROC` и `PEND` применяются только в процедурах. Они используются для обозначения начала и конца JCL-процедуры.
- Перед оператором `PROC` вводится метка или имя; в примере 6.2 определено имя `MYPROC`.
- Причиной использования JCL-процедур является возможность подстановки переменных JCL. Единственной переменной в примере 6.2 является &SORTDSN.

В примере 6.3 представлена встраиваемая (instream) процедура из примера 6.2 в наш поток задач.

Пример 6.3. Образец встраиваемой процедуры

```
//MYJOB    JOB 1
//*---------------------------------*
//MYPROC   PROC
//MYSORT   EXEC  PGM=SORT
//SORTIN   DD  DISP=SHR,DSN=&SORTDSN
//SORTOUT  DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//         PEND
//*---------------------------------*
//STEP1    EXEC  MYPROC,SORTDSN=ZPROF.AREA.CODES
//SYSIN    DD  *
  SORT  FIELDS=(1,3,CH,A)
```

При передаче на выполнение `MYJOB` JCL-код из примера 6.2 подставляется вместо `EXEC MYPROC`. Необходимо также задать значение `&SORTDSN`.

SORTDSN и его значения помещаются в отдельную строку, являющуюся продолжением оператора EXEC. Обратите внимание на запятую после MYPROC.

//SYSIN DD *, за которым следует управляющий оператор SORT, добавляется к подставленному JCL-коду.

6.7.1 Замещение оператора JCL-процедуры

Если требуется заменить оператор JCL-процедуры целиком, можно использовать оператор замещения JCL-процедуры. Оператор замещения имеет следующий формат:

```
//stepname.ddname DD ...
```

В примере 6.4 представлено замещение DD-оператора SORTOUT в MYPROC. Здесь SORTOUT направляется в только что созданный последовательный набор данных.

Пример 6.4. Образец процедуры с замещением оператора

```
//MYJOB    JOB 1
//*---------------------------------*
//MYPROC   PROC
//MYSORT   EXEC PGM=SORT
//SORTIN   DD DISP=SHR,DSN=&SORTDSN
//SORTOUT  DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//         PEND
//*---------------------------------*
//STEP1    EXEC MYPROC,SORTDSN=ZPROF.AREA.CODES
//MYSORT.SORTOUT DD DSN=ZPROF.MYSORT.OUTPUT,
//         DISP=(NEW,CATLG),SPACE=(CYL,(1,1)),
//         UNIT=SYSDA,VOL=SER=SHARED,
//         DCB=(LRECL=20,BLKSIZE=0,RECFM=FB,DSORG=PS)
//SYSIN DD *
  SORT FIELDS=(1,3,CH,A)
```

6.7.2 Как выполняется передача задания для пакетной обработки?

Используем UNIX и AIX® в качестве аналогии. В UNIX, для того чтобы процесс обрабатывался в фоновом режиме, следует добавить амперсанд (&) в конце команды или скрипта. После этого по нажатии Enter происходит передача работы в качестве фонового процесса.

В терминологии z/OS работа (задание) передается для пакетной обработки. Пакетная обработка является приблизительным аналогом фоновой обработки в UNIX. Задание выполняется независимо от интерактивного сеанса. Обработка называется пакетной, так как она выполняется для большого количества заданий, организованных в очереди и ожидающих своей очереди для выполнения при наличии требуемых ресурсов. Команды, передающие задания, могут иметь следующий вид:

Командная строка редактора ISPF	SUBmit и нажмите Enter.
Командная оболочка ISPF	SUBmit 'USER.JCL', где набор данных является последовательным.
Командная строка ISPF	TSO SUBmit 'USER.JCL', где набор данных является последовательным.
Командная строка ISPF	TSO SUBmit 'USER.JCL(MYJOB)', где набор данных является библиотекой или секционированным набором данных, содержащим раздел `MYJOB`.
Командная строка TSO	SUBmit 'USER.JCL'.

В примере 6.5 представлено три разных места, где можно ввести команду SUBMIT.

Пример 6.5. Различные способы передачи JCL-потока на обработку

```
EDIT ---- userid.SORT.JCL ------------------------ LINE 00000000 COL 001
COMMAND ===> SUBMIT                                          SCROLL ===>
****************************** TOP OF DATA  ***************************
//userid   JOB 'accounting data',
                      .
                      .
                      .
```

TSO/E command line:

```
------------------------  TSO COMMAND PROCESSOR  ------------------------
ENTER TSO COMMAND OR CLIST BELOW:


===> SUBMIT 'userid.SORT.JCL'




ENTER SESSION MANAGER MODE ===> NO     (YES or NO)
```

After READY mode message:

```
    .
    .
    .
  READY
 SUBMIT 'userid.SORT.JCL'
```

6.8 Общее представление об SDSF

После передачи задания часто используют *System Display and Search Facility (SDSF)* для просмотра выходных данных на предмет успешного выполнения или для просмотра и исправления ошибок JCL. SDSF позволяет отображать печатные выходные данные, содержащиеся в области спула JES. Большая часть печатных выходных данных, передаваемых в JES пакетными (и другими) заданиями, в действительности никогда не распечатывается. Вместо этого выходные данные просматриваются с использованием SDSF и удаляются или каким-то образом используются.

SDSF – отображает печатные выходные данные, содержащиеся в области спула JES, для изучения

SDSF содержит множество дополнительных функций, включая:

- просмотр системного журнала и поиск текстовых фрагментов;
- ввод системных команд (в ранних версиях операционной системы только оператор мог вводить команды);
- управление обработкой заданий (задержка, освобождение, отмена и завершение заданий);
- мониторинг заданий во время их обработки;
- отображение выходных данных заданий перед их распечатыванием;
- управление порядком обработки заданий;
- управление порядком распечатывания выходных данных;
- управление принтерами и инициаторами.

На рис. 6.4 представлено главное меню SDSF.

```
 Display  Filter  View  Print  Options  Help
-------------------------------------------------------------------------------
ISFPCU41 ----------------  SDSF PRIMARY OPTION MENU  ----------------------------
COMMAND INPUT ===> _                                          SCROLL ===> PAGE

DA     Active users                    INIT  Initiators
I      Input queue                     PR    Printers
O      Output queue                    PUN   Punches
H      Held output queue               RDR   Readers
ST     Status of jobs                  LINE  Lines
                                       NODE  Nodes
LOG    System log                      SO    Spool offload
SR     System requests                 SP    Spool volumes
MAS    Members in the MAS
JC     Job classes                     ULOG  User session log
SE     Scheduling environments
RES    WLM resources
ENC    Enclaves
PS     Processes

END    Exit SDSF

Licensed Materials - Property of IBM

5694-A01 (C) Copyright IBM Corp. 1981, 2002. All rights reserved.
US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with IBM Corp.


 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=IFIND     F6=BOOK
 F7=UP        F8=DOWN      F9=SWAP      F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Рис. 6.4. Главное меню SDSF

SDSF использует иерархию оперативных панелей для направления использования пользователями функций инструмента, как показано на рис. 6.5.

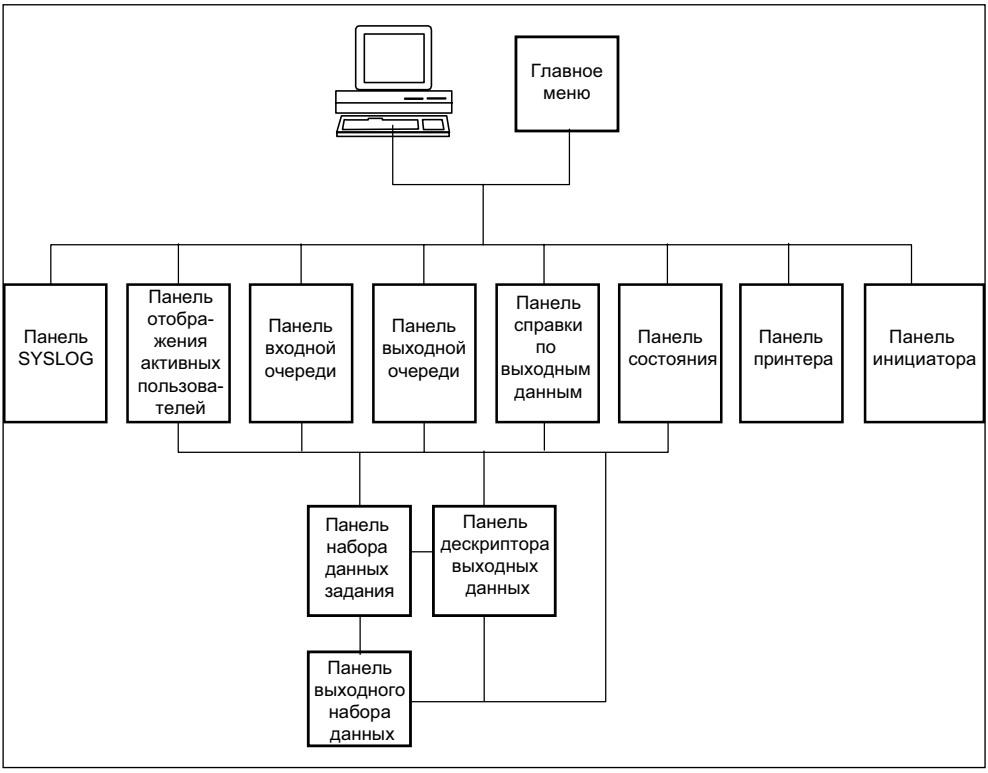


Рис. 6.5. Иерархия панелей SDSF

Можно просмотреть выходные наборы данных JES, созданные во время выполнения вашего пакетного задания. Они хранятся в наборе данных спула JES.

Можно просматривать наборы данных JES в любой из следующих очередей:

I Входная очередь.
DA Очередь выполнения.
O Выходная очередь.
H Задержанная очередь.
ST Очередь состояния.

В выходной и задержанной очередях нельзя увидеть наборы данных JES, для которых была задана автоматическая очистка путем назначения класса системного вывода MSGCLASS, который был определен таким образом, чтобы не сохранять выходные данные. Кроме того, в зависимости от MSGCLASS, выбранного в JOB-карте, вывод может быть либо в выходной очереди, либо в задержанной очереди.

Экран 1 (Screen 1) на рис. 6.6 содержит список переданных нами заданий на выполнение, а также указывает, какие выходные данные были направлены в задержанную очередь (Class T), что определяется параметром `MSGCLASS=T` в карте задания.

В нашем случае было передано и выполнено только одно задание. Таким образом, задержанная очередь содержит только одно задание. Ввод команды ? в столбце NP отображает выходные файлы, генерируемые заданием 7359.

```
Screen 1
  Display  Filter  View  Print  Options  Help
--------------------------------------------------------------------------------
SDSF HELD OUTPUT DISPLAY ALL CLASSES  LINES 44          LINE 1-1 (1)
COMMAND INPUT ===>                                        SCROLL ===> PAGE
PREFIX=*  DEST=(ALL)  OWNER=*  SYSNAME=
NP   JOBNAME  JobID    Owner    Prty C ODisp Dest                  Tot-Rec  Tot-
?_   MIRIAM2  JOB26044 MIRIAM    144 T HOLD  LOCAL                     44

Screen 2
  Display  Filter  View  Print  Options  Help
--------------------------------------------------------------------------------
SDSF JOB DATA SET DISPLAY - JOB MIRIAM2  (JOB26044)    LINE 1-3 (3)
COMMAND INPUT ===> _                                      SCROLL ===> PAGE
PREFIX=*  DEST=(ALL)  OWNER=*  SYSNAME=
NP   DDNAME   StepName ProcStep DSID Owner    C Dest              Rec-Cnt Page
     JESMSGLG JES2                2 MIRIAM   T LOCAL                  20
     JESJCL   JES2                3 MIRIAM   T LOCAL                  12
     JESYSMSG JES2                4 MIRIAM   T LOCAL                  12
```

Рис. 6.6. Просмотр выходных файлов JES2 через SDSF

Имя задания – имя, под которым задание известно в системе (JCL-оператор)

Экран 2 (Screen 2) на рис. 6.6 выводит три DD-имени: файл журнала сообщений JES2, JCL-файл JES2 и файл системных сообщений JES2. Эта опция полезна при просмотре заданий, направивших множество файлов в SYSOUT, когда требуется вывести файл, связанный с определенным шагом. Можно ввести S в столбец NP для выбора требуемого файла.

Для вывода всех файлов вместо ? в столбец NP следует ввести S; выводится журнал задания JES2, подобный представленному в примере 6.6.

Пример 6.6. Журнал задания JES2

```
J E S 2  J O B  L O G  --  S Y S T E M  S C 6 4  --  N O D E

13.19.24 JOB26044 ---- WEDNESDAY, 27 AUG 2003 ----
13.19.24 JOB26044  IRR010I  USERID MIRIAM   IS ASSIGNED TO THIS JOB.
13.19.24 JOB26044  ICH70001I MIRIAM   LAST ACCESS AT 13:18:53 ON WEDNESDAY,
AUGU
13.19.24 JOB26044  $HASP373 MIRIAM2  STARTED - INIT 1    - CLASS A - SYS SC64
13.19.24 JOB26044  IEF403I MIRIAM2 - STARTED - ASID=0027 - SC64
13.19.24 JOB26044  -                                          --TIMINGS
(MINS.)--
13.19.24 JOB26044  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP    CPU    SRB
CLOCK
13.19.24 JOB26044  -MIRIAM2           STEP1       00      9    .00    .00
.00
13.19.24 JOB26044  IEF404I MIRIAM2 - ENDED - ASID=0027 - SC64
13.19.24 JOB26044  -MIRIAM2  ENDED.  NAME-MIRIAM              TOTAL CPU TIME=
```

```
13.19.24 JOB26044  $HASP395 MIRIAM2  ENDED
------ JES2 JOB STATISTICS ------
  27 AUG 2003 JOB EXECUTION DATE
           11 CARDS READ
           44 SYSOUT PRINT RECORDS
            0 SYSOUT PUNCH RECORDS
            3 SYSOUT SPOOL KBYTES
         0.00 MINUTES EXECUTION TIME
        1 //MIRIAM2 JOB 19,MIRIAM,NOTIFY=&SYSUID,MSGCLASS=T,
          // MSGLEVEL=(1,1),CLASS=A
          IEFC653I SUBSTITUTION JCL -
19,MIRIAM,NOTIFY=MIRIAM,MSGCLASS=T,MSGLEVE
        2 //STEP1 EXEC PGM=IEFBR14
          //*-------------------------------------------------*
//* THIS IS AN EXAMPLE OF A NEW DATA SET ALLOCATION
          //*-------------------------------------------------*
        3 //NEWDD DD   DSN=MIRIAM.IEFBR14.TEST1.NEWDD,
          //           DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,
          //           SPACE=(CYL,(10,10,45)),LRECL=80,BLKSIZE=3120
        4 //SYSPRINT  DD  SYSOUT=T
          /*
ICH70001I MIRIAM   LAST ACCESS AT 13:18:53 ON WEDNESDAY, AUGUST 27, 2003
IEF236I ALLOC. FOR MIRIAM2 STEP1
IGD100I 390D ALLOCATED TO DDNAME NEWDD    DATACLAS (        )
IEF237I JES2 ALLOCATED TO SYSPRINT
IEF142I MIRIAM2 STEP1 - STEP WAS EXECUTED - COND CODE 0000
IEF285I   MIRIAM.IEFBR14.TEST1.NEWDD                   CATALOGED
IEF285I   VOL SER NOS= SBOX38.
IEF285I   MIRIAM.MIRIAM2.JOB26044.D0000101.?           SYSOUT
IEF373I STEP/STEP1   /START 2003239.1319
IEF374I STEP/STEP1   /STOP  2003239.1319 CPU    0MIN 00.00SEC SRB    0MIN
00.00S
IEF375I  JOB/MIRIAM2 /START 2003239.1319
IEF376I  JOB/MIRIAM2 /STOP  2003239.1319 CPU    0MIN 00.00SEC SRB    0MIN
00.00S
```

6.9 Utilities

z/OS содержит множество программ, полезных при пакетной обработке, называемых *утилитами (utilities)*. Эти программы выполняют множество небольших, простых и полезных функций. Базовый набор системных утилит представлен в приложении C, «Утилиты».

Утилита – программа, осуществляющая множество полезных функций пакетной обработки

Клиенты часто добавляют собственные, самостоятельно разработанные утилиты (хотя большинство пользователей не называют их утилитами), и многие из них широко применяются в сообществе пользователей. Существует также подобные (платные) продукты от независимых изготовителей программного обеспечения.

6.10 Системные библиотеки

z/OS содержит множество стандартных системных библиотек. Мы приведем краткое описание некоторых библиотек. К традиционным библиотекам относятся:

Системная библиотека – наборы данных PDS на системных дисковых томах, содержащие управляющие параметры z/OS, JCL-процедуры, базовые исполняемые модули и т. д.

- SYS1.PROCLIB. Эта библиотека содержит JCL-процедуры, распространяемые вместе с z/OS. На практике с ней может быть сцеплено множество других библиотек JCL-процедур (поставляемых с различными программными продуктами).
- SYS1.PARMLIB. Эта библиотека содержит управляющие параметры z/OS и некоторых программных продуктов. На практике с ней могут быть сцеплены другие библиотеки.
- SYS1.LINKLIB. Эта библиотека содержит многие основные исполняемые модули системы. На практике она представляет собой лишь одну из множества сцепленных исполняемых библиотек.
- SYS1.LPALIB. Эта библиотека содержит системные исполняемые модули, загружаемые в область загрузки модулей (link pack area) при инициализации системы. С ней может быть сцеплено несколько других библиотек. Хранящиеся в ней программы доступны для других адресных пространств.
- SYS1.NUCLEUS. Эта библиотека содержит базовые модули супервизора (ядра) операционной системы z/OS.
- SYS1.SVCLIB. Эта библиотека содержит подпрограммы операционной системы, называемые *вызовами супервизора (supervisor call, SVC)*.

Эти библиотеки имеют стандартный формат PDS и находятся на системных дисковых томах. Более подробно они рассматриваются в разделе 16.3.1, «Системные библиотеки z/OS».

6.11 Заключение

Простой JCL-код содержит три типа операторов: JOB, EXEC и DD. Задание может содержать несколько операторов EXEC (*шагов*), и каждый шаг может содержать несколько операторов DD. JCL содержит множество параметров и средств управления; мы рассмотрели лишь базовый набор.

Пакетное задание использует DD-имена для доступа к наборам данных. JCL-оператор DD связывает DD-имя с определенным набором данных (DS-именем) для однократного выполнения программы. Программа может осуществлять доступ к разным группам наборов данных (в различных заданиях) посредством изменения JCL для каждого задания.

Параметры DISP операторов DD позволяют не допустить нежелательного одновременного доступа к наборам данных. Это очень важно для общего функционирования системы. Параметр DISP не является средством управления безопасностью, а скорее помогает управлять целостностью наборов данных. Посредством JCL можно создавать новые наборы данных с использованием параметра DISP=NEW, указав требуемый объем пространства и требуемый том.

Пользователи системы могут составлять простой JCL-код, однако обычно для сложных заданий используются JCL-процедуры. Каталогизированная процедура создается один раз, после чего ее может использовать множество пользователей. В состав z/OS входит много JCL-процедур, и можно с легкостью добавлять собственные процедуры. Пользователь должен понимать, каким образом замещаются или расширяются операторы в JCL-процедуре, чтобы передать параметры (как правило, операторы DD), требуемые для конкретного задания.

Основные термины в этой главе		
Сцепление	Оператор DD	Оператор EXEC
Язык управления заданиями (JCL)	Оператор JOB	Шаг задания
Имя задания	PROC	SDSF
Символическое имя файла	Системная библиотека	Утилита

6.12 Контрольные вопросы

Чтобы проверить понимание материала этой главы, ответьте на следующие вопросы:

1. Где во фрагменте процедуры и в задании в разделе 6.7, «JCL-процедуры (PROC)», находится исходный код на языке COBOL? Каков возможный выходной набор данных для приложения? Каков возможный входной набор данных? Как составить JCL-код для набора данных SYSOUT для приложения?
2. Мы имеем три оператора DD:

   ```
   //DD1 DD UNIT=3480,...
   //DD2 DD UNIT=0560,...
   //DD3 DD UNIT=560,...
   ```

   Что означают эти числа? Откуда это известно?
3. JCL-код может быть передан на выполнение или запущен. В чем разница между этими понятиями?
4. Опишите связь между именем набора данных, именем DD и именем файла в программе.
5. Какой JCL-оператор (JOB, EXEC или DD) имеет больше всего параметров? Почему?
6. Какая разница между JCL и JCL PROC? Какое преимущество в использовании JCL PROC?
7. Чтобы заместить оператор JCL PROC в JCL-потоке, выполняющем PROC, какие имена PROC требуется знать? Каков порядок имен в операторе замещения JCL?
8. Когда JCL-задание имеет несколько операторов EXEC, какой тип имени связывается с каждым оператором EXEC?

6.13 Темы для дальнейшего обсуждения

Этот материал предназначен для обсуждения в классе, и такие обсуждения следует рассматривать как часть основного курса.

1. Каким образом появление систем управления базами данных могло устранить необходимость в большом количестве операторов DD?
2. Первым позиционным параметром оператора JOB является поле учетной информации. Насколько важен учет в использовании мэйнфрейма? Почему?

6.14 Упражнения

Практические упражнения в этой главе помогают развить навыки создания пакетных заданий и их передачи на выполнение в z/OS. Эти навыки необходимы для выполнения практических упражнений в остальной части книги.

Для выполнения практических упражнений вам или вашей подгруппе нужен идентификатор и пароль пользователя TSO (для этого следует обратиться к преподавателю).

Упражнения посвящены следующим темам:

- «Создание простого задания».
- «Использование ISPF в режиме разделенного экрана».
- «Управление текстом в ISPF».
- «Передача задания и просмотр результатов».
- «Создание раздела PDS».
- «Копирование раздела PDS».

6.14.1 Создание простого задания

1. В ISPF перейдите на панель Data Set List Utility и введите *yourid*.JCL в поле Dsname Level (см. предыдущие упражнения).
2. Введите e (edit) слева от *yourid*.JCL (в столбце command). Введите s (select) слева от раздела JCLTEST. Введите RESet в командной строке редактора.
3. Обратите внимание на то, что набор данных содержит только одну строку JCL: EXEC PGM=IEFBR14. Это системная утилита, не запрашивающая каких-либо входных или выходных данных и возвращающая код успешного выполнения (0). Введите SUBMIT или SUB в командную строку и нажмите Enter.
4. Введите 1 в ответ на сообщение:

   ```
   IKJ56700A ENTER JOBNAME CHARACTER(S) -
   ```

   В результате появится следующее сообщение:

   ```
   IKJ56250I JOB yourid1(JOB00037) SUBMITTED
   ```

> **Примечание.** Вывод трех звездочек (***) указывает на то, что выведены не все данные. Нажмите Enter для продолжения.

   После выполнения задания должно появиться сообщение

   ```
   $HASP165 yourid1 ENDED AT SYS1 MAXCC=0 CN(INTERNAL)
   ```

5. Добавьте (вставьте) новую первую строку в свой файл, который будет содержать оператор JOB. Оператор JOB должен предшествовать оператору EXEC. (Совет:

реплицируйте (r) один оператор EXEC, затем перезапишите оператор EXEC своим оператором JOB.) Этот оператор JOB должен иметь следующий вид:

```
//yourid A JOB 1
```

Вместо «yourid» введите свой идентификатор пользователя, оставив «A», после чего передайте на выполнение этот JCL-код и нажмите PF3, чтобы сохранить файл и выйти из редактора.

6. В главном меню ISPF найдите SDSF (описываемый в разделе 7.9.5, «Использование SDSF»). Вы можете использовать функцию разделения экрана для создания нового экранного сеанса, вследствие чего получается один сеанс DSLIST и другой сеанс для SDSF.
7. В меню SDSF введите `PREFIX yourid*`, после чего введите `ST` (Status Panel). Оба переданных задания должны быть в списке. Введите `S` (select) слева от любого из заданий, затем перейдите к предыдущей и последующей странице для просмотра сообщений, сгенерированных при выполнении. Нажмите PF3 для выхода.
8. Еще раз отредактируйте JCLTEST и вставьте в конец следующие строки:

```
//CREATE DD DSN=yourid.MYTEST,DISP=(NEW,CATLG),
// UNIT=SYSDA,SPACE=(TRK,1)
```

9. Передайте для выполнения содержимое JCLTEST, созданного выше, нажмите PF3 (сохранение и выход из редактора), после чего просмотрите выходные данные этого задания в SDSF. Обратите внимание на то, что имеется два задания с одинаковым именем. Последним запускалось задание с более высоким значением JOBID.
   а) Чему равен код завершения? Если он выше нуля, перейдите в конец списка выходных данных, чтобы просмотреть сообщение об ошибке JCL. Исправьте JCLTEST и повторно передайте задание на выполнение. Повторяйте до тех пор, пока не будет получено условие code=0000.
   б) Перейдите в панель Data Set List Utility (=3.4) и введите *yourid*.MYTEST в поле DSNAME level. Какой том использовался для хранения набора данных?
   в) Введите `DEL /` в нумерованный левый столбец (command) набора данных, чтобы удалить набор данных. Может возникнуть сообщение о подтверждении, запрашивающее подтверждение удаления набора данных.
   г) Мы увидели, что пакетное выполнение программы IEFBR14, не требующей входных или выходных данных, возвращает код завершения 0 (успешное выполнение), если не возникало ошибок JCL. Хотя IEFBR14 не осуществляет операции ввода-вывода, выполняется чтение JCL-инструкций и их выполнение системой. Эта программа полезна для создания (DISP=NEW) и удаления [DISP=(OLD,DELETE)] наборов данных в операторе DD.
10. Из любой панели ISPF введите в Command Field ==>

```
TSO SUBMIT JCL(JCLERROR)
```

Ваш идентификатор пользователя является префиксом (старшим квалификатором) набора данных JCL, содержащего раздел JCLERROR.

   а) Система запросит ввести суффикс для сгенерированной карты задания. Запомните имя и номер задания, выводимый в сообщениях передачи на выполнение.

б) В SDSF откройте выходные данные задания. Перейдите в конец. Видите ли вы ошибку JCL? Какие операнды DD являются правильными, а какие неправильными? Исправьте ошибку JCL, расположенную в *yourid*.JCL(JCLERROR). Повторно передайте на выполнение JCLERROR, чтобы подтвердить свое исправление.

11. Из любой панели ISPF введите `TSO SUBMIT JCL(SORT)`. Ваш идентификатор пользователя считается префиксом JCL набора данных, содержащего раздел SORT.
    а) Система запросит ввести суффикс для сгенерированной карты задания. Запомните имя и номер задания, выводимый в сообщениях передачи на выполнение.
    б) В SDSF введите `?` слева от имени задания. Появится отдельный листинг задания. Введите `s` (select) слева от SORTOUT, чтобы просмотреть выходные данные сортировки, после чего нажмите PF3 для возврата. Выберите JESJCL. Обратите внимание на сообщение сгенерированного оператора JOB и сообщения подстановки JCL.
12. Выполним очистку некоторых (или всех) необязательных выходных данных задания. В SDSF введите `p` (purge) слева от любого задания, которое следует очистить из выходной очереди JES.
13. Из панели ISPF введите `TSO SUBMIT JCL(SORT)` и просмотрите выходные данные.
14. Из панели ISPF введите `TSO SUBMIT JCL(SORTPROC)` и просмотрите выходные данные. Выходные данные могут не выводиться в панели SDSF ST. Это связано с тем, что имя задания не начинается с *yourid*. Чтобы просмотреть все выходные данные, введите `PRE *`, после чего введите `OWNER yourid`, чтобы просматривать только те задания, которыми вы владеете.
15. В чем разница между SORT и SORTPROC? В обоих JCL-потоках оператор SYSIN DD обращается к оператору управления сортировкой. Где находится оператор управления сортировкой?

> **Замечание.** Все JCL-обращения к &SYSUID заменяются на идентификатор пользователя, передавшего задание.

16. Откройте для редактирования раздел секционированного набора данных, содержащий управляющий оператор SORT. Замените FIELD=(1,3,CH,A) на FIELD=(6,20,CH,A). Нажмите PF3, после чего с панели ISPF введите `TSO SUBMIT JCL(SORT)`. Просмотрите выходные данные задания, используя SDSF. Отсортированы ли они по коду или по области?
17. Из панели ISPF введите `TSO LISTC ALL`. По умолчанию команда выведет все записи каталога для наборов данных, начинающихся с *yourid*. Системный каталог возвратит имена наборов данных, имя каталога, содержащего подробную информацию, расположение на томе и тип устройства, соответствующий заданным значениям операнда JCL UNIT=. LISTC является сокращением от LISTCAT.

6.14.2 Использование ISPF в режиме разделенного экрана

Как уже говорилось выше, большинство пользователей ISPF предпочитает применять разделенный экран. Для этого выполните следующие действия:

1. Переместите курсор в нижнюю (или верхнюю) строку.
2. Нажмите PF2, чтобы разделить экран.
3. Нажмите PF9 для переключения между двумя экранами.
4. Нажмите PF3 (возможно, несколько раз) для закрытия одного из разделенных экранов. Экран необязательно разделять по верхней или нижней строке. Для разделения можно выбрать любую строку, используя PF2. Можно применять больше двух экранов. Попробуйте использовать эти ISPF-команды:

```
START
SWAP LIST
SWAP <screen number.>
```

6.14.3 Управление текстом в ISPF

После подключения к TSO/E и активации ISPF перейдите на главное меню.

1. Введите каждую опцию и выпишите ее назначение и функцию. Каждая группа должна подготовить краткий обзор по одной из 12 функций панели ISPF (элементы 0–11). Обратите внимание на то, что в разных инсталляциях z/OS часто выполняется настройка панелей ISPF под свои потребности.
2. Создайте тестовый раздел в секционированном наборе данных. Введите несколько строк информации, после чего поэкспериментируйте с нижеперечисленными командами. Если нужна справка, нажмите PF1.

i	Вставка строки.
клавиша Enter	Нажмите Enter, ничего не вводя, чтобы выйти из режима вставки.
i5	Добавление пяти строк ввода.
d	Удаление строки.
d5	Удаление пяти строк.
dd/dd	Удаление блока строк (поместите один DD в первую строку блока, а другой DD в последнюю строку блока).
r	Повторение (репликация) строки.
rr/rr	Повторение (репликация) блока строк (где первое RR обозначает первую строку блока, а второе RR обозначает последнюю строку).
c, затем a или b	Копирование строки до или после другой строки.
c5, затем a или b	Копирование пяти строк до или после другой строки.
cc/cc, затем a или b	Копирование блока строк до или после другой строки.
m, m5, mm/mm	Перемещение строк.
x, x5, xx/xx	Исключение строк.
s	Повторный вывод (отображение) исключенных строк.
(	Сдвиг столбцов вправо.

)	Сдвиг столбцов влево.
<	Сдвиг данных влево.
>	Сдвиг данных вправо.

6.14.4 Передача задания на выполнение и просмотр результатов

Откройте раздел COBOL1 библиотеки *yourid*.LIB.SOURCE и просмотрите COBOL-программу. В нем отсутствует JCL-код. Теперь откройте раздел COBOL1 в *yourid*.JCL[1] . Внимательно просмотрите JCL-код. Он использует JCL-процедуру для компиляции и запуска COBOL-программы[2]. Выполните следующие действия:

1. Измените имя задания на *yourid* с дополнительными символами.
2. Измените параметр NOTIFY на ваш идентификатор пользователя.
3. Добавьте TYPRUN=SCAN в свою карту задания.
4. Введите `SUB` в командную строку ISPF для передачи задания на выполнение.
5. Разделите свой экран ISPF и перейдите в SDSF на новом экране (он мог остаться после предыдущего упражнения).
6. В SDSF перейдите в дисплей ST (Status) и найдите имя своего задания.

   Вам может потребоваться ввести команду PRE или OWNER в командную строку SDSF, чтобы просмотреть имена заданий (предыдущий пользователь мог ввести команду префикса для вывода только некоторых имен заданий).
7. Введите `S` рядом с именем своего задания, чтобы просмотреть печатные выходные данные:
   - сообщения JES2;
   - сообщения инициатора;
   - сообщения компилятора COBOL;
   - сообщения компоновщика;
   - выходные данные COBOL-программы.
8. Удалите TYPRUN=SCAN, когда вы будете готовы запустить свое задание.
9. Нажмите PF3 для перехода на уровень вверх и введите `?` рядом с именем своего задания для вывода другого формата выходных данных.

   Преподаватель может рассказать вам о назначении различных сообщений JES2 и инициатора.
   - Повторно передайте на выполнение задание с MSGLEVEL=(1,1) в операторе JOB.
   - Повторно передайте на выполнение задание с MSGLEVEL=(0,0) в операторе JOB.

   Параметр MSGLEVEL контролирует количество генерируемых сообщений инициатора.

---

[1] Совпадение имен разделов (COBOL1) необязательно, однако оно удобно.

[2] Это не совсем та COBOL-процедура, которую мы обсуждали ранее. В разных версиях операционной системы эти процедуры содержат некоторые изменения.

6.14.5 Создание раздела PDS

Существует несколько способов создать новый раздел PDS. Под своим идентификатором пользователя попробуйте каждый из приведенных ниже способов. На следующих этапах TEST3, TEST4, TEST5 и TEST6 представляют имена новых разделов. Введите несколько строк текста для каждого случая.

Используя панель редактирования ISPF:

- перейдите в основное меню ISPF;
- перейдите к опции 2 (Edit);
- в строку Data Set Name введите `JCL(TEST3)` (без кавычек!);
- введите несколько строк текста и нажмите PF3 для сохранения нового раздела.

Новый раздел можно создать при просмотре списка разделов в режиме редактирования:

- Выберите опцию 3.4 (или опцию 2) для редактирования *yourid*.JCL.
- При просмотре списка разделов введите `S TEST4` в командную строку.
- Введите несколько строк текста и нажмите `PF3`, чтобы сохранить новый раздел.

Новый раздел можно создать при редактировании существующего раздела:

- Откройте *yourid*.JCL(TEST1) или любой другой существующий раздел.
- Выберите блок строк, введя cc (в области строчных команд) в первую и последнюю строку блока.
- Введите `CREATE TEST5` в командную строку. Это создаст раздел TEST5 в текущей библиотеке.

Новый раздел можно создать через JCL. Введите следующий JCL-код в *yourid*.JCL(TEST5) или в любой другой набор данных:

```
//yourid1 JOB 1,JOE,MSGCLASS=X
//STEP1 EXEC PGM=IEBGENER
//SYSIN DD DUMMY
//SYSPRINT DD SYSOUT=*
//SYSUT2 DD DISP=OLD,DSN= yourid.JCL(TEST6)
//SYSUT1 DD *
This is some text to put in the member
More text
/*
```

Сохраните раздел, содержащий этот JCL-код. Его можно будет использовать позже.

6.14.6 Копирование раздела PDS

Существует много способов скопировать раздел библиотеки. В предыдущем упражнении использовалась функция панели ISPF 3.3 для копирования всех разделов библиотеки. Ту же функцию можно использовать для копирования одного или нескольких разделов.

Во время редактирования раздела библиотеки можно скопировать в него другой раздел библиотеки:

- откройте раздел библиотеки;
- пометьте строку этого раздела символом a (after) или b (before), чтобы указать, куда следует скопировать другой раздел;
- введите `COPY xxx` в командную строку, где xxx – имя другого раздела в текущем наборе данных.

Можно скопировать раздел из другого набора данных (или последовательного набора данных) следующим образом:

- откройте раздел или последовательный набор данных;
- пометьте строку этого раздела символом A (after) или B (before), чтобы указать, куда следует скопировать новый материал;
- введите `COPY` в командную строку, чтобы вывести панель Edit/View-Copy;
- введите полное имя последовательного набора данных (если нужно, с одинарными кавычками) или полное имя библиотеки (включая имя раздела) в поле Data Set Name.

7

Пакетная обработка и JES

Цель. Как специалисту по мэйнфреймам, вам нужно понимать способы обработки системой основных приложений компании, таких, как приложений ведения ведомостей. Такие задачи обычно выполняются посредством пакетной обработки, что предполагает выполнение одного или нескольких пакетных заданий в последовательном потоке.

Кроме того, вам необходимо понимать, каким образом подсистема ввода заданий (JES) обеспечивает пакетную обработку. JES помогает операционной системе z/OS принимать задания, назначать время их обработки и определять способ обработки выходных данных задания.

После завершения работы над этой главой вы сможете:

- приводить обзор пакетной обработки и способов инициирования и управления работой в системе;
- объяснить, каким образом JES управляет потоком работы в системе z/OS.

7.1 Что такое пакетная обработка?

Термин *пакетное задание (batch job)* появился во времена перфокарт, содержавших указания, которым должен был следовать компьютер при выполнении одной или нескольких программ. Пачки перфокарт, представлявших множество заданий, часто складывались друг на друга в магазин устройства чтения перфокарт и обрабатывались в пакетах.

Историческая справка. Герман Холлерит (Herman Hollerith, 1860–1929) изобрел перфокарту в 1890 году, работая статистиком в Бюро переписи населения (United States Census Bureau). Чтобы облегчить составление таблиц результатов переписи населения США в 1890 году, Холлерит разработал бумажную карту с 80 колонками и 12 строками, сделав ее по размеру равной долларовой купюре того времени. Для представления наборов значений данных он пробивал отверстия в карте на пересечениях соответствующих строк и столбцов. Кроме того, Холлерит разработал электромеханическое устройство для «чтения» отверстий в карте, и получающийся электрический сигнал сортировался и табулировался вычислительным устройством (позже г-н Холлерит основал компанию Computing Tabulating Recording Company, которая в конце концов и стала корпорацией IBM).

В настоящее время пакетными заданиями называются такие задания, которые могут выполняться без вмешательства пользователя или для которых можно назначить выполнение при наличии требуемых ресурсов. Например, программа, считывающая большой файл и генерирующая отчет, считается пакетным заданием.

Пакетное задание – программа, которая может выполняться с минимальным вмешательством человека, обычно выполняющаяся в назначенное время

В системах PC и UNIX нет непосредственного аналога пакетной обработке z/OS. Пакетная обработка предназначена для тех часто используемых программ, которые можно выполнять с минимальным вмешательством человека. Они обычно выполняются в назначенное время или при необходимости. Возможно, ближайшим аналогом будут процессы, запускаемые командами AT или CRON в UNIX, хотя существуют существенные различия. Пакетную обработку можно в некоторой степени считать аналогичной очереди принтера, обрабатываемой в операционных системах для процессоров Intel. Пользователи отправляют задания на печать, и задания печати ожидают обработки до тех пор, пока они не будут выбраны по приоритету из очереди заданий, называемой *спулом печати (print spool)*.

Для осуществления обработки пакетного задания специалисты в z/OS используют язык управления заданиями (JCL), чтобы сообщить z/OS, какие программы следует выполнять и какие файлы будут нужны выполняемым программам. Как уже рассматривалось в главе 6, «Использование JCL и SDSF», JCL дает возможность пользователю описывать в z/OS некоторые атрибуты пакетного задания, в частности:

- кто вы (пользователь, передавший пакетное задание);
- какую программу следует выполнить;
- где расположены входные и выходные данные;
- когда следует выполнить задание.

После передачи пользователем задания в систему пользователь обычно не осуществляет каких-либо действий, связанных с заданием, пока оно не завершится.

7.2 Что такое JES?

z/OS использует *подсистему ввода заданий (job entry subsystem, JES)* для ввода заданий в операционную систему, для назначения времени их обработки в z/OS и для управления обработкой их выходных данных. JES является компонентом операционной системы, обеспечивающим дополнительные функции управления заданиями, управления данными и управления задачами, такие, как планирование, управление потоками заданий, а также чтение и запись входных и выходных потоков на устройствах вспомогательной памяти, параллельно с выполнением заданий; этот процесс называется *спулингом (spooling)*.

JES – набор программ, обрабатывающих пакетные задания в z/OS

z/OS осуществляет управление работой в виде задач и подзадач. И транзакции, и пакетные задания связаны с внутренней очередью задач, управляемой на основе приоритетов. JES является компонентом z/OS, работающим перед выполнением программы, осуществляя подготовку работы к выполнению. JES работает также после выполнения программы, помогая выполнять очистку после выполнения работы. Это включает управление распечатыванием выходных данных, генерируемых активными программами.

Если конкретнее, JES осуществляет управление входными и выходными очередями заданий и данными.

Например, JES управляет следующими аспектами пакетной обработки в z/OS:

- получением заданий в операционной системе;
- планированием заданий для обработки в z/OS;
- управлением обработкой выходных данных заданий.

В z/OS существует две версии систем ввода заданий: JES2 и JES3. Из них JES2 является гораздо более распространенной и используется в примерах в этой книге. JES2 и JES3 содержат множество функций и возможностей, однако к их базовым функциям относятся следующие:

- прием заданий, переданных различными способами:
  - из ISPF посредством команды SUBMIT;
  - по сети;
  - из выполняющейся программы, которая может передавать другие задания через внутренний считывающий модуль (internal reader) JES;
  - с устройства чтения перфокарт (крайне редко!).
- управление очередями заданий, ожидающих выполнения. Можно определить несколько очередей для различных целей;
- управление очередями заданий для *инициатора* (*initiator*), который представляет собой системную программу, запрашивающую следующее задание в соответствующей очереди;
- прием печатных выходных данных задания во время его выполнения и управление очередью выходных данных;
- (необязательно) передача выходных данных на принтер или их сохранение в *спуле* (*spool*) для обработки в PSF, InfoPrint или другом менеджере выходных данных.

Спулинг – чтение и запись (в JES) входных и выходных потоков на устройствах вспомогательной памяти, параллельно с выполнением заданий

JES использует один или несколько дисковых наборов данных для *спулинга (spooling)*, представляющего собой процесс чтения и записи входных и выходных потоков на устройствах вспомогательной памяти, параллельно с выполнением заданий, в формате, удобном для последующей обработки или операций вывода. Слово *spool* является сокращением от *simultaneous peripheral operations online*.

JES объединяет несколько наборов данных спулов (если они существуют) в единый абстрактный набор данных. Его внутренний формат не соответствует стандартному формату метода доступа и не записывается или считывается приложениями непосредственно. Входные задания и печатные выходные данные множества заданий хранятся в едином (абстрактном) наборе данных спула. В небольшой системе z/OS наборы данных спулов могут занимать несколько сотен цилиндров дискового пространства; в больших инсталляциях они могут занимать множество полных томов дискового пространства.

Основные элементы пакетной обработки представлены на рис. 7.1.

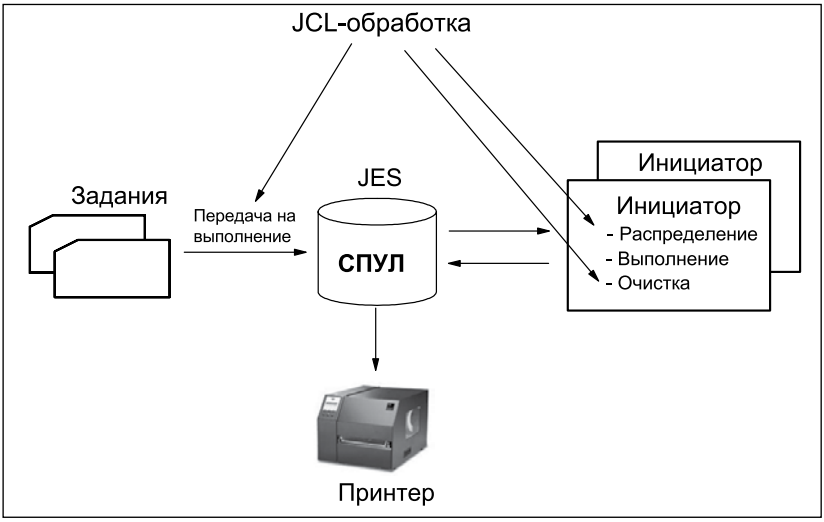


Рис. 7.1. Поток пакетной обработки

Инициатор (initiator) представляет собой компонент z/OS, осуществляющий чтение, интерпретацию и выполнение JCL-кода. Обычно он выполняется в нескольких адресных пространствах (в виде *нескольких инициаторов*). Инициатор управляет поочередным выполнением пакетных заданий в одном адресном пространстве. Если активны 10 инициаторов (в 10 адресных пространствах), то одновременно могут выполняться 10 пакетных заданий. JES частично выполняет обработку JCL, однако основная обработка JCL осуществляется инициатором.

Инициатор – компонент операционной системы, осуществляющий чтение и обработку операторов языка управления операциями, поступающих от системного устройства ввода

Задания, изображенные на рис. 7.1, представляют собой JCL и, возможно, данные, смешанные с JCL-кодом. Примером данных, которые могут быть перемешаны с JCL, является исходный код (операторы исходного кода), подаваемый на вход компилятора. Другим примером является бухгалтерское задание, готовящее недельную ведомость для разных подразделений компании (предположительно, во всех подразделениях используется одно приложение ведения ведомостей, однако входной файл и файл главного сводного отчета могут различаться).

На схеме задания представлены в виде перфокарт (с использованием традиционного символа перфокарты), хотя ввод с настоящих перфокарт в настоящее время выполняется очень редко. Обычно задание состоит из образов карт (80-байтовых записей фиксированной длины) в разделе секционированного набора данных.

7.3 Что делает инициатор?

Для асинхронного выполнения нескольких заданий система должна выполнять ряд функций:

- выбор заданий из входных очередей (этим занимается JES);
- проверку отсутствия конфликтов при использовании наборов данных несколькими заданиями (включая пользователей TSO и прочие интерактивные приложения);
- проверку правильности распределения однопользовательских устройств, в частности приводов для носителей на магнитных лентах;
- поиск исполняемых программ, запрашиваемых заданием;
- очистку после завершения задания и запрос следующего задания.

Основную часть этой работы выполняет инициатор, используя информацию JCL для каждого задания. Самой сложной функцией является проверка отсутствия конфликтов при использовании наборов данных. Например, если два задания попытаются одновременно выполнить запись в один набор данных (или если одно задание осуществляет чтение, а другое – запись), возникает конфликт[1]. Такие конфликты обычно вызывают повреждение данных. Основное назначение JCL состоит в том, чтобы сообщить инициатору, что нужно заданию.

Предотвращение конфликтов при использовании наборов данных крайне важно для z/OS и является одним из определяющих свойств этой операционной системы. При правильном построении JCL-кода предотвращение конфликтов выполняется автоматически. Например, если задание A и задание B должны выполнить запись в определенный набор данных, система (через инициатор) не позволяет обоим заданиям выполняться одновременно. Вместо этого, после того как запущено первое задание, инициатор, пытающийся запустить другое задание, переводится в режим ожидания до тех пор, пока не завершится первое задание.

[1] Существуют случаи, когда такое использование является корректным, и можно разработать JCL-код для таких случаев. В случае простых пакетных заданий такие конфликты обычно неприемлемы.

7.4 Управление заданиями и выходными данными с использованием JES и инициаторов

Рассмотрим, каким образом JES и инициаторы z/OS совместно осуществляют обработку пакетных заданий, используя два сценария.

7.4.1 Сценарий пакетного задания 1

Представьте, что вы являетесь программистом приложений для z/OS, разрабатывающим программу для неопытных пользователей. Ваша программа должна осуществлять чтение нескольких файлов, выполнять запись в другие несколько файлов и генерировать печатный отчет. В z/OS такая программа будет выполняться как пакетное задание.

Какого рода функции нужны операционной системе для выполнения требований вашей программы? И каким образом ваша программа будет осуществлять доступ к этим функциям?

Прежде всего вам потребуется специальный язык, чтобы сообщить операционной системе о ваших потребностях. В z/OS таким языком является JCL (Job Control Language). Использование JCL подробно рассматривается в главе 6, «Использование JCL и SDSF», но на данном этапе можем предположить, что JCL содержит средства запрашивания ресурсов и служб операционной системы для пакетного задания.

Спецификации и запросы, которые могут выполняться для пакетного задания, включают функции, требуемые для компиляции и выполнения программы и для выделения памяти для программы во время ее выполнения.

Используя JCL, можно задать следующую информацию:

- Кто вы (важно в целях безопасности).
- Какие ресурсы (программы, файлы, память) и службы системы нужны для обработки вашей программы. Вам, например, может потребоваться выполнить следующие действия:
    - загрузить код компилятора в память;
    - сделать исходный код доступным для компилятора; другими словами, когда компилятор запрашивает чтение, операторы исходного кода передаются в память компилятора;
    - выделить некоторый объем памяти для кода компилятора, буферов ввода-вывода и рабочих областей;
    - сделать выходной дисковый набор данных, принимающий объектный код, обычно называемый *object deck* («объектная колода») или просто *OBJ*, доступным для компилятора;
    - сделать файл печати, содержащий информацию об ошибках, доступным для компилятора;
    - при необходимости инициировать загрузку операционной системой z/OS нового объектного модуля в память (этот шаг нужно пропустить при отказе компиляции);

- выделить определенный объем памяти для использования программой;
- сделать все входные и выходные файлы доступными для программы;
- сделать принтер для вывода сообщений доступным для программы.

В свою очередь, операционная система должна выполнить следующие действия:

- преобразовать JCL в управляющие блоки, описывающие требуемые ресурсы;
- выделить требуемые ресурсы (программы, память, файлы);
- запланировать своевременное выполнение программы, например чтобы программа запускалась только при успешном выполнении компиляции;
- освободить ресурсы после завершения выполнения программы.

Компонентами z/OS, выполняющими эти задачи, являются JES и программа инициатора пакетной обработки.

JES можно рассматривать как диспетчер заданий, ожидающих в очереди. Этот компонент управляет приоритетом набора заданий и соответствующими входными данными и выходными результатами. Инициатор использует операторы в JCL-картах для указания ресурсов, требуемых для каждого задания, после их запуска (диспетчеризации) системой JES.

Описанный JCL-код называется заданием, в данном случае состоящем из двух последовательных шагов – компиляции и выполнения. Шаги задания всегда выполняются последовательно. Для выполнения задания оно должно быть передано на выполнение в JES. Чтобы упростить задачу, z/OS содержит набор процедур в наборе данных под названием SYS1.PROCLIB. Процедура представляет собой набор JCL-операторов, готовых к выполнению.

Процедура – набор JCL-операторов

В примере 7.1 представлена JCL-процедура, которая может осуществлять компиляцию, компоновку и выполнение программы (более подробно эти шаги описываются в главе 8, «Проектирование и разработка приложений для z/OS»). Первый шаг вызывает компилятор языка COBOL, заданный в `//COBOL EXEC PGM=IGYCRCTL`. Оператор `//SYSLIN DD` описывает выходные данные компилятора (object deck).

Объектный модуль представляет собой входные данные для второго шага, который выполняет компоновку (посредством программы IEWL). Компоновка нужна для разрешения внешних ссылок и для *импорта* или *связывания* ранее разработанных общих подпрограмм (своего рода повторного использования кода).

На третьем шаге происходит выполнение программы.

Пример 7.1. Процедура компиляции, компоновки и выполнения программ

```
000010 //IGYWCLG PROC   LNGPRFX='IGY.V3R2M0',SYSLBLK=3200,
000020 //               LIBPRFX='CEE',GOPGM=GO
000030 //*
000040 //*************************************************
000050 //* *
000060 //* Enterprise COBOL for z/OS and OS/390 *
```

```
000070 //*       Version 3 Release 2 Modification 0 *
000080 //* *
000090 //* LICENSED MATERIALS - PROPERTY OF IBM. *
000100 //* *
000110 //* 5655-G53 5648-A25 (C) COPYRIGHT IBM CORP. 1991, 2002 *
000120 //* ALL RIGHTS RESERVED *
000130 //* *
000140 //* US GOVERNMENT USERS RESTRICTED RIGHTS - USE, *
000150 //* DUPLICATION OR DISCLOSURE RESTRICTED BY GSA *
000160 //* ADP SCHEDULE CONTRACT WITH IBM CORP. *
000170 //* *
000180 //*****************************************************
000190 //*
000300 //COBOL EXEC PGM=IGYCRCTL,REGION=2048K
000310 //STEPLIB  DD DSNAME=&LNGPRFX..SIGYCOMP,
000320 //            DISP=SHR
000330 //SYSPRINT DD SYSOUT=*
000340 //SYSLIN   DD DSNAME=&&LOADSET,UNIT=SYSDA,
000350 //            DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
000360 //            DCB=(BLKSIZE=&SYSLBLK)
000370 //SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
000440 //LKED EXEC PGM=HEWL,COND=(8,LT,COBOL),REGION=1024K
000450 //SYSLIB   DD DSNAME=&LIBPRFX..SCEELKED,
000460 //            DISP=SHR
000470 //SYSPRINT DD SYSOUT=*
000480 //SYSLIN   DD DSNAME=&&LOADSET,DISP=(OLD,DELETE)
000490 //         DD DDNAME=SYSIN
000500 //SYSLMOD  DD DSNAME=&&GOSET(&GOPGM),SPACE=(TRK,(10,10,1)),
000510 //            UNIT=SYSDA,DISP=(MOD,PASS)
000520 //SYSUT1   DD UNIT=SYSDA,SPACE=(TRK,(10,10))
000530 //GO EXEC PGM=*.LKED.SYSLMOD,COND=((8,LT,COBOL),(4,LT,LKED)),
000540 //            REGION=2048K
000550 //STEPLIB  DD DSNAME=&LIBPRFX..SCEERUN,
000560 //            DISP=SHR
000570 //SYSPRINT DD SYSOUT=*
000580 //CEEDUMP  DD SYSOUT=*
000590 //SYSUDUMP DD SYSOUT=*
```

Для вызова процедуры можно написать простой JCL-код, подобный представленному в примере 7.2. В этом примере были добавлены другие операторы DD, в частности

```
//COBOL.SYSIN DD *
```

Он содержит исходный код на языке COBOL.

Пример 7.2. COBOL-программа

```
000001 //COBOL1 JOB (POK,999),MGELINSKI,MSGLEVEL=(1,1),MSGCLASS=X,
000002 // CLASS=A,NOTIFY=&SYSUID
000003 /*JOBPARM SYSAFF=*
000004 // JCLLIB ORDER=(IGY.SIGYPROC)
000005 //*
000006 //RUNIVP EXEC IGYWCLG,PARM.COBOL=RENT,REGION=1400K,
000007 //              PARM.LKED='LIST,XREF,LET,MAP'
000008 //COBOL.STEPLIB DD DSN=IGY.SIGYCOMP,
000009 //              DISP=SHR
000010 //COBOL.SYSIN DD *
000011     IDENTIFICATION DIVISION.
000012     PROGRAM-ID. CALLIVP1.
000013     AUTHOR. STUDENT PROGRAMMER.
000014     INSTALLATION. MY UNIVERSITY
000015     DATE-WRITTEN. JUL 27, 2004.
000016     DATE-COMPILED.
000017     /
000018     ENVIRONMENT DIVISION.
000019     CONFIGURATION SECTION.
000020     SOURCE-COMPUTER. IBM-390.
000021     OBJECT-COMPUTER. IBM-390.
000022
000023     PROCEDURE DIVISION.
000024        DISPLAY «***** HELLO WORLD *****» UPON CONSOLE.
000025        STOP RUN.
000026
000027 //GO.SYSOUT DD SYSOUT=*
000028 //
```

При выполнении этого шага программа управляется операционной системой z/OS, а не JES (рис. 7.2). Кроме того, на данном этапе процесса нужна функция спулинга.

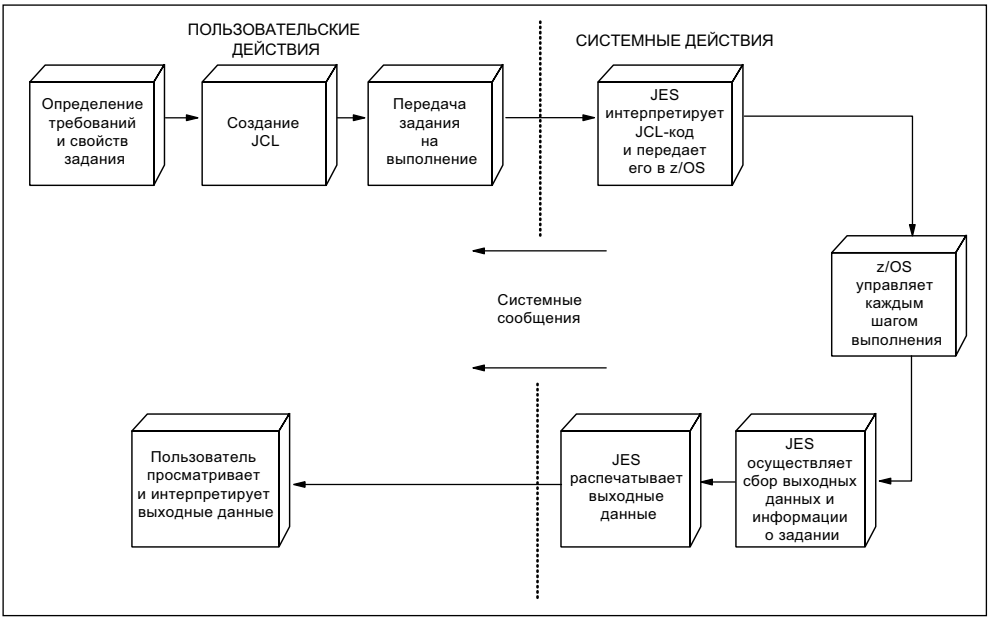


Рис. 7.2. Действия, связанные с JCL

Спулинг является средством, применяя которое система управляет своей работой, включая:

- использование пространства на *устройствах хранения с прямым доступом (direct access storage devices, DASD)* в качестве буферной памяти для уменьшения задержек в обработке при передаче данных между периферийным оборудованием и запускаемыми программами;
- чтение и запись входных и выходных потоков на промежуточное устройство для последующей обработки или вывода;
- выполнение операций, например связанных с печатью, в то время, когда компьютер занят выполнением другой работы.

Существует два вида спулинга: входной и выходной. Оба они повышают производительность программы, выполняющей чтение входных данных и запись выходных данных.

Для реализации входного спулинга в JCL следует объявить оператор `// DD *`, определяющий один файл, записи содержимого которого написаны в JCL между оператором `// DD *` и оператором `/*`. Все логические записи должны иметь 80 символов. В этом случае данный файл считывается и хранится в определенной области спула JES2 (огромном JES-файле на диске), как показано на рис. 7.3.

Позже, когда программа выполняется и запрашивает чтение этих данных, JES2 берет записи из спула и передает их программе (с дисковой скоростью передачи).

Чтобы реализовать выходной спулинг в JCL, следует ввести ключевое слово `SYSOUT` в оператор DD. `SYSOUT` определяет пустой файл в спуле, распределяемый

с логическими записями длиной 132 символа в печатном формате (EBCDIC/ASCII/UNICODE). Этот файл распределяется JES при интерпретации DD-карты с ключевым словом `SYSOUT` и используется позже в программе шага. Как правило, по завершении задания этот файл распечатывается JES-функцией.

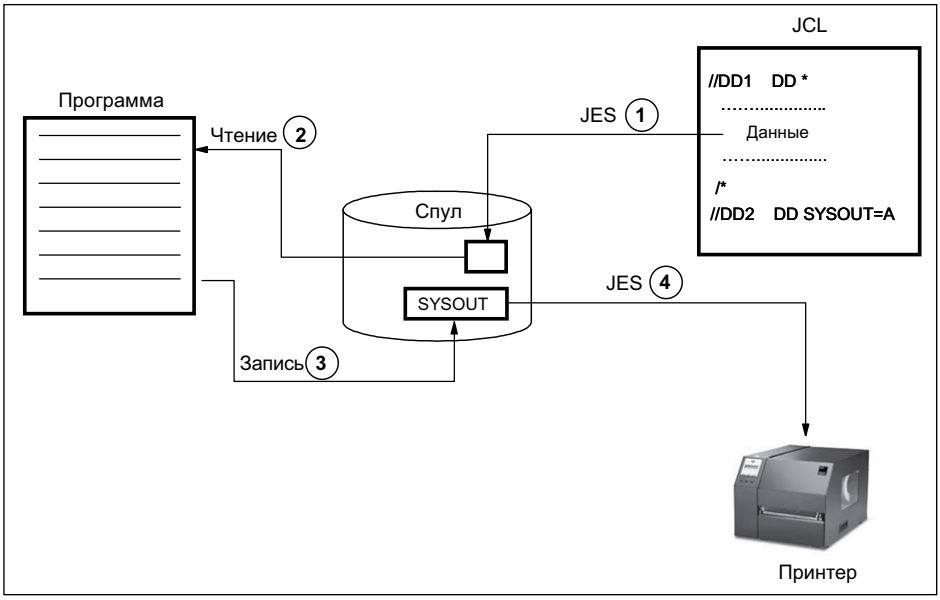


Рис. 7.3. Спулинг

7.4.2 Сценарий пакетного задания 2

Теперь предположим, что требуется создать резервную копию одного главного файла, после чего обновить главный файл путем вставки записей, считываемых из другого файла (*файла обновления*). В этом случае необходимо задание, содержащее два шага. На шаге 1 задание осуществляет чтение главного файла и записывает его на магнитную ленту. На шаге 2 запускается другая программа (которая может быть написана на языке COBOL), осуществляющая чтение записи из файла обновления и поиск ее соответствия в главном файле. Программа обновляет существующую запись (если она находит ее соответствие) или добавляет новую запись.

Какие функции при таком сценарии нужны в операционной системе для соответствия вашим требованиям?

Следует создать задание, состоящее из двух шагов, определяющих следующее:

- кто вы;
- какие ресурсы нужны заданию, в частности может потребоваться:
  - загрузить программу резервного копирования (только что скомпилированную);
  - определить, сколько памяти системе нужно выделить для распределения программы резервного копирования, буферов ввода-вывода и рабочих областей;

- сделать выходной набор данных на магнитной ленте, предназначенный для записи резервной копии, копии и главного набора данных доступным для программы резервного копирования;
- в конце работы программы сообщить операционной системе, что теперь программу обновления следует загрузить в память (однако этого не следует делать в случае отказа программы резервного копирования);
- сделать файл обновления и главный файл доступным для программы обновления;
- сделать принтер для вывода сообщений доступным для программы.

Ваш JCL-код должен содержать два шага, первый из которых указывает ресурсы для программы резервного копирования, а второй – для программы обновления (рис. 7.4).

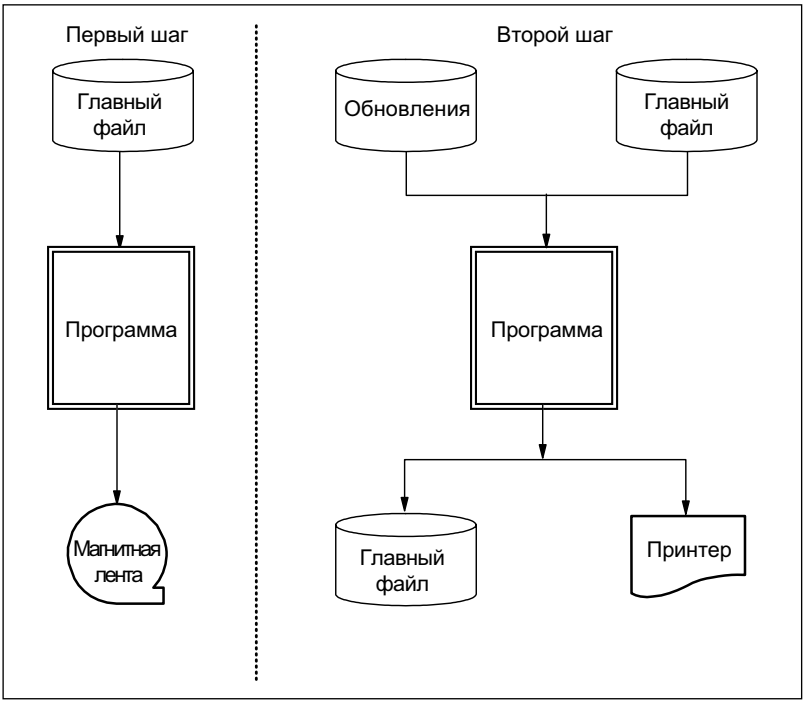


Рис. 7.4. Сценарий 2

Логически второй шаг не будет выполняться, если на первом шаге по какой-то причине произойдет отказ. Второй шаг содержит оператор `// DD SYSOUT`, указывающий на необходимость спулинга выходных данных.

Задания могут запускаться при достаточном количестве ресурсов. В этом случае система становится более эффективной: JES осуществляет управление заданиями до и после выполнения программы; основная управляющая программа управляет заданиями в процессе обработки.

z/OS содержит два типа подсистем ввода заданий: JES2 и JES3. В этом разделе обсуждается JES2. Краткое сравнение JES2 и JES3 приведено в разделе 7.6, «Сравнение JES2 и JES3».

7.5 Поток заданий в системе

Давайте подробнее рассмотрим, каким образом обрабатывается задание с использованием сочетания JES и программы инициатора пакетной обработки.

За время существования задания JES2 и базовая управляющая программа z/OS управляют различными фазами общей обработки. Очереди заданий содержат задания, ожидающие запуска, уже выполняющиеся, ожидающие генерирования выходных данных, уже имеющие сгенерированные выходные данные и ожидающие очистки из системы.

В сущности, задание проходит следующие фазы:

- ввод;
- преобразование;
- обработка;
- вывод;
- печать/перфорирование (создание твердой копии);
- завершение (очистка).

Во время обработки пакетных заданий существует множество контрольных точек (*checkpoints*). Контрольной точкой называется точка процесса выполнения, в которой может быть записана информация о состоянии задания и системы (в файл, называемый набором данных контрольной точки). Контрольные точки позволяют перезапустить шаг задания позже в случае его аварийного завершения вследствие ошибки.

Контрольная точка – точка, в которой может быть записана информация о состоянии задания и системе, что позволяет перезапустить шаг задания позже

На рис. 7.5 представлены следующие фазы задания, выполняемые при пакетной обработке.

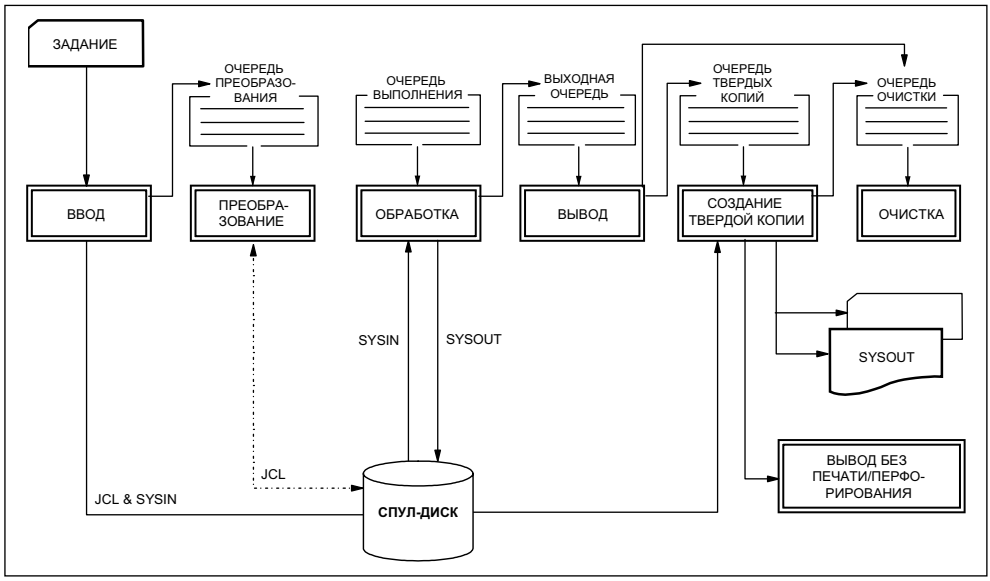


Рис. 7.5. Поток заданий в системе

1. Фаза ввода.

   JES2 принимает задания в виде входного потока от устройств ввода, от других программ посредством внутренних считывающих модулей или от других узлов в сети ввода заданий.

   Внутренний считывающий модуль представляет собой программу, которую другие программы могут использовать для передачи заданий, управляющих операторов и команд в JES2. Любое задание, запущенное в z/OS, может использовать внутренний считывающий модуль для передачи входного потока в JES2. JES2 может принимать несколько заданий одновременно через несколько внутренних считывающих модулей.

   Системный программист определяет внутренние считывающие модули, используемые для обработки всех пакетных заданий, отличных от запускаемых задач *(started tasks, STC)* и TSO-запросов.

   JES2 считывает входной поток и назначает идентификатор задания каждому JCL-оператору JOB. JES2 записывает JCL-код задания, необязательные управляющие операторы JES2 и данные SYSIN в наборы данных DASD, называемые наборами данных спула (spool data sets). Затем JES2 выбирает из наборов данных спула задания для обработки и последующего выполнения.

2. Фаза преобразования.

   JES2 использует программу-преобразователь для анализа JCL-операторов задания. Преобразователь осуществляет объединение JCL-кода задания с JCL-кодом библиотеки процедур. Библиотека процедур может быть определена JCL-оператором JCLLIB, а системные/пользовательские библиотеки процедур могут быть определены оператором PROCxx DD процедуры запуска JES2. Затем JES2 преобразует смешанный JCL-код в текст преобразователя/интерпретатора, распознаваемый как JES2, так и инициатором. Затем JES2 сохраняет текст преобразователя/интерпретатора в наборе данных спула. Если JES2 обнаружит какие-либо ошибки JCL, он выдает сообщения, и задание помещается в очередь обработки вывода, а не в очередь выполнения. Если ошибок нет, JES2 помещает задание в очередь выполнения.

3. Фаза обработки.

   На фазе обработки JES2 отвечает на запросы заданий от инициаторов. JES2 выбирает задания, ожидающие выполнения из очереди заданий и передает их инициаторам.

   Инициатор представляет собой системную программу, принадлежащую к z/OS, но управляемую JES или компонентом управления рабочей нагрузкой (WLM), который запускает задание, выделяя требуемые ресурсы, позволяя ему соревноваться с уже запущенными заданиями (WLM обсуждается в разделе 3.5, «Что такое управление рабочей нагрузкой?»).

   JES2-инициаторы запускаются оператором или подсистемой JES2 автоматически при инициализации системы. Они определены в JES2 посредством операторов инициализации JES2. Инсталляция сопоставляет каждый инициатор с одним или несколькими классами заданий, чтобы обеспечить эффективное использование доступных системных ресурсов. Инициаторы выбирают задания, классы которых соответствуют классу, назначенному инициатору, учитывая приоритет заданий в очереди.

WLM-инициаторы запускаются системой автоматически исходя из целей производительности, относительной важности пакетной рабочей нагрузки и способности системы выполнять больше работы. Инициаторы выбирают задания, исходя из их класса обслуживания и порядка их доступности для выполнения. Задания направляются на WLM-инициаторы JES2-оператором инициализации JOBCLASS.

4. Фаза вывода.

   JES2 контролирует всей обработкой SYSOUT. SYSOUT представляет вывод, генерируемый системой, т. е. все выходные данные, генерируемые заданием или для задания. Эти выходные данные включают системные сообщения, требующие распечатки, а также наборы данных, запрашиваемые пользователем, для которых требуется выполнить распечатку или перфорирование. После выполнения задания JES2 анализирует свойства выходных данных задания с точки зрения их класса вывода и требований настройки устройств; затем JES2 группирует наборы данных с похожими свойствами. JES2 помещает выходные данные в очередь для распечатки или перфорирования.

   SYSOUT – определяет целевой объект вывода выходных данных, генерируемых заданием и системой

5. Фаза создания твердой копии.

   JES2 выбирает выходные данные для обработки из выходных очередей по классу вывода, коду маршрута, приоритету и другим критериям. Выходная очередь может содержать данные вывода, подлежащие локальной или удаленной обработке. После обработки всех выходных данных определенного задания JES2 помещает задание в очередь очистки.

6. Фаза очистки.

   После завершения обработки задания JES2 освобождает пространство спула, выделенное для задания, что делает это пространство доступным для распределения последующим заданиям. Затем JES2 выдает сообщение оператору, указывая, что задание было очищено из системы.

   Очистка – освобождение пространства спула, выделенного для задания, после обработки задания

7.6 Сравнение JES2 и JES3

Как говорилось выше, IBM предлагает два вида подсистем ввода заданий: JES2 и JES3. Во многих случаях JES2 и JES3 выполняют похожие функции: они считывают задания в систему, преобразуют их во внутренний машинный формат, выбирают задания для выполнения, обрабатывают их выходные данные и очищают их из системы.

В мэйнфрейм-инсталляции, имеющий только один процессор, JES3 обеспечивает настройку работы с носителями на магнитной ленте, управление зависимыми заданиями и планирование заданий по сроку завершения, тогда как JES2 в такой же системе потребовал бы, чтобы пользователи осуществляли управление этими операциями другими средствами. В инсталляции с многопроцессорной конфигурацией существуют заметные различия между этими двумя версиями, главным образом состоящие в способе осуществления JES2 независимого управления своими функциями об-

работки заданий. Другими словами, в этой конфигурации каждый JES2-процессор управляет собственным вводом заданий, планированием заданий и обработкой вывода заданий. В большинстве инсталляций используется JES2, равно как и в примерах, приведенных в этой книге.

На рис. 7.6 перечислены некоторые различия между JES2 и JES3.

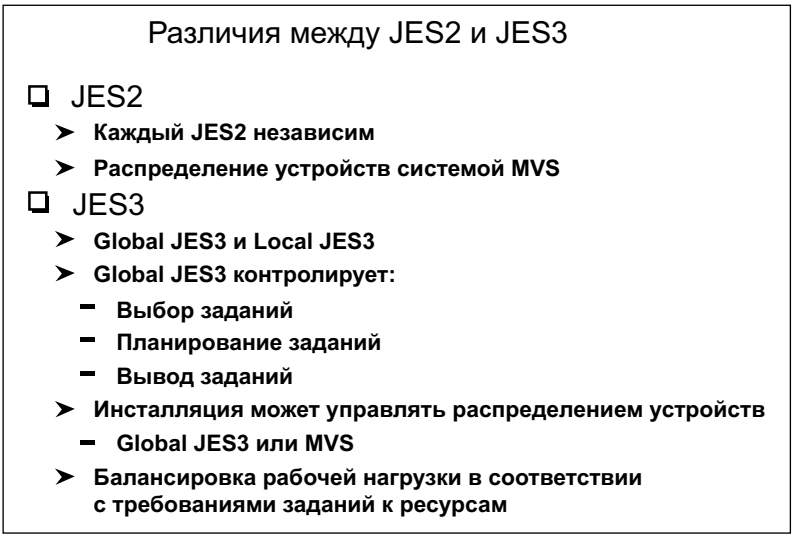


Рис. 7.6. Различия между JES2 и JES3

В случаях кластеризации нескольких систем z/OS (в *сисплексе*) можно настроить JES2 на совместное использование спула и наборов данных контрольных точек с другими системами JES2 в том же сисплексе. Такая конфигурация называется Multi-Access Spool (MAS). С другой стороны, JES3 осуществляет централизованное управление своими функциями обработки с использованием одного глобального процессора JES3. Этот глобальный процессор осуществляет все функции выбора заданий, планирования и распределения устройств для всех остальных систем JES3.

7.7 Заключение

Пакетная обработка является наиболее фундаментальной функцией z/OS. Многие пакетные задания выполняются параллельно, и для управления выполнением каждого задания используется JCL. Правильное применение параметров JCL (особенно параметра DISP операторов DD) позволяет осуществлять параллельное, асинхронное выполнение заданий, которым может требоваться доступ к одним и тем же наборам данных.

Инициатор представляет собой системную программу, обрабатывающую JCL, устанавливающую требуемую среду в адресном пространстве и запускающую пакетные задания в том же адресном пространстве. Использование нескольких инициаторов (каждый в отдельном адресном пространстве) позволяет осуществлять параллельное выполнение пакетных заданий.

Целью операционной системы является обработка заданий при наиболее оптимальном использовании системных ресурсов. Для достижения этой цели необходимо осуществлять управление ресурсами для выполнения следующих действий:

- перед обработкой задания – зарезервировать входные и выходные ресурсы для заданий;
- во время обработки задания – осуществлять управление данными SYSIN и SYSOUT в спуле;
- после обработки задания – освободить все ресурсы, использовавшиеся завершенными заданиями, что делает ресурсы доступными для других заданий.

z/OS осуществляет управление заданиями и ресурсами совместно с JES. JES получает задания, передаваемые в систему, планирует их обработку в z/OS и управляет обработкой их выходных данных. JES представляет собой диспетчер заданий, ожидающих в очереди. Он осуществляет управление приоритетом заданий и соответствующими входными данными и выводимыми результатами. Инициатор использует операторы в JCL-записях для указания ресурсов, требуемых для каждого задания, после их запуска (диспетчеризации) системой JES.

IBM предлагает два вида подсистем ввода заданий: JES2 и JES3. Во многих случаях JES2 и JES3 выполняют похожие функции.

За время существования задания JES и базовая управляющая программа z/OS управляют различными фазами общей обработки. Управление заданиями осуществляется в очередях: задания, ожидающие запуска (очередь преобразования), уже выполняющиеся (очередь выполнения), ожидающие генерирования выходных данных (выходная очередь), уже имеющие сгенерированные выходные данные (очередь твердых копий) и ожидающие очистки из системы (очередь очистки).

Основные термины в этой главе		
Пакетное задание	Контрольная точка	Инициатор
Подсистема ввода заданий (JES)	Процедура	Очистка
Спулинг	SYSIN	SYSOUT

7.8 Контрольные вопросы

Чтобы проверить понимание материала этой главы, ответьте на следующие вопросы:

1. Что такое пакетная обработка?
2. Зачем в операционной системе z/OS нужна JES?
3. Какие типы обработки обычно выполняет JES2 за время существования задания?
4. Что означает сокращение *spool*?
5. Какие действия выполняет инициатор?

7.9 Упражнения

Упражнения посвящены следующим темам:

- «Изучение системных томов».

- «Использование утилиты в задании».
- «Изучение JCL для подключения к TSO».
- «Изучение главного каталога».
- «Использование SDSF».
- «Использование TSO REXX и ISPF».

7.9.1 Изучение системных томов

Используя функции ISPF, изучите несколько системных томов. В частности, интересует следующее:

- Изучите именование наборов данных VSAM. Обратите внимание на использование слов DATA и INDEX в качестве последнего квалификатора.
- Найдите область спула. Ее можно найти по имени набора данных. Каков ее размер?
- Найдите основные системные библиотеки, в частности SYS1.PROCLIB и прочие. Обратите внимание на имена разделов.
- Обратите внимание на поле статистики ISPF, отображаемое в списке разделов. В чем его различие для исходных библиотек и библиотек выполнения?

7.9.2 Использование утилиты в задании

z/OS содержит утилиту под названием IEBGENER, предназначенную для копирования данных. Она использует четыре оператора DD:

- SYSIN – для управляющих операторов. Можно использовать DD DUMMY для этого оператора, так как у нас нет каких-либо управляющих операторов для этого задания.
- SYSPRINT – для сообщений программы. В этом упражнении следует использовать SYSOUT=X.
- SYSUT1 – для входных данных.
- SYSUT2 – для выходных данных.

Основная функция программы состоит в том, чтобы скопировать набор данных, заданный оператором SYSUT1, в набор данных, заданный оператором SYSUT2. Оба набора данных должны быть последовательными наборами данных или разделами библиотеки.

Программа автоматически получает атрибуты блока управления данными (DCB) из входного набора данных и применяет их к выходному набору данных. Напишите JCL-код задания, чтобы вывести раздел *yourid*.JCL(TEST1) в SYSOUT=X.

7.9.3 Изучение JCL для подключения к TSO

Панель ввода пароля при подключении к TSO содержит имя JCL-процедуры, используемой для создания TSO-сеанса. Существует несколько процедур с различными свойствами.

Просмотрите процедуру ISPFPROC. Преподаватель может помочь найти правильную библиотеку для ISPFPROC.

- Каково имя основной выполняемой TSO-программы?
- Почему используется так много операторов DD? Обратите внимание на сцепление наборов.

Найдите процедуру IKJACCNT. Она представляет минимальный вариант процедуры подключения к TSO.

7.9.4 Изучение главного каталога

Перейдите к опции ISPF 6 и выполните следующие действия:

- Используйте команду LISTC LEVEL(SYS1) для вывода простого списка всех наборов данных SYS1 из главного каталога.
- Обратите внимание на то, что они имеют тип NONVASM или CLUSTER (и связанные записи тип DATA и INDEX). CLUSTER соответствует наборам данных VSAM.
- Нажмите клавишу PA1 для завершения вывода списка (для справки см. раздел 3.3.3, «Использование клавиши PA1»).
- Для вывода расширенного списка используйте команду LISTC LEVEL(SYS1) ALL.

  Обратите внимание на серийный номер тома и тип устройства, соответствующие наборам данных NONVSAM. Они представляют базовую информацию в каталоге.
- Используйте LISTC LEVEL(xxx) для просмотра одного из уровней ALIAS, и обратите внимание на то, что данные извлекаются из пользовательского каталога.

> **Примечание.** Ввод команды profile с NOPREFIX приводит к генерации списка на уровне всей системы при вводе команд LISTC и LISTC ALL. Эти команды позволяют просматривать все записи из главного каталога, включая записи ALIAS.

7.9.5 Использование SDSF

В главном меню ISPF найдите и выберите System Display and Search Facility (SDSF). Эта утилита позволяет отображать выходные наборы данных. Главное меню ISPF обычно содержит больше пунктов, чем показано на первой панели, а также инструкции по выводу дополнительных пунктов.

При необходимости откройте раздел 6.14.1, «Создание простого задания», и повторите действия до шага 5. Это выдаст листинг задания для этого упражнения.

Упражнение SDSF 1

При просмотре выходного листинга, предположим, что вам требуется сохранить его в наборе данных для последующего просмотра. В строкк ввода команд введите `PRINT D`. Появится окно, которое запросит ввод имени набора данных, в который требуется выполнить сохранение. Можно использовать уже существующий набор данных или создать новый.

Для этого примера создайте новый набор данных, введя `yourid.cobol.list`. В поле диспозиции введите `NEW`. Нажмите Enter для возврата на предыдущий экран.

Обратите внимание на то, что в верхнем правом углу экрана выводится `PRINT OPENED`. Это означает, что теперь можно распечатать листинг. В строку ввода команд введите `PRINT`. В верхнем правом углу выводится количество распечатанных строк (xxx LINES PRINTED). Это означает, что листинг был записан в созданный вами набор данных. В командную строку введите `PRINT CLOSE`. В верхнем правом углу экрана должна появиться надпись PRINT CLOSED.

Теперь найдем созданный набор данных yourid.cobol.list и просмотрим листинг. Перейдите к `=3.4` и введите ваш идентификатор пользователя. Должен появиться список со всеми вашими наборами данных. Найдите yourid.cobol.list и введите B рядом с ним в поле команды. Должен появиться листинг точно в таком же виде, как и при использовании SDSF. Теперь можно возвратиться в SDSF ST и очистить (`P`) ваш листинг, так как теперь у вас есть постоянная копия.

Возвратитесь в главную панель SDSF и введите `LOG` для вывода журнала всех действий в системе. Здесь можно увидеть большую часть информации, доступной для операционного персонала. Например, в нижней части списка могут выводиться сообщения, ожидающие ответа (Reply), на которые может отвечать оператор.

```
/R xx,/DISP TRAN ALL
```

Перейдите в конец, чтобы просмотреть результаты. Обратите внимание на то, что команды оператора, выводимые SDSF-командой LOG, должны предваряться прямым слэшем (/), чтобы она распознавалась как системная команда.

Теперь введите `M` в строку ввода команд и нажмите F7; это выведет начало журнала. Введите F и ваш идентификатор пользователя, чтобы вывести первую запись, связанную с вашим идентификатором пользователя. Скорее всего, это будет запись о вашем подключении к TSO. Затем введите `F youridX`, где X представляет одно из заданий, переданных на выполнение ранее. Здесь вы сможете увидеть, что ваше задание принято внутренним считывающим модулем JES2, а также следующие строки, указывающие состояние вашего задания при выполнении. Вы можете также увидеть сообщения об ошибке JCL или `youridX started | ended`.

Упражнение SDSF 2

Это упражнение использует функции печати, рассматривавшиеся выше. Сохраните журнал в набор данных, как это делалось в упражнении с печатью.

Упражнение SDSF 3

В этом упражнении вам предстоит вводить команды оператора из экрана Log. Введите следующие команды в строку ввода команд и просмотрите результаты:

/D A,L	Эта команда выводит все активные задания в системе.
/D U,,,A80,24	Эта команда выводит подключенные DASD-тома.
/V A88,OFFLINE	Перейдите в конец, чтобы просмотреть результаты (M F8).
/D U,,,A88,2	Проверьте состояние; обратите внимание на то, что VOLSER не выводится для отключенных томов. Пока том отключен, можно использовать такие утилиты, как ICKDSF, позволяющие отформатировать том.
/V A88,ONLINE	Перейдите в конец, чтобы просмотреть результаты.

/D U,,,A88,2	Проверьте состояние; теперь VOLSER выводится.
/C U=*yourid*	Отменяет задание (в данном случае, ваш TSO-сеанс).
Logon yourid	Повторно подключитесь под своим идентификатором.

7.9.6 Использование TSO REXX и ISPF

Набор данных USER.CLIST содержит REXX-программу под названием ITSODSN. Эту программу можно запустить путем ввода в любой строке ввода команд ISPF следующей команды: TSO ITSODSN. Система запросит у вас имя набора данных, который требуется создать. Вам не требуется вводить `yourid`, так как TSO добавит его к имени, если ваш префикс активен. Система предложит выбрать один из двух типов наборов данных, последовательного или секционированного, и спросит, на каком томе требуется сохранить набор данных. Затем она распределит набор данных, добавив ваш идентификатор пользователя к его имени. Перейдите к пункту =3.4, найдите набор данных и изучите его с использованием опции S, чтобы убедиться в том, что он соответствует вашим требованиям.

Упражнение REXX 1

В программе REXX вы найдете несколько свойств набора данных, имеющих предопределенные значения, например, LRECL и BLKSIZE. Измените программу таким образом, чтобы пользователю предлагалось ввести любые нужные ему свойства набора данных. Можно также изменить другие аспекты программы. Совет: прежде чем начать, сделайте резервную копию программы.

Упражнение REXX 2

REXX в TSO и при пакетной обработке может непосредственно обращаться к другим подсистемам, как уже было показано в этой программе, когда выполнялось прямое распределение набора данных с использованием TSO-команды, заключенной в кавычки. Другой способ выполнения функций вне REXX состоит в использовании host command environment («среды системных команд»). Некоторыми примерами таких host command environments являются:

TSO	
MVS	Для REXX-программы, выполняемой в среде, отличной от TSO.
ISPEXEC	Доступ к среде ISPF в TSO.

Измените REXX-программу таким образом, чтобы после распределения набора данных она открывала его командой ISPF Edit, вводила некоторые данные, выполняла выход с использованием PF3 и использовала =3.4 для проверки вашего набора данных. Помните, что, если набор данных является секционированным (PO), вам потребуется открыть раздел. В качестве имени раздела можно использовать любое имя в виде yourid.name(membername).

Советы:

- Проще использовать второй формат – host command environment, представленный выше.
- Обратите внимание на использование логической конструкции «if then else» в REXX и на «do end» в этой логической конструкции.
- Используйте команду ADDRESS ISPEXEC «edit DATASET(....)»

Часть 2

Программирование приложений в z/OS

В этой части рассматриваются инструменты и утилиты для разработки простой программы, запускаемой в z/OS. Следующие главы представляют студентам процесс проектирования приложения, выбора языка программирования и использования среды выполнения.

8

Проектирование и разработка приложений для z/OS

Цель. Как новому проектировщику или программисту приложений для z/OS, вам будет предложено проектировать и разрабатывать новые программы или изменять существующие программы для достижения бизнес-целей компании. Это занятие требует полного понимания различных требований пользователей к вашему приложению, а также знаний о том, какие системные службы z/OS следует использовать.

В этой главе представлен краткий обзор проектирования, кодирования и тестирования нового приложения. Большая часть этой информации применима ко всем вычислительным платформам в целом, а не только к мэйнфреймам.

После завершения работы над этой главой вы сможете:

- описать роли проектировщика приложений и программиста приложений;
- перечислить основные аспекты проектирования приложения для z/OS;
- описать преимущества и недостатки пакетной обработки в сравнении с оперативной обработкой для приложения;
- кратко описать процесс тестирования нового приложения в z/OS;
- перечислить три преимущества использования z/OS в качестве базовой системы для нового приложения.

8.1 Проектировщики и программисты приложений

Задачи *проектирования* (*designing*) и *разработки* (*developing*) приложения достаточно четко отделены одна от другой, чтобы рассматривать каждую из них в отдельной книге. В крупных организациях, использующих z/OS, выполнением этих задач могут заниматься разные отделы. В этой главе представлен обзор этих ролей и показано, какое место каждая из них занимает в общем представлении о стандартном жизненном цикле разработки приложений в z/OS.

Проектировщик приложений отвечает за определение наилучшего программного решения для важного бизнес-требования. Успех любого проектирования зависит, в том числе, и от знания проектировщиком работы предприятия, осведомленности о других ролях в организации, использующей мэйнфрейм, в частности, о программировании и проектировании баз данных и понимания аппаратного и программного обеспечения предприятия. Если кратко, проектировщик должен иметь общее представление о проекте в целом.

Кроме того, в этом процессе задействована роль аналитика бизнес-систем. Этот специалист отвечает за работу с пользователями из определенного отдела (бухгалтерии, отдела продаж, отдела производственного контроля, производственного отдела, и т. д.), чтобы определить бизнес-требования к приложению. Подобно проектировщику приложения аналитику бизнес-систем нужно иметь широкое понимание бизнес-целей организации и возможностей информационной системы.

Приложение – набор файлов, составляющих программное обеспечение для пользователя.

Проектировщик приложений принимает требования от аналитиков бизнес-систем и конечных пользователей. Кроме того, проектировщик определяет, какие информационные ресурсы должны быть доступны для поддержки приложения. Затем проектировщик приложения составляет техническое задание для реализации программистами приложений.

Программист приложений отвечает за разработку и поддержку приложений. Другими словами, программист разрабатывает, тестирует и доставляет приложения, выполняющиеся на мэйнфрейме, конечным пользователям. На основании технического задания, разработанного проектировщиком приложения, программист разрабатывает приложение, используя различные инструменты. Процесс разработки включает несколько итераций изменений кода и компиляций, компоновки приложений и модульного тестирования.

В процессе разработки проектировщик и программист должны взаимодействовать с другими ролями на предприятии. Программист, например, часто работает в команде вместе с другими программистами, разрабатывающими код для связанных модулей приложения.

После разработки каждый модуль проходит процесс тестирования, который может включать функциональные, интеграционные и системные тесты. После выполнения тестов приложения должны пройти приемочное тестирование сообществом пользователей, чтобы определить, соответствует ли код поставленным требованиям.

Помимо создания кода новых приложений программист отвечает за обслуживание и доработку существующих мэйнфрейм-приложений компании. В действительности, это часто является основной задачей для многих современных программистов мэйнфрейм-приложений. Несмотря на то, что для создания новых программ для мэйнфреймов все еще используется COBOL (Common Business Oriented Language) и PL/I, такие языки как Java набирают популярность точно так же, как и на распределенных платформах.

8.2 Проектирование приложения в z/OS

На ранних этапах проектирования проектировщик приложения принимает решения относительно свойств приложения. Эти решения основываются на множестве критериев, которые необходимо собрать и подробно исследовать, чтобы прийти к решению, приемлемому для пользователя. Решения не являются независимыми друг от друга, в том смысле, что каждое решение оказывает влияние на другие решения, и все решения должны приниматься, исходя из области применения проекта и его ограничений.

Проектирование – задача определения наилучшего программного решения для определенного бизнес-требования

Проектирование приложения для z/OS включает множество этапов, используемых при проектировании приложений для других платформ, включая распределенную среду. Тем не менее, z/OS включает некоторые особые аспекты. В этой главе представлено несколько примеров решений, принимаемых проектировщиком приложений z/OS в процессе проектирования определенного приложения. Этот список не является исчерпывающим, а скорее предназначен для того, чтобы дать вам представление о процессе.

- «Проектирование для z/OS: пакетная или оперативная обработка?».
- «Проектирование для z/OS: источники данных и методы доступа».
- «Проектирование для z/OS: требования к доступности и рабочей нагрузке».
- «Проектирование для z/OS: обработка исключений».

Помимо этих решений, другими факторами, воздействующими на проектирование приложения для z/OS, могут быть выбор одного или нескольких языков программирования и сред разработки. Кроме того, в этой главе обсуждаются следующие аспекты:

- использование наборов символов для мэйнфреймов в разделе «Использование набора символов EBCDIC»;
- использование интерактивной среды разработки (interactive development environment, IDE) в разделе «Использование средств разработки приложений»;
- различия между языками программирования обсуждаются в главе 9, «Использование языков программирования в z/OS».

Помните, что лучше всего заниматься проектированием, постоянно помня о конечном результате. Необходимо знать к чему стремиться прежде чем приступать к проектированию.

8.2.1 Проектирование для z/OS: пакетная или оперативная обработка?

При проектировании приложения для z/OS и мэйнфрейма основной вопрос заключается в том, будет ли приложение выполняться как пакетная программа или как оперативная программа. В некоторых случаях решение очевидно, однако большинство приложений можно спроектировать в соответствии с любой из этих парадигм. Тогда как проектировщику определить, какой подход следует использовать?

Существуют следующие причины для выбора пакетной или оперативной обработки:

- Причины для выбора пакетной обработки:
  - данные хранятся на ленте;
  - транзакции передаются для ночной обработки;
  - пользователь не требует оперативного доступа к данным.
- Причины для выбора оперативной обработки:
  - пользователь требует оперативного доступа к данным;
  - требования к быстрому времени реагирования.

8.2.2 Проектирование для z/OS: источники данных и методы доступа

Обычно проектирование включает следующие аспекты:

- Какие данные требуется сохранять?
- Каким образом будет осуществляться доступ к данным? Это включает выбор метода доступа.
- Являются ли запросы произвольными или предсказуемыми?
- Будет ли выбрана PDS, VSAM или система управления базами данных (СУБД), например, DB2?

8.2.3 Проектирование для z/OS: требования к доступности и рабочей нагрузке

Относительно приложения, которое будет выполняться в z/OS, проектировщик должен быть способен ответить на следующие вопросы:

- Для какого количества данных требуется осуществлять хранение и доступ?
- Есть ли необходимость совместного доступа к данным?
- Каковы требования к времени реагирования?
- Каковы ограничения стоимости для проекта?
- Сколько пользователей будут одновременно осуществлять доступ к приложению?

Каковы требования к доступности приложения (24 часа в день / 7 дней в неделю или от 8:00 до 17:00 в рабочие дни и т. д.)?

8.2.4 Проектирование для z/OS: обработка исключений

Могут ли возникнуть какие-либо необычные ситуации? Если да, тогда нужно учитывать их при проектировании с целью предотвращения отказов в итоговом приложении. Например, нельзя считать, что ввод всегда будет выполняться должным образом.

8.3 Жизненный цикл разработки приложения: обзор

Приложение представляет собой набор программ, соответствующих определенным требованиям (решающих определенные задачи). Решение может находиться на любой платформе или сочетании платформ, с точки зрения оборудования или операционной системы.

Как и в случае с другими операционными системами, разработка приложений в z/OS обычно состоит из нескольких этапов:

- этап проектирования:
  - сбор требований – требования к пользователям, аппаратному и программному обеспечению;
  - выполнение анализа;
  - разработка структуры за несколько итераций:
    - высокоуровневая структура;
    - подробная структура;
- передача структуры программистам приложения;
- кодирование и тестирование приложения;
- выполнение пользовательских тестов – пользователь тестирует приложение на функциональность и практичность;
- выполнение системных тестов:
  - выполнение интеграционного теста (тестирование приложения с использованием других программ, чтобы убедиться в том, что все программы продолжают работать должным образом);
  - выполнение теста производительности (нагрузочного теста) с использованием рабочих данных;
- перенос в рабочую среду – передача операционному персоналу – убедитесь в том, что имеется вся документация (по обучению пользователей, операционные процедуры);
- этап сопровождения – непрерывные повседневные изменения и усовершенствования приложения.

На рис. 8.1 представлена схема процесса на различных этапах жизненного цикла разработки приложения.

Разработка – написание, тестирование и поставка приложения

На рис. 8.2 представлен этап проектирования до момента начала разработки. После сбора, анализа и проверки всех требований и проведения проектирования, можно передавать программные требования программистам приложений.

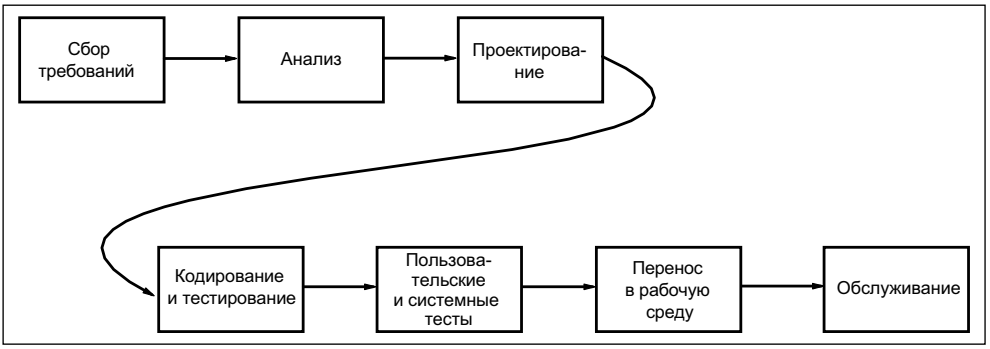


Рис. 8.1. Жизненный цикл разработки приложения

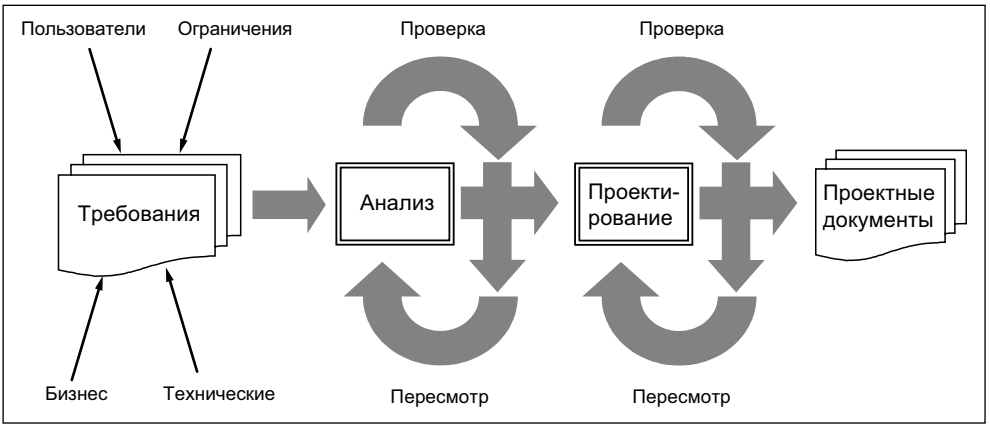


Рис. 8.2. Этап проектирования

Программисты получают проектные документы (программные требования), после чего продолжают итерационный процесс кодирования, тестирования, пересмотра и повторного тестирования, как показано на рис. 8.3.

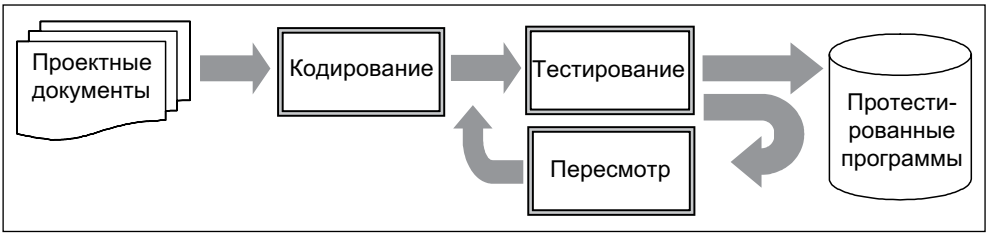


Рис. 8.3. Этап разработки

После тестирования программ программистами они подлежат серии формальных пользовательских и системных тестов. Эти тесты выполняются для того, чтобы убедиться в практичности и функциональности с точки зрения пользователя, а также для проверки функционирования приложения в составе крупной инфраструктуры (рис. 8.4).

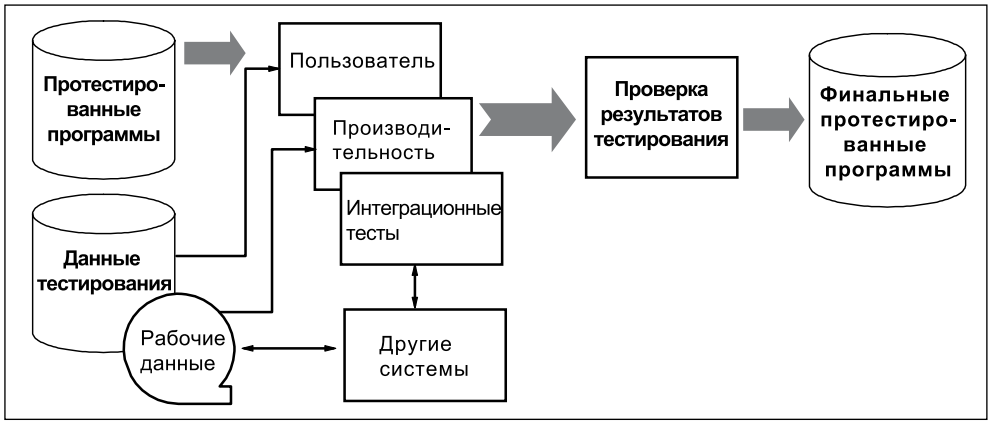


Рис. 8.4. Тестирование

Последний этап жизненного цикла разработки состоит в переносе в рабочую среду и достижении стабильного состояния. Перед переносом в рабочую среду команда разработки должна обязательно предоставить документацию. Она обычно включает документацию по обучению пользователей и операционным процедурам. Обучающая документация знакомит пользователей с новым приложением. Документация по операционным процедурам позволяет операционному персоналу принять ответственность за использование приложения на постоянной основе.

В рабочей среде изменения и усовершенствования осуществляются группой (возможно, той же группой программистов), выполняющей обслуживание. В этой точке жизненного цикла приложения изменения строго контролируются и должны тщательным образом тестироваться перед реализацией в рабочей среде (рис. 8.5).

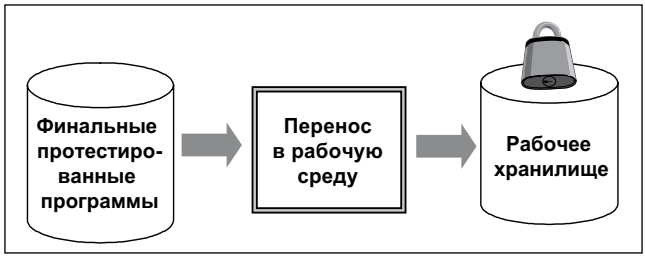


Рис. 8.5. Рабочая среда

Как говорилось выше, для соответствия пользовательским требованиям или решения проблем приложение может быть разработано таким образом, чтобы выполняться на любой платформе или сочетании платформ. Как показано на рис. 8.6, наше приложение может находиться в любой из трех сред: в Интернете, в сети предприятия или на центральном узле. Операционная система должна осуществлять доступ к любой из сред.

Для того чтобы начать процесс проектирования, необходимо сначала оценить, что требуется сделать. Исходя из ограничений проекта, определяется, каким образом

и какими средствами будут достигаться цели проекта. Для этого проводятся собеседования с пользователями (запрашивающими решение проблемы), а также с другими участниками.

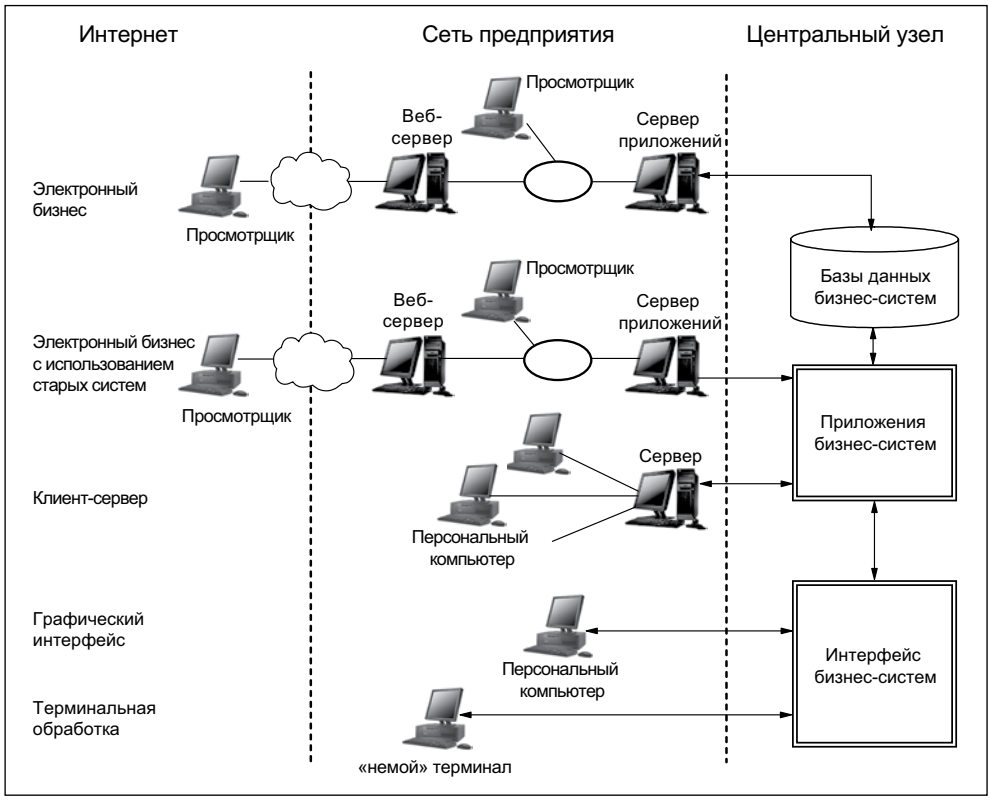


Рис. 8.6. Сложность растущей инфраструктуры

Результаты этих собеседований должны передаваться на все последующие этапы жизненного цикла проекта приложения. На определенных этапах проекта требуется опять обратиться к пользователям, чтобы убедиться в том, что мы правильно поняли их требования и что наше решение соответствует их требованиям. На этих этапах проекта следует также просить пользователей утвердить то, что было сделано, чтобы можно было перейти к следующему этапу проекта.

8.3.1 Сбор требований для проектирования

При проектировании приложений существует множество способов классификации требований, в частности: функциональные требования, нефункциональные требования, новые требования, системные требования, требования обработки, ограничения по разработке и операциям.

Компьютерные приложения работают с данными, расположенными в некотором хранилище, доступ к которым следует осуществлять с локального или удаленного уз-

ла. Приложения управляют данными, выполняя определенную их обработку, после чего представляют результаты тому, кто их запрашивает.

Такое простое описание включает множество процессов и множество операций, имеющих множество различных требований от компьютеров до программных продуктов.

Несмотря на то, что каждый проект приложения представляет особый случай и может иметь множество уникальных требований, некоторые из них являются общими для всех приложений, входящих в одну систему. И не только потому, что они являются частью одного набора приложений, составляющего данную информационную систему, но и потому, что они являются частью одной инсталляции, подключенной к одним и тем же внешним системам.

Платформа – часто относится к операционной системе, означая и ОС, и аппаратное обеспечение (среду)

Одна из проблем, с которой сталкиваются системы в целом, состоит в том, что компоненты разбросаны по разным машинам, разным платформам, и т. д., и каждый из них выполняет свою работу в среде *серверной фермы* (*server farm*).

Важное преимущество подхода zSeries состоит в том, что обслуживание приложений может выполняться с использованием инструментов, имеющихся на мэйнфрейме. Некоторые из этих инструментов мэйнфреймов позволяют различным платформам осуществлять совместный доступ к ресурсам и данным согласованным и безопасным образом в соответствии с рабочей нагрузкой или приоритетом.

Ниже представлен список различных типов требований приложения. Элементы списка не являются взаимоисключающими; некоторые элементы уже включают другие:

- возможность доступа;
- восстанавливаемость;
- удобство обслуживания;
- доступность;
- безопасность;
- подключаемость;
- цели производительности;
- управление ресурсами;
- практичность;
- частота резервного копирования данных;
- портативность;
- веб-службы;
- модифицируемость;
- возможность взаимосвязи;
- предотвращение отказов и анализ сбоев.

8.4 Разработка приложения на мэйнфрейме

После выполнения анализа и принятия решений процесс переходит к программисту приложения. Программисту не дается полная свобода; наоборот, он должен придерживаться спецификаций проектировщика. Однако учитывая, что проектировщик,

скорее всего, не является программистом, могут потребоваться изменения из-за ограничений при программировании. Но на данном этапе проекта речь идет не об изменениях в проекте, а об изменениях в способе выполнения программой того, что проектировщик указал ей делать.

Процесс разработки является итеративным и обычно осуществляется на уровне модулей. Программист обычно следует приведенной ниже процедуре.

1. Разработка модуля.
2. Тестирование функциональности модуля.
3. Внесение исправлений в модуль.
4. Повторение с шага 2 до успешного выполнения.

После тестирования модуля он утверждается и фиксируется; в этом случае, если в него впоследствии будут вноситься изменения, он будет снова тестироваться. После разработки и тестирования достаточного количества модулей, их можно тестировать вместе, используя тесты возрастающей сложности.

Этот процесс повторяется до тех пор, пока не будут разработаны и протестированы все модули. Несмотря на то, что на схеме процесса показано продолжение тестирования после завершения разработки, на самом деле тестирование постоянно выполняется на этапе разработки.

8.4.1 Использование набора символов EBCDIC

Наборы данных z/OS имеют кодировку EBCDIC (Extended Binary Coded Decimal Interchange). Она представляет собой 8-разрядный набор символов, разработанный до того, как такой набор символов ASCII (American Standard Code for Information Interchange) стал общепринятым. С другой стороны, файлы z/OS UNIX имеют кодировку ASCII.

Большинство знакомых вам систем используют ASCII. Важно знать о различиях в схемах кодировки при перемещении данных с ASCII-систем на EBCDIC-системы. Обычно выполняется внутреннее преобразование, например, при передаче текста из эмулятора 3270, выполняющегося на PC в TSO-сеанс. Однако при передаче программ обычно не требуется выполнять такое преобразование, и должна быть задана двоичная передача. Иногда даже при передаче текста проблемы возникают с некоторыми символами, например, с символом OR (|) или с логическим *не (not),* и программист должен проверить итоговое значение преобразованного символа.

Список символов в наборах EBCDIC и ASCII представлен в Приложении D, «Таблица EBCDIC – ASCII», которое может быть полезным в этом обсуждении. И ASCII, и EBCDIC являются 8-разрядными наборами символов. Различие состоит в способе назначения битов конкретным символам. Ниже приведено несколько примеров:

Символ	EBCDIC	ASCII
A	11000001 (x'C1')	01000001 (x'41')
B	11000010 (x'C2')	01000010 (x'42')
a	10000001 (x'81')	01100001 (x'61')
1	11110001 (x'F1')	00110001 (x'31')
пробел	01000000 (x'40')	00100000 (x'20')

Несмотря на то, что назначение битов в ASCII может выглядеть более логичным, имеется огромное количество существующих данных в кодировке EBCDIC и огромное количество программ, чувствительных к набору символов, что делает преобразование всех существующих данных и программ в кодировку ASCII неосуществимым.

Набор символов имеет последовательность сортировки, соответствующую двоичному значению битов символов. Например, A имеет меньшее значение, чем B и в ASCII, и в EBCDIC. Последовательность сортировки важна для сортировки, а также практически для всех программ, просматривающих текстовые строки и управляющих ими. Последовательность сортировки для основных групп символов в этих двух наборах символов имеет следующий вид:

	EBCDIC	**ASCII**
Наименьшее значение:	пробел	пробел
	знаки препинания	знаки препинания
	нижний регистр	цифры
	верхний регистр	верхний регистр
Наибольшее значение:	цифры	нижний регистр

Например, в EBCDIC «a» меньше, чем «A», тогда как в ASCII «a» больше, чем «A». Цифровые символы меньше любой буквы алфавита в ASCII, однако больше любой буквы в EBCDIC. A-Z и a-z представляют собой две непрерывные последовательности в ASCII. В EBCDIC существуют промежутки между некоторыми буквами. В ASCII, если отнять A от Z, получится 25. Если отнять A от Z в EBCDIC, получится 40 (в связи с промежутками в двоичных значениях между некоторыми буквами).

Преобразование простых текстовых строк между ASCII и EBCDIC выполняется просто. Ситуация усложняется, если преобразуемый символ отсутствует в стандартном наборе символов целевого кода. Хорошим примером является символ логического *не (not),* используемый в одном из основных языков программирования для мэйнфреймов (PL/I); в наборе символов ASCII нет соответствующего символа. Подобным образом некоторые символы ASCII, используемые в программировании на языке C, отсутствовали в первоначальном наборе символов EBCDIC, хотя позже и были добавлены в EBCDIC. Все еще существует некоторая путаница в связи со знаком цента (¢) и символом крышки (^), а также с некоторыми другими редко использующимися символами.

Кроме того, на мэйнфреймах используется несколько версий двухбайтовых наборов символов (DBCS) главным образом для азиатских языков. Такие же наборы символов используются некоторыми программами на PC.

В традиционном программировании для мэйнфреймов не используются специальные символы для завершения полей. В частности, не используются нулевые символы и символы новой строки (или пары символов CL/LF). Не существует понятий *двоичного* и *текстового* файла. При правильном программировании байты могут интерпретироваться как относящиеся к EBCDIC, ASCII или другой кодировке. При отправке таких файлов на принтер мэйнфрейма он попытается интерпретировать их как EBCDIC-символы, так как принтер чувствителен к набору символов. Веб-сервер z/OS Web обычно сохраняет ASCII-файлы, так как данные будут интерпретироваться браузерами на PC, ожидающими ASCII-данных. Если никто не будет пытаться распечатывать ASCII-файлы на принтере мэйнфрейма (или выводить их на терминале 3270), системе все равно, какой набор символов используется.

8.4.2 Unicode на мэйнфрейме

Unicode является промышленным стандартом и представляет собой 16-разрядный набор символов, предназначенный для отображения текста и символов во всех современных языках и ИТ-протоколах. Мэйнфреймы (использующие EBCDIC для однобайтовых символов), PC и различные RISC-системы используют одинаковую кодировку Unicode.

Поддержкой Unicode занимается Unicode Consortium (http://www.unicode.org/).

В настоящее время Unicode все чаще используется в мэйнфрейм-приложениях. Последние мэйнфреймы zSeries содержат множество уникальных аппаратных инструкций для поддержки Unicode. На момент написания этой книги, Unicode на мэйнфреймах использовался главным образом в Java. Однако программные продукты промежуточного уровня для z/OS также начинают использовать Unicode, и в этой области, очевидно, следует ожидать изменений в ближайшем будущем.

8.4.3 Интерфейсы для программистов приложений для z/OS

При разработке операционных систем для удовлетворения потребностей компьютерного рынка создаются также и приложения для выполнения на этих операционных системах. За прошедшие годы было разработано множество приложений, выполняющихся в z/OS, а позднее и в UNIX. Для того чтобы обеспечить клиентов UNIX-приложениями, z/OS содержит полную операционную систему UNIX в дополнение к своим традиционным интерфейсам z/OS. Реализация UNIX-интерфейсов в z/OS называется z/OS UNIX System Services (или кратко z/OS UNIX).

Самым распространенным интерфейсом для разработчиков z/OS является TSO/E с его панельным интерфейсом, ISPF и использованием терминала 3270. Как правило, разработчики используют эмуляторы терминала 3270, запускаемые на персональных компьютерах, а не настоящие терминалы 3270. Эмуляторы предоставляют разработчикам дополнительные функции, в частности, возможность создания нескольких сеансов и загрузка/выгрузка кода и данных с PC. TSO/E и прочие пользовательские интерфейсы для z/OS описываются в главе 4, «TSO/E, ISPF и UNIX: интерактивные средства z/OS».

Разработка программ в z/OS обычно предполагает использование построчного редактора для работы с файлами исходного кода, использование пакетных задания для компиляции и множество механизмов для тестирования кода. Для распространенных языков доступны интерактивные отладчики, основанные на функциях терминалов 3270. В этой главе рассматриваются инструменты и утилиты для разработки простой программы, выполняемой в z/OS.

Разработка с использованием только подсистемы z/OS UNIX может выполняться через сеансы Telnet (из которых доступен редактор vi), через 3270 и TSO/E при использовании других редакторов или через сеансы X Window System с персональных компьютеров с запущенными X-серверами. Интерфейсы X-сервера используются реже.

Существуют альтернативные методы, которые можно использовать в сочетании с различными программными продуктами промежуточного уровня. Например, продукты WebSphere содержат средства разработки с графическим интерфейсом для персональных компьютеров. Эти средства используют TCP/IP-связь с z/OS для авто-

матического вызова элементов мэйнфрейма, необходимых на этапах разработки и тестирования нового приложения.

В этой книге использование оперативных приложений и программных продуктов промежуточного уровня рассматривается в части 3 «Оперативная рабочая нагрузка в z/OS», включающей такие вопросы, как сетевые коммуникации, управление базами данных и задачи веб-сервера.

8.4.4 Использование инструментов разработки приложений

Составление тщательно протестированного кода требует использования определенных инструментов на мэйнфрейме. Основным инструментом для программиста является редактор ISPF.

При разработке традиционных процедурных программ на таких языках, как COBOL и PL/I, программист часто подключается к мэйнфрейму и использует IDE или редактор ISPF для изменения кода, его компиляции и запуска. Программист использует общее хранилище (например, IBM Software Configuration Library Manager – SCLM) для хранения кода, находящегося в процессе разработки. Хранилище позволяет программисту осуществлять извлечение или возврат кода и гарантирует, что программисты не будут нарушать работу друг друга. SCLM входит в ISPF в качестве опции главного меню.

Для простоты исходный код можно хранить и обрабатывать в секционированном наборе данных (partitioned data set, PDS). Однако использование PDS не позволит ни осуществлять управление изменениями, ни предотвратить множество изменений в одной версии кода таким же образом, как это делает SCLM. Таким образом, понятия «извлечение» или «сохранение» при работе с SCLM можно заменять на «редактирование раздела PDS» или «сохранение раздела PDS».

Исполняемый код – файл программы, готовый к выполнению в определенной среде

После внесения изменений в исходный код программист передает на выполнение JCL-файл для компиляции исходного кода, связывания модулей приложения и создания исполняемого кода для тестирования. Программист проводит «модульные тесты» функциональности программы, используя инструменты мониторинга и просмотра заданий для отслеживания запущенных программ, просмотра выходных данных и внесения соответствующих исправлений в исходный код или другие объекты. Иногда при возникновении отказа программа создает «дамп» памяти. Программист может также использовать инструменты для запроса вывода дампа и для трассировки выполняющегося кода с целью определения точек отказов.

Некоторые программисты приложений для мэйнфреймов перешли на использование инструментов интерактивной среды разработки (IDE), чтобы ускорить процесс редактирования, компиляции и тестирования. IDE позволяют программистам приложений осуществлять редактирование, тестирование и отладку исходного кода на рабочей станции, а не непосредственно на мэйнфрейм-системе. Использование IDE особенно полезно для разработки «гибридных» приложений, использующих централизованные программы или транзакционные системы, но при этом содержащих пользовательский интерфейс, подобный веб-браузеру.

По окончании разработки и тестирования компонентов программист приложений выполняет их упаковку в соответствующем формате развертывания и передает их группе, координирующей развертывание рабочего кода.

К средствам разработки приложений, доступным в z/OS, относятся:

- Language Environment®;
- C/C++ IBM Open Class® Library;
- DCE Application Support1;
- Encina® Toolkit Executive2;
- C/C++ с отладчиком Debug Tool;
- DFSORT;
- GDDM®-PGF;
- GDDM-REXX;
- HLASM Toolkit;
- традиционные языки, такие как COBOL, PL/I и Fortran.

8.4.5 Сеанс отладки

Программист приложений проводит «модульный тест» для тестирования функциональности определенного разрабатываемого модуля, используя средства мониторинга и просмотра заданий, такие как SDSF (описывается в разделе 6.8 «Общее представление о SDSF»), для трассировки выполнения заданий компиляции, просмотра выходных данных компилятора и проверки результатов модульных тестов. Если необходимо, программист вносит соответствующие исправления в исходный код и другие объекты.

Иногда при возникновении отказа программа создает «дамп» памяти. Когда это происходит, программист приложений для z/OS может использовать такие инструменты, как IBM Debug Tool и IBM Fault Analyzer для запроса вывода дампа и для трассировки выполняющегося кода с целью поиска ошибки или неправильно выполняющегося кода.

Стандартный сеанс разработки включает следующие действия:

1. Подключение к z/OS.
2. Вход в ISPF и открытие/извлечение исходного кода из хранилища SCLM (или PDS).
3. Редактирование исходного кода с внесением требуемых изменений.
4. Передача на выполнение JCL-кода для компоновки приложения и тестового прогона.
5. Переключение в SDSF для просмотра состояния выполняющегося задания.
6. Просмотр выходных данных задания в SDSF с целью проверки на ошибки.
7. Просмотр выходного дампа для поиска ошибок («багов»[1]).

[1] Происхождение термина «программный баг» (*англ.* bug –насекомое) часто связывается с лейтенантом военно-морских сил США Грейс Муррей Хоппером (Grace Murray Hopper) и относится к 1945 г. Как рассказывают, лейтенант Хоппер тестировал Mark II Aiken Relay Calculator в Гарвардском университете. В один из дней программа, которая прежде работала, неожиданно загадочным образом отказала. В результате осмотра оператор обнаружил моль, попавшую между точками переключения цепи и ставшую причиной короткого замыкания (первые калькуляторы занимали большую площадь и содержали десятки тысяч вакуумных ламп). Запись в журнале от 9 сентября 1945 года содержала саму моль и подпись: «Первый случай обнаружения насекомого (bug)», а также сообщала, что они «избавили от насекомых (debugged) машину».

8. Повторное выполнение компиляции/связывания/запуска и просмотра состояния.
9. Проверка правильности выходных данных задания.
10. Сохранение исходного кода в SCLM (или PDS).

Некоторые программисты приложений для мэйнфреймов перешли на использование инструментов интерактивной среды разработки (IDE), чтобы ускорить процесс редактирования, компиляции и тестирования. Инструменты IDE, такие как WebSphere Studio Enterprise Developer, позволяют выполнять редактирование исходного кода на рабочей станции, а не непосредственно на центральной системе, выполнять компиляцию «вне платформы» и удаленную отладку.

Использование IDE в особенности полезно при построении гибридных приложений, использующих централизованные программы на языке COBOL или системы транзакций, такие как CICS и IMS, но при этом содержащие пользовательский интерфейс, подобный веб-браузеру. IDE обеспечивает унифицированную среду для разработки как компонентов обработки оперативных транзакций (online transaction processing, OLTP) на высокоуровневом языке, так и компонентов пользовательского HTML-интерфейса переднего плана. После разработки и тестирования компонентов выполняется их упаковка в соответствующем формате развертывания, затем они передаются группе, координирующей развертывание рабочего кода.

Помимо создания кода новых приложений программист приложений отвечает за обслуживание и доработку существующих мэйнфрейм-приложений компании. В действительности это часто является основной задачей для многих современных программистов мэйнфрейм-приложений, использующих высокоуровневые языки. Несмотря на то, что для создания новых программ для мэйнфреймов часто все еще используется COBOL и PL/I, такие языки как Java набирают популярность точно так же, как и на распределенных платформах.

Транзакция – действие или запрос. Выполняет обновление главных файлов при заказах, изменениях, добавлениях и т. д.

Тем не менее, для тех, кто интересуется традиционными языками, скажем, что все еще продолжается интенсивная разработка программ для мэйнфреймов на высокоуровневых языках, таких как COBOL и PL/I. Существует много тысяч программ в рабочих средах на мэйнфрейм-системах по всему миру, и эти программы критически важны для повседневной работы корпораций, которые их используют. Программисты на языке COBOL и других высокоуровневых языках нужны для поддержки существующего кода и внесения обновлений и изменений в эти программы.

Кроме того, многие корпорации продолжают разрабатывать новые приложения на языке COBOL и других традиционных языках, и IBM продолжает совершенствовать компиляторы высокоуровневых языков, включая в них новые функции и возможности, позволяющие этим языкам продолжать использовать новые технологии и форматы данных.

8.4.6 Выполнение системного тестирования

Разница между тестированием на этом этапе и тестированием на этапе разработки состоит в том, что теперь выполняется тестирование приложения в целом, а также совместно с другими приложениями. Кроме того, выполняются тесты, которые мож-

но осуществлять только после окончания разработки приложения, так как нам необходимо знать, как работает приложение в целом, а не только его часть.

На данном этапе выполняются следующие тесты:

- пользовательское тестирование – тестирование приложения на предмет функциональности и практичности.
- интеграционное тестирование – новое приложение тестируется совместно с другими приложениями, чтобы убедиться в том, что они взаимодействуют должным образом.
- тестирование производительности (нагрузочный тест) – приложение тестируется с использованием реальных рабочих данных или, по меньшей мере, данных реальных рабочих объемов, чтобы увидеть, как работает приложение в условиях большой нагрузки.

Результаты пользовательских и интеграционных тестов нужно проверять, чтобы убедиться в их допустимости. Кроме того, производительность приложения должна соответствовать требованиям. Любые проблемы, возникающие во время этих тестов, нужно решать до переноса в рабочую среду. Многие проблемы, возникающие на этапе тестирования, являются индикатором того, насколько хорошо было выполнено проектирование.

8.5 Перенос в рабочую среду на мэйнфрейме

Перенос в рабочую среду – это не просто нажатие кнопки, которая сообщает, что теперь приложение готово к выполнению в рабочей среде. Это нечто гораздо более сложное. И в разных проектах способ переноса программы в рабочую среду может изменяться. В некоторых случаях, когда имеется существующая система, которую требуется заменить, может быть принято решение о параллельном выполнении в течение определенного периода времени до переключения на новое приложение. В этом случае старая и новая системы в течение этого периода работают с одинаковыми данными, после чего результаты сравниваются. Если после некоторого периода времени результаты удовлетворительны, выполняется переход на новое приложение. Если обнаруживаются проблемы, их можно исправить и продолжить параллельное выполнение до тех пор, пока не прекратиться возникновение новых проблем.

В других случаях мы работаем с новой системой, и у нас может быть только один день для того, чтобы начать ее использовать. В случае новой системы обычно выполняется замена некоторой системы, даже если это ручная система, поэтому при желании все равно можно выполнить параллельное тестирование.

Какой бы метод ни использовался для переноса в рабочую среду, все равно остаются какие-то мелочи, которые следует решить до передачи системы операционному персоналу. Одной из задач является разработка документации системы, а также процедур ее запуска и использования. Требуется обучить всех, кто будет взаимодействовать с системой.

После завершения создания документации и обучения можно передать ответственность за работу приложения операционному персоналу, а ответственность за

сопровождение приложения группе сопровождения. В некоторых случаях сопровождением приложений также занимается группа разработки.

На данном этапе жизненный цикл разработки приложений достигает стабильного состояния, и происходит переход к этапу сопровождения приложения. С этого момента только лишь применяются усовершенствования и выполняются повседневные изменения в приложении. Так как приложение теперь находится под контролем за изменениями, все изменения требуют тестирования в соответствии с процессом контроля за изменениями до их принятия в рабочей среде. Таким образом, обеспечивается стабильная работа приложения с точки зрения конечных пользователей.

8.6 Заключение

В этой главе описываются роли проектировщика приложений и программиста приложений. Обсуждение направлено на то, чтобы выявить типы решений, задействованных в проектировании и разработке приложения, выполняющегося в среде мэйнфреймов. Это не означает, что процесс намного отличается от других платформ, однако некоторые вопросы и выводы могут отличаться.

Затем в этой главе описывается жизненный цикл проектирования и разработки приложения, выполняющегося в z/OS. Процесс начинается с этапа сбора требований, на котором проектировщик приложения анализирует пользовательские требования, чтобы увидеть, как их удовлетворить. Может существовать множество вариантов решения; цель этапов анализа и проектирования состоит в том, чтобы обеспечить выбор оптимального решения. В данном случае под «оптимальным» не понимается «самое быстрое», хотя время является важным аспектом любого проекта. Вместо этого, оптимальным называется наилучшее в целом решение, с точки зрения пользовательских требований и анализа задачи.

Набор символов EBCDIC отличается от набора символов ASCII. Посимвольное преобразование между этими двумя наборами символов выполняется просто. С учетом последовательностей сортировки различия становятся более значимыми, и программы преобразования из одного набора символов в другой могут быть простыми или довольно сложными. Набор символов EBCDIC стал общепринятым стандартом до того, как современный 8-разрядный набор символов ASCII получил широкое применение.

По окончании этапа проектирования наступает очередь программиста. Теперь программист должен преобразовать проект приложения в безошибочный программный код. На протяжении этапа разработки программист тестирует код по мере добавления каждого модуля в общую программу. Он должен исправить все обнаруженные проблемы в логике и добавить обновленные модули к завершенному набору протестированных программ.

Приложение редко существует в изоляции. Обычно приложение является частью крупного набора приложений, где выходные данные одного приложения являются входными данными следующего. Чтобы убедиться в том, что новое приложение не вызывает проблем при внедрении в более крупный набор приложений, программист приложения проводит системный тест или интеграционный тест. Эти тесты сами

проектируются, и результаты многих тестов проверяются непосредственными пользователями приложений. Если во время системного теста обнаруживаются какие-либо проблемы, их необходимо решить и повторить тест, прежде чем можно будет перейти к следующему этапу процесса.

После успешного выполнения системного теста приложение готово к переносу в рабочую среду. Этот этап иногда называется внедрением приложения. После внедрения код приложения подлежит более тщательному контролю. Предприятию не следует вносить изменения в рабочую систему, не будучи уверенным в их надежности. На большинстве z/OS-узлов существуют строгие правила внедрения приложений (или модулей внутри приложений), позволяющие не допустить попадания непротестированного кода в «чистую» систему.

На данном этапе жизненного цикла приложение достигло стабильного состояния. Изменения, вносимые в рабочее приложение, представляют собой усовершенствования, функциональные изменения (например, изменения программ ведения ведомостей в связи с изменениями в налоговом законодательстве) или исправления.

Основные термины в этой главе

Приложение	ASCII	База данных
Проектирование	Разработка	EBCDIC
Исполняемый код	Платформа	Транзакция

8.7 Контрольные вопросы

Чтобы проверить понимание материала этой главы, ответьте на следующие вопросы:

1. В чем различия между проектировщиком приложений и программистом приложений? Какая из этих ролей должна иметь общее представление о проекте в целом?
2. На каком этапе жизненного цикла разработки приложения проектировщик проводит собеседования?
3. Какая цель использования хранилища для управления исходным кодом?
4. Какие существуют этапы жизненного цикла разработки приложения? Вкратце расскажите, что происходит на каждом этапе.
5. Если бы вы были проектировщиком в конкретном проекте и нужно очень быстро внедрить новое приложение в рабочую среду, какие решения вы бы приняли, чтобы сократить общее время проекта?
6. На этапе системного тестирования выполняется тест производительности приложения. Почему следует использовать рабочие данные при проведении этого теста?
7. Назовите возможные причины для решения использовать в приложении пакетную, а не оперативную обработку.
8. Почему не получается сохранять все документы в формате ASCII, чтобы их не приходилось преобразовывать из формата EBCDIC?

9

Использование языков программирования в z/OS

Цель. Как новому программисту приложений для z/OS, вам необходимо знать о том, какие языки программирования поддерживаются в z/OS, и как определить, какой из них является наилучшим для поставленных требований.
После завершения работы над этой главой вы сможете:

- перечислить некоторые языки программирования для мэйнфрейма;
- объяснить различия между компилируемым языком и интерпретируемым языком;
- создать простую CLIST- или REXX-программу;
- выбрать правильную организацию файлов данных для оперативного приложения;
- сравнивать преимущества высокоуровневых языков и ассемблера;
- объяснить связь между именем набора данных, DD-именем и именем файла в программе;
- объяснить, как использование z/OS Language Environment влияет на решения, принимаемые проектировщиком приложений.

9.1 Обзор языков программирования

Язык программирования – средство общения между человеком и компьютером

Язык программирования является средством общения между человеком и компьютером. Он нужен потому, что компьютер работает только со своим машинным языком (состоящим из битов и байтов). Для человека машинный язык является сложным и медленным в использовании. Поэтому написание программ выполняется на компьютерном языке, после чего они преобразуются в машинный язык, с которым работает компьютер.

Поколение – этапы эволюции компьютерных языков

Существует множество компьютерных языков, являющихся результатом эволюции от машинного языка к более естественному способу письма. Некоторые языки были адаптированы к типам приложений, для которых они предназначены, и к подходу, используемому в проектировании. Для обозначения этапов этой эволюции используется слово *поколение*.

Ниже приведена классификация компьютерных языков.

1. Машинный язык, 1-е поколение, непосредственный машинный код.
2. Ассемблер, 2-е поколение, использование мнемоники для представления инструкций, преобразуемых в машинный код программой ассемблирования, в частности языком ассемблер.
3. Процедурные языки, 3-е поколение, также называются высокоуровневыми языками (high-level languages, HLL); сюда относятся Pascal, FORTRAN, Algol, COBOL, PL/I, Basic и C. Написанную программу, называемую исходной программой, необходимо транслировать посредством этапа компиляции.
4. Непроцедурные языки, 4-е поколение, также имеют общее название 4GL; используются для предопределенных функций в приложениях, предназначенных для работы с базами данных, генераторах отчетов, запросах; сюда относятся RPG, CSP, QMF™.
5. Визуальные языки программирования, использующие мышь и значки, такие как VisualBasic и VisualC++.
6. HyperText Markup Language, используемый для написания веб-документов.
7. Объектно-ориентированные языки, OO-технология; сюда относятся Smalltalk, Java и C++.
8. Прочие языки, например для написания 3D-приложений.

Каждый компьютерный язык развивается самостоятельно в условиях создания и адаптации к новым стандартам. В следующих разделах мы рассмотрим некоторые из наиболее используемых компьютерных языков, поддерживаемых в z/OS.

- Ассемблер – «Использование ассемблера в z/OS».
- COBOL – «Использование COBOL в z/OS».
- PL/I – «Использование PL/I в z/OS».
- C/C++ – «Использование C/C++ в z/OS».
- Java – «Использование Java в z/OS».
- CLIST – «Использование CLIST в z/OS».
- REXX – «Использование REXX в z/OS».

К этому списку можно добавить использование скриптов оболочки и языка PERL в среде z/OS UNIX System Services.

Что касается рассматриваемых компьютерных языков, в этой главе описывается их эволюция и классификация. Среди них есть процедурные и непроцедурные, компилируемые и интерпретируемые, машинно-зависимые и машинно-независимые языки.

Программы на ассемблере являются *машинно-зависимыми* (*machine-dependent*), так как язык представляет собой символьную версию машинного языка того компьютера, на котором выполняется программа. Инструкции ассемблера на разных компьютерах могут отличаться, поэтому программа на ассемблере, написанная для одного компьютера, может не работать на другом компьютере. Скорее всего, ее придется переписать, используя набор инструкций другого компьютера. Программа, написанная на высокоуровневом языке, будет работать на других платформах, но ее потребуется перекомпилировать в машинный язык целевой платформы.

Большинство высокоуровневых языков, рассматриваемых в этой главе, являются *процедурными языками* (*procedural languages*). Этот тип языков хорошо подходит для написания структурированных программ. *Непроцедурные языки* (*non-procedural languages*), такие как SQL и RPG, больше подходят для специальных целей, таких как генерирование отчетов.

Большинство высокоуровневых языков компилируются в машинный код, однако некоторые из них являются интерпретируемыми. Результатом компиляции является машинный код, очень эффективный для многократного выполнения. Программы на интерпретируемых языках анализируются, интерпретируются и выполняются при каждом запуске программы. При использовании интерпретируемых языков уменьшается время составления программы, однако увеличивается использование машинных ресурсов.

Преимущества компилируемых и интерпретируемых языков более подробно рассматриваются в разделе 9.11 «Компилируемые и интерпретируемые языки».

9.2 Выбор языка программирования в z/OS

При разработке программы, предназначенной для выполнения в z/OS, выбор языка программирования может определяться следующими аспектами:

- Какой тип имеет приложение?
- Каковы требования к времени реагирования?
- Каковы бюджетные ограничения для разработки и последующей поддержки?
- Каковы временные ограничения проекта?
- Нужно ли составлять некоторые подпрограммы на других языках из-за преимуществ определенного языка по сравнению с основным выбранным языком?
- Использовать ли компилируемый или интерпретируемый язык?

В следующих разделах рассматриваются особенности некоторых языков, поддерживаемых на мэйнфреймах.

9.3 Использование ассемблера в z/OS

Ассемблер – компилятор для программ, написанных на языке ассемблера

Ассемблер является символьным языком программирования, который можно использовать для написания последовательностей инструкций вместо составления программ на машинном языке. Этот символьный язык программирования по форме и содержанию является ближайшим к машинному языку. Поэтому ассемблер является отличным языком для написания программ, имеющих следующие требования:

- требуется полный контроль над программой, вплоть до уровня отдельных байтов или битов;
- необходимо написать подпрограммы[1] для функций, отсутствующих в других символьных языках программирования, таких как COBOL, FORTRAN или PL/I.

Компилятор – программный продукт, выполняющий преобразование набора операторов высокоуровневого языка в низкоуровневое представление

Ассемблер состоит из операторов, представляющих инструкции либо комментарии. Операторы-инструкции являются рабочей частью языка, и они разделяются на следующие три группы:

- *Машинная инструкция* (*machine instruction*) является символьным представлением инструкции машинного языка или наборов инструкций, таких как:
  - IBM Enterprise Systems Architecture/390 (ESA/390)
  - IBM z/Architecture

  Название «машинная инструкция» используется потому, что ассемблер преобразует ее в машинный код, выполняемый компьютером.
- *Инструкция ассемблера* (*assembler instruction*) представляет собой запрос к ассемблеру для выполнения определенных операций во время ассемблирования исходного модуля; например, для определения констант данных, резервирования областей хранения и определения конца исходного модуля.
- *Макроинструкция* (*macro instruction*) или *макрос* (*macro*) представляет собой запрос к ассемблеру для обработки предопределенной последовательности инструкций, называемой *макроопределением* (*macro definition*). Из этого определения ассемблер генерирует машинные инструкции и инструкции ассемблера, которые он впоследствии обрабатывает, как если бы они были частью первоначально введенного кода в исходном модуле.

Ассемблер создает листинг программы, содержащий информацию, сгенерированную на различных этапах процесса ассемблирования[2]. Он является полноценным компилятором для программ, написанных на языке ассемблера.

[1] Подпрограммами называются программы, часто вызываемые другими программами и по определению требующие высокой производительности. Ассемблер является отличным языком для написания подпрограмм.

[2] Листинг программы не содержит *всей* информации, генерируемой в процессе ассемблирования. Для захвата всей информации, которая может находиться в листинге (и сверх этого), программист z/OS может задать опцию ассемблера ADATA, чтобы ассемблер генерировал файл SYSADATA в качестве выходных данных. Файл SYSADATA не предназначен для чтения человеком; его содержимое имеет формат, предназначенный для обработки специальным инструментом. Использование файла SYSADATA проще для обработки инструментами, чем извлечение подобных данных посредством «listing scrapers» (анализаторов листингов), что применялось ранее.

Компоновщик – осуществляет компоновку (связывание) объектных модулей в загрузочные модули

Кроме того, ассемблер генерирует информацию для других обработчиков, в частности для *компоновщика* (или *редактора связей*, как он назывался в ранних версиях операционной системы). Прежде чем компьютер сможет выполнить вашу программу, объектный код (называемый объектным модулем или просто OBJ) должен пройти через процесс разрешения адресов размещения инструкций и данных. Этот процесс называется *редактированием связей* (*linkage-editing* или *link-editing*) и выполняется компоновщиком.

Загрузочный модуль – генерируется редактором связей из объектных модулей; готов к загрузке и выполнению

Компоновщик или редактор связей (дополнительные сведения см. в разделе 10.3.7 «Как используется редактор связей?») использует информацию в объектных модулях для их объединения в загрузочные модули. При вызове программы загрузочный модуль, сгенерированный компоновщиком, загружается в виртуальную память. После загрузки программы его можно выполнять.

Эти этапы представлены на рис. 9.1.

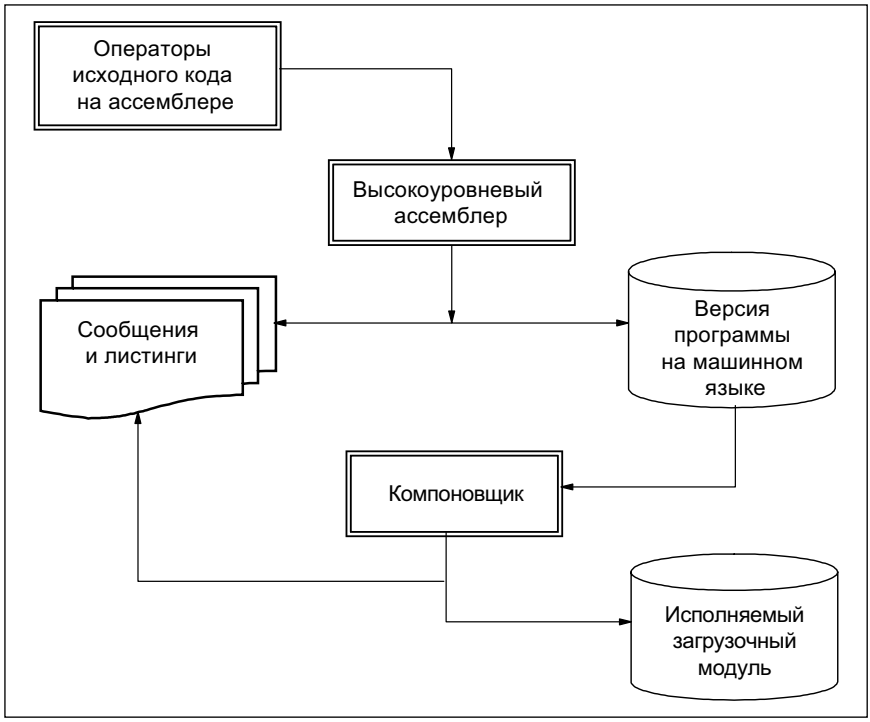


Рис. 9.1. Преобразование исходного кода на ассемблере в исполняемый модуль

Дополнительные сведения. Дополнительные сведения об использовании языка ассемблера в z/OS см. в руководствах *HLASM General Information*, GC26-4943 и *HLASM Language Reference*, SC26-4940. Эти книги доступны в Интернете по адресу:

`http://www.ibm.com/servers/eserver/zseries/zos/bkserv/find_shelves.html`

9.4 Использование COBOL в z/OS

COBOL (Common Business-Oriented Language) представляет собой сходный с английским язык программирования, широко используемый для разработки бизнес-ориентированных приложений в сфере обработки коммерческих данных. В свое время COBOL был почти общим обозначением языков программирования на компьютере. Однако в этой главе все упоминания COBOL относятся к продукту IBM Enterprise COBOL для z/OS и OS/390®.

Отладка – под отладкой программ понимается поиск ошибок в исходном коде (программной логике)

Помимо традиционных возможностей, предоставляемых языком COBOL, эта версия COBOL способна посредством использования функций COBOL осуществлять интеграцию COBOL-приложений в веб-ориентированные бизнес-процессы. Возможности этого продукта позволяют разработчикам приложений следующее:

- использовать новые функции отладки в Debug Tool;
- обеспечивать взаимодействие с Java, когда приложение выполняется в Java-зависимом регионе в IMS;
- упростить компонентное представление COBOL-программ и обеспечить взаимодействие с Java-компонентами в распределенных приложениях;
- обеспечивать обмен и использование данных в стандартных форматах, включая XML и Unicode.

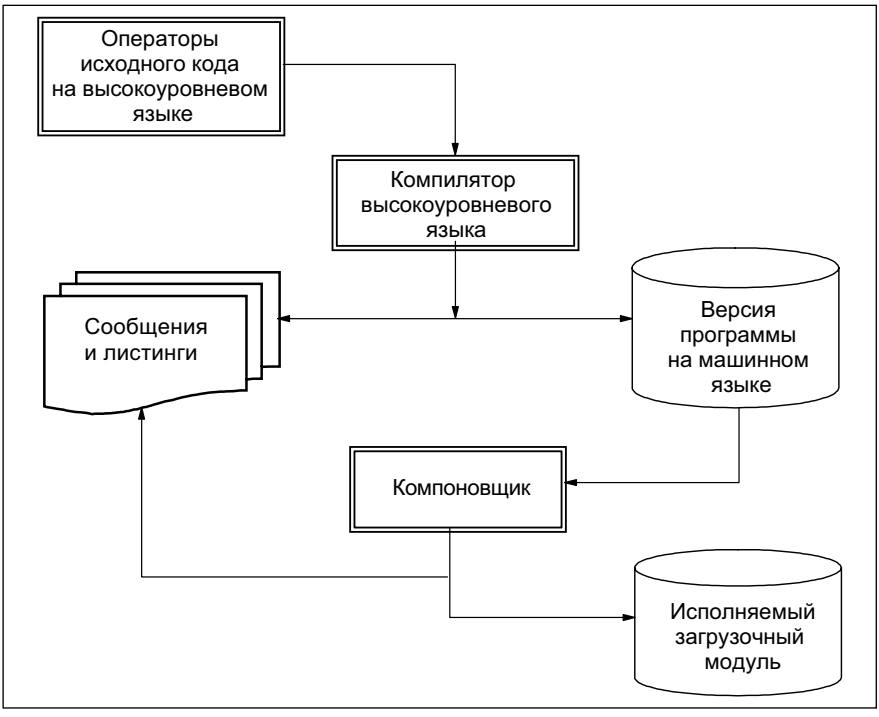


Рис. 9.2. Преобразование исходного кода на высокоуровневом языке в исполняемый модуль

Использование Enterprise COBOL для z/OS и OS/390 позволяет обеспечить взаимодействие между COBOL- и Java-приложениями в среде электронного бизнеса.

Компилятор COBOL создает листинг программы, содержащий всю информацию, сгенерированную во время компиляции. Компилятор также создает информацию для других обработчиков, в частности, для компоновщика.

Прежде чем компьютер сможет выполнить вашу программу, объектный модуль должен пройти еще один этап обработки для разрешения адресов размещения инструкций и данных. Этот процесс называется *редактированием связей* и выполняется компоновщиком.

Компоновщик использует информацию в объектных модулях для их объединения в загрузочные модули (дополнительные сведения см. в разделе 10.3.7 «Как используется редактор связей?»). При вызове программы загрузочный модуль, сгенерированный компоновщиком, загружается в виртуальную память. После загрузки программы его можно выполнять.

На рис. 9.2 представлен процесс трансляции операторов исходного кода на языке COBOL в исполняемый загрузочный модуль.

Этот процесс подобен трансляции программ на ассемблере. В действительности такой же процесс используется во всех компилируемых высокоуровневых языках.

9.4.1 Формат программы на языке COBOL

За исключением операторов COPY и REPLACE и маркера конца программы, все операторы, записи, параграфы и разделы исходной программы на языке COBOL группируются в четыре раздела (divisions):

- Раздел идентификаций (IDENTIFICATION DIVISION), определяющий имя программы и, если нужно, содержащий прочую идентифицирующую информацию.
- Раздел оборудования (ENVIRONMENT DIVISION), где описываются аспекты программы, зависящие от вычислительной среды.
- Раздел данных (DATA DIVISION), где определяются свойства данных в одной из следующих секций внутри раздела данных (DATA DIVISION):
  - Секция файлов (FILE SECTION), определяющая данные, используемые в операторах ввода-вывода.
  - Секция связей (LINKAGE SECTION), описывающая данные из другой программы.

  При определении данных, предназначенных для внутренней обработки:
  - Секция рабочей памяти (WORKING-STORAGE SECTION), описывающая статически выделяемую память, используемую на протяжении существования запущенного модуля.
  - Секция локальной памяти (LOCAL-STORAGE SECTION), описывающая память, выделяемую при каждом вызове программы, и освобождаемую по завершении выполнения программы.
  - Секция связей (LINKAGE SECTION), описывающая данные из другой программы.
- Раздел процедур (PROCEDURE DIVISION), где определяются инструкции, связанные с управлением данными, и интерфейсы с другими процедурами.

Раздел процедур программы разделен на несколько секций и параграфов, содержащих предложения и операторы, как описано ниже.

- Секция – логический подраздел логики обработки. Секция содержит заголовок, за которым может следовать один или несколько параграфов. Секция может вызываться оператором PERFORM. Один из типов секций предназначен для объявления.
- Объявления (declaratives) представляют собой набор из одной или нескольких секций специального назначения, написанных в начале раздела процедур (PROCEDURE DIVISION), первая из которых предваряется ключевым словом DECLARATIVES и последняя из которых завершается ключевым словом END DECLARATIVES.
- Параграф – подраздел секции, процедуры или программы. Параграф может вызываться оператором.
- Предложение – набор из одного или нескольких операторов языка COBOL, завершающийся точкой.
- Оператор – выполняет определенный этап обработки на языке COBOL, например, сложение двух чисел.
- Фраза – подраздел оператора.

Примеры разделов в языке COBOL

Пример 9.1. Раздел идентификаций (IDENTIFICATION DIVISION)

```
IDENTIFICATION DIVISION.
Program-ID. Helloprog.
Author. A. Programmer.
Installation. Computing Laboratories.
Date-Written. 08/21/2002.
```

Пример кодирования ввода-вывода

Пример 9.2. Раздел оборудования (ENVIRONMENT DIVISION)

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. computer-name.
OBJECT-COMPUTER. computer-name.
SPECIAL-NAMES.
special-names-entries.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT [OPTIONAL] file-name-1
ASSIGN TO system-name [FOR MULTIPLE {REEL | UNIT}]
[.... .
I-O-CONTROL.
SAME [RECORD] AREA FOR file-name-1 ... file-name-n.
```

Описания пользовательской информации представлены ниже Примера 9.3.

Пример 9.3. Входные и выходные файлы в FILE-CONTROL

```
       IDENTIFICATION DIVISION.
. . .
    ENVIRONMENT DIVISION.
    INPUT-OUTPUT SECTION.
    FILE-CONTROL.
       SELECT filename ASSIGN TO assignment-name
       ORGANIZATION IS org ACCESS MODE IS access
       FILE STATUS IS file-status
    . . .
    DATA DIVISION.
    FILE SECTION.
    FD filename
    01 recordname
       nn . . . fieldlength & type
       nn . . . fieldlength & type
    . . .
    WORKING-STORAGE SECTION
    01 file-status PICTURE 99.
    . . .
    PROCEDURE DIVISION.
       . . .
       OPEN iomode filename
       . . .
       READ filename
       . . .
       WRITE recordname
       . . .
       CLOSE filename
       . . .
       STOP RUN.
```

`org` обозначает организацию и может принимать значения SEQUENTIAL, LINE SEQUENTIAL, INDEXED или RELATIVE.

`access` обозначает режим доступа и может принимать значения SEQUENTIAL, RANDOM или DYNAMIC.

`iomode` обозначает режим INPUT или OUTPUT. Если осуществляется только чтение из файла, следует использовать значение INPUT. Если выполняется только запись в файл, следует использовать значение OUTPUT или EXTEND. Если осуществляется и чтение, и запись, следует использовать значение I-O, если не используется организация LINE SEQUENTIAL.

Также задаются прочие значения, такие как `filename`, `recordname`, `fieldname` (`nn` в нашем примере), `fieldlength` и `type`.

9.4.2 Связь между JCL и программными файлами в COBOL

В Примере 9.4 представлена связь между JCL-операторами и файлами COBOL-программы. Отсутствие ссылок на физическое расположение файлов данных в программе позволяет достичь независимости от устройств. Другими словами, можно изменить расположение данных и их имя, не изменяя программу. Для этого требуется только лишь изменить JCL-код.

Пример 9.4. Связь между JCL и программными файлами в COBOL

```
//MYJOB    JOB
//STEP1    EXEC IGYWCLG
...
    INPUT-OUTPUT SECTION.
    FILE-CONTROL.
      SELECT INPUT ASSIGN TO INPUT1 .....
      SELECT DISKOUT ASSIGN TO OUTPUT1 ...
    FILE SECTION.
      FD INPUT1
        BLOCK CONTAINS...
        DATA RECORD IS RECORD-IN
      01 INPUT-RECORD
...
    FD OUTPUT1
      DATA RECORD IS RECOUT
    01 OUTPUT-RECORD
...
/*
//GO.INPUT1 DD DSN=MY.INPUT,DISP=SHR
//GO.OUTPUT1 DD DSN=MY.OUTPUT,DISP=OLD
```

В Примере 9.4 представлен поток задания, выполняющего компиляцию, связывание и запуск программы на COBOL, содержащий операторы файловой программы и соответствующие им JCL-операторы.

Операторы COBOL SELECT создают связи между DD-именами INPUT1 и OUTPUT1 и описаниями COBOL FD INPUT1 и OUTPUT1, соответственно. COBOL FD связаны с групповыми элементами INPUT-RECORD и OUTPUT-RECORD.

DD-карты INPUT1 и OUTPUT1 связаны с наборами данных MY.INPUT и MY.OUTPUT, соответственно. Конечный результат этого связывания в нашем примере состоит в том, что эти записи, считываемые из файла INPUT1, будут считываться из физического набора данных MY.INPUT, и записи, записываемые в файл OUTPUT1, будут записываться в физический набор данных MY.OUTPUT. Программа полностью независима от размещения данных и имени наборов данных.

На рис. 9.3 показана связь между физическим набором данных, JCL и программой для Примера 9.4.

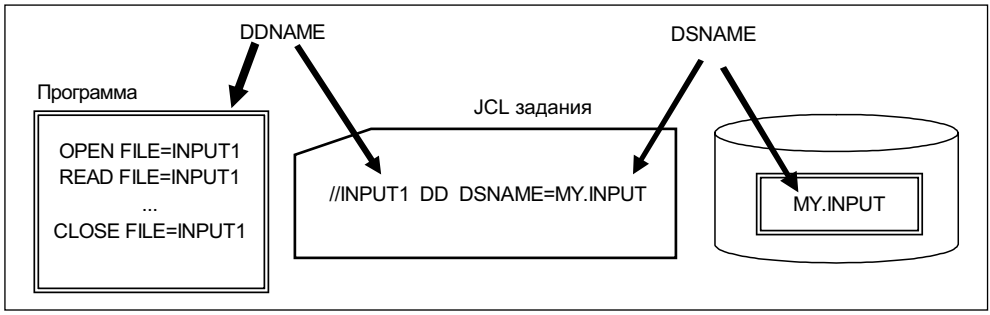


Рис. 9.3. Связь между JCL, программой и набором данных

И снова, так как программа не осуществляет каких-либо обращений к физическому набору данных, не требуется перекомпилировать программу в случае изменения имени или расположения набора данных.

9.4.3 Выполнение COBOL-программ в среде UNIX

Для запуска COBOL-программ в среде UNIX необходимо их скомпилировать с помощью компилятора Enterprise COBOL или COBOL для OS/390 и VM. Они должны быть реентерабельными, поэтому следует использовать опцию компилятора и компоновщика RENT.

9.4.4 Связь с Java-методами

Для обеспечения межъязыкового взаимодействия с Java необходимо следовать определенным правилам и указаниям по следующим аспектам:

- использование служб в интерфейсе JNI (Java Native Interface);
- кодирование типов данных;
- компиляция COBOL-программ.

Из COBOL-программ можно вызывать методы, написанные на языке Java, и из Java-программ можно вызывать методы, написанные на языке COBOL. Для простых возможностей работы с объектами в Java можно использовать объектно-ориентированный COBOL. Для дополнительных возможностей Java можно вызывать службы JNI.

Так как Java-программы могут быть многопоточными и использовать асинхронные сигналы, следует компилировать COBOL-программы с опцией THREAD.

9.4.5 Создание DLL или DLL-приложения

Динамическая библиотека (DLL) представляет собой файл, содержащий исполняемый код и данные, подключаемые к программе во время выполнения. Код и данные в DLL могут совместно использоваться несколькими приложениями одновременно. Создание DLL или DLL-приложения подобно созданию обычного COBOL-приложения. Оно включает написание, компиляцию и компоновку исходного кода.

При разработке DLL или DLL-приложения следует обращать особое внимание на следующее:

- определение того, каким образом части загрузочного модуля или приложения связаны друг с другом или с другими DLL;
- определение того, какие механизмы компоновки или вызовов использовать.

В зависимости от того, требуется ли использовать загрузочный модуль DLL или загрузочный модуль, обращающийся к отдельной DLL, нужно использовать разные опции компилятора и компоновщика.

9.4.6 Структурирование объектно-ориентированных приложений

Можно структурировать приложения, использующие синтаксис объектно-ориентированной версии COBOL, одним из трех способов. Объектно-ориентированное приложение может начинаться с:

- COBOL-программы, которая может иметь любое имя;
- определения Java-класса, содержащего метод с именем *main*. Можно запустить приложение Java-командой, указав имя класса, содержащего метод main, и ни одной или несколько строк в качестве аргументов командной строки;
- определения COBOL-класса, содержащего фабричный (factory) метод с именем *main*. Можно запустить приложение Java-командой, указав имя класса, содержащего метод main, и ни одной или несколько строк в качестве аргументов командной строки.

Дополнительные сведения. Дополнительные сведения об использовании языка COBOL в z/OS см. в руководствах *Enterprise COBOL for z/OS and OS/390 V3R2 Language Reference*, SC27-1408, и *Enterprise COBOL for z/OS and OS/390 V3R2 Programming Guide*, SC27-1412. Эти книги доступны в Интернете по адресу:

```
http://www.ibm.com/servers/eserver/zseries/zos/bkserv/find_
shelves.html
```

9.5 Связь между JCL и программными файлами в высокоуровневых языках

В разделе 9.4.2 «Связь между JCL и программными файлами в COBOL» мы рассмотрели, как изолировать COBOL-программу от изменений имени и расположения набора данных. Метод обращения к физическим файлам с использованием символического имени файла используется не только в COBOL; он используется во всех высокоуровневых языках и даже в ассемблере. Общий пример программы на высокоуровневом языке, обращающейся к наборам данных посредством символических имен файлов, см. в Примере 9.5.

Изоляция вашей программы от изменений имени или размещения набора данных является обычной целью. Однако могут возникать ситуации, когда программе требуется доступ к определенному набору данных, имеющему определенное расположение на устройстве хранения с прямым доступом (direct access storage device, DASD). Это можно выполнить в ассемблере и даже в некоторых высокоуровневых языках.

Жесткое кодирование имен наборов данных и прочей подобной информации в программе обычно не считается в программировании хорошей практикой. Жестко закодированные значения в программе могут требовать изменений, что, в свою очередь, потребует перекомпилировать программу при каждом изменении значения. Вынос этих значений за пределы программ, как в случае с обращением к наборам данных из программы с использованием символического имени, является более эффективной практикой, которая позволяет программе продолжать работу даже при изменении имени набора данных.

Пример 9.5. Связь между JCL и программными файлами в высокоуровневых языках

```
//MYJOB  JOB
//STEP1  EXEC  CLG
...
     OPEN  FILE=INPUT1
     OPEN  FILE=OUTPUT1
     READ  FILE=INPUT1
...
     WRITE  FILE=OUTPUT1
...
     CLOSE  FILE=INPUT1
     CLOSE  FILE=OUTPUT1
/*
//GO.INPUT1  DD  DSN=MY.INPUT,DISP=SHR
//GO.OUTPUT1  DD  DSN=MY.OUTPUT,DISP=OLD
```

Более подробное описание использование символического имени для обращения к файлу см. в разделе 6.5 «Почему z/OS использует символические имена файлов».

9.6 Использование PL/I в z/OS

Programming Language/I (PL/I, произносится «Пи-эЛ ван») является полнофункциональным высокоуровневым языком общего назначения, подходящим для разработки следующих типов программного обеспечения:

- коммерческие приложения;
- инженерные/научные приложения;
- многие другие приложения.

Процесс компиляции исходной программы на языке PL/I и последующей компоновки объектного модуля в загрузочный модуль практически ничем не отличается от соответствующего процесса в COBOL. См. Пример 9.2, раздел 10.3.7 «Как используется редактор связей?» и рис. 9.3.

Связь между JCL и программными файлами в PL/I осуществляется так же, как в языке COBOL и других высокоуровневых языках. См. рис. 9.3 и Пример 9.5.

9.6.1 Структура PL/I-программы

PL/I является языком с блочной структурой, состоящим из пакетов, процедур, операторов, выражений и встроенных функций, как показано на рис. 9.4.

Переменная – содержит данные, назначенные ей, до тех пор, пока не будет назначено новое значение

PL/I-программы состоят из блоков. *Блок* (*block*) может представлять подпрограмму либо просто набор операторов. PL/I-блок позволяет создавать модульные приложения, так как блоки могут содержать объявления, определяющие имена переменных и классы хранения. Таким образом, можно ограничить область действия переменной одним блоком или набором блоков, либо сделать их известными в единице компиляции или загрузочном модуле.

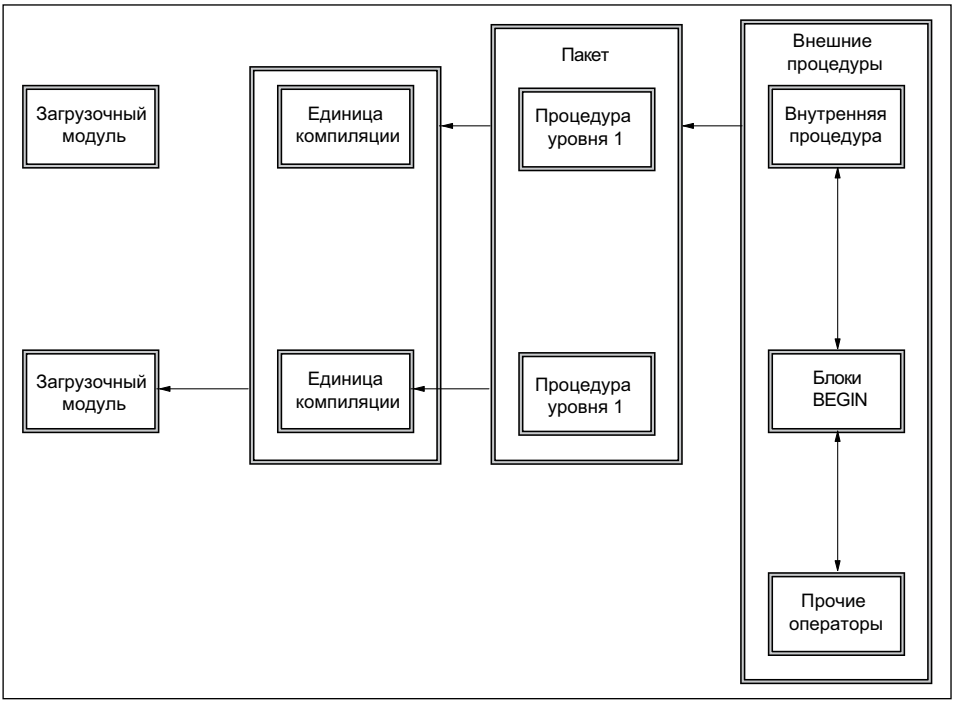


Рис. 9.4. Структура PL/I-приложения

PL/I-приложение состоит из одной или нескольких раздельно загружаемых сущностей, называемых *загрузочными модулями* (*load modules*). Каждый загрузочный модуль может содержать одну или несколько раздельно компилируемых сущностей, называемых *единицами компиляции* (*compilation units*). Если не обусловлено иное, термин *программа* относится к PL/I-приложению или единице компиляции.

Единицей компиляции является пакет PL/I или внешняя процедура. Каждый пакет может содержать ни одной или несколько процедур, некоторые или все из которых могут быть экспортированы. Внешняя или внутренняя процедура PL/I содержит ни одного или несколько блоков.

PL/I-блок представляет собой либо процедуру (PROCEDURE), либо блок BEGIN, каждый из которых содержит ни одного или несколько операторов и/или ни одного или несколько блоков. Процедура представляет собой последовательность операторов, отделенную оператором procedure и соответствующим оператором end, как показано в Примере 9.6. Процедура может представлять собой главную процедуру, подпрограмму или функцию.

Пример 9.6. Блок PROCEDURE

```
A: procedure;
     statement-1
     statement-2
     .
     .
     statement-n
     end Name;
```

Блок BEGIN представляет собой последовательность операторов, отделенную оператором begin и соответствующим оператором end, как показано в Примере 9.7. Программа завершается при завершении главной процедуры.

Пример 9.7. Блок BEGIN

```
B: begin;
     statement-1
     statement-2
     .
     .
     statement-n
     end B;
```

9.6.2 Препроцессор

Компилятор PL/I позволяет выбрать один или несколько интегрированных препроцессоров, использование которых требуется программой. Можно выбрать препроцессор включений, препроцессор макросов, препроцессор SQL или препроцессор CICS, а также выбрать порядок их вызова.

Препроцессор – программа, выполняющая некоторую предварительную обработку входных данных до их обработки главной программой

Каждый препроцессор поддерживает множество опций, позволяющих настраивать обработку под свои требования.

- Препроцессор включений (include preprocessor).
  Позволяет включать внешние исходные файлы в свои программы, используя директивы включения, отличные от PL/I-директивы %INCLUDE (директива %INCLUDE используется для включения внешнего текста в исходную программу).
- Препроцессор макросов.
  Макросы позволяют создавать распространяемый PL/I-код таким способом, при котором скрываются детали реализации и обрабатываемые данные и показываются

только операции. В отличие от обобщенной подпрограммы, макросы позволяют генерировать только код, требуемый для каждого отдельного использования.

- Препроцессор SQL.

  В целом составление PL/I-программы происходит одинаково, независимо от того, требуется ли, чтобы она осуществляла доступ к базе данных DB2. Однако для извлечения, обновления, вставки и удаления данных в DB2 и использования прочих операций DB2 необходимо использовать SQL-операторы. В PL/I-приложениях можно использовать динамические и статические операторы EXEC SQL.

  Для связи с DB2 нужно выполнить следующие действия:

  - составить требуемые SQL-операторы, отделяя их оператором EXEC SQL;
  - использовать прекомпилятор DB2 или выполнить компиляцию с опцией компилятора PL/I PP(SQL()).

  Прежде чем использовать поддержку EXEC SQL, необходимо иметь полномочия для доступа к системе DB2.

  Обратите внимание на то, что PL/I SQL Preprocessor в настоящее время не поддерживает DBCS.

- Препроцессор CICS.
  В PL/I-приложениях можно использовать операторы EXEC CICS, выполняющиеся как транзакции в CICS.

Дополнительные сведения. Дополнительные сведения об использовании языка PL/I в z/OS см. в руководствах *Enterprise PL/I for z/OS V3R3 Language Reference*, SC27-1460, и *Enterprise PL/I for z/OS V3R3 Programming Guide*, SC27-1457. Эти книги доступны в Интернете по адресу:

`http://www.ibm.com/servers/eserver/zseries/zos/bkserv/find_shelves.html`

9.6.3 Использование синтаксического анализатора SAX

Компилятор PL/I содержит интерфейс PLISAXx (x = A или B), обеспечивающий основные возможности XML. Эти средства поддержки включают высокоскоростной синтаксический анализатор XML, который позволяет принимать в программах входящие XML-сообщения, выполнять их проверку правильности и преобразовывать их содержимое в структуры данных PL/I.

Средства поддержки XML не обеспечивают генерирование XML, которое должно выполняться программной логикой PL/I. Средства поддержки XML не имеют особых требований к окружению. Они работают во всех основных средах выполнения, включая CICS, IMS и MQ Series, а также пакетную обработку в z/OS и TSO.

9.7 Использование C/C++ в z/OS

C представляет собой язык программирования, предназначенный для широкого диапазона целей программирования, включая:

- составление кода системного уровня;
- обработку текстов;
- графику.

Язык C содержит краткий набор операторов и библиотеку, обеспечивающую дополнительные функциональные возможности. Такое разделение обеспечивает гибкость и эффективность языка C. Дополнительное преимущество состоит в высокой согласованности этого языка на различных системах.

Процесс компиляции исходной программы на языке C и последующей компоновки объектного модуля в загрузочный практически ничем не отличается от соответствующего процесса в COBOL. См. Пример 9.2, раздел 10.3.7 «Как используется редактор связей?» и рис. 9.3.

Связь между JCL и программными файлами осуществляется так же, как в языке COBOL и других высокоуровневых языках. См. рис. 9.3 и Пример 9.5.

Дополнительные сведения. Дополнительные сведения об использовании языков C и C++ в z/OS см. в руководствах *C/C++ Language Reference*, SC09-4764, и *C/C++ Programming Guide*, SC09-4765. Эти книги доступны в Интернете по адресу:

`http://www.ibm.com/servers/eserver/zseries/zos/bkserv/find_shelves.html`

9.8 Использование Java в z/OS

Java представляет собой объектно-ориентированный язык программирования, разработанный компанией Sun™ Microsystems™ Inc. Java можно использовать как для разработки традиционных коммерческих мэйнфрейм-приложений, так и для разработки приложений для Интернета и интрасетей, использующих стандартные интерфейсы.

Java представляет собой все более популярный язык программирования, используемый для разработки большого числа приложений в различных операционных системах. IBM поддерживает и использует Java на всех вычислительных платформах IBM, включая z/OS. В z/OS Java-продукты обеспечивают такие же полнофункциональные Java API, как и на всех других платформах IBM. Кроме того, лицензированные Java-программы для z/OS были усовершенствованы таким образом, чтобы осуществлять доступ из Java к уникальным файловым системам z/OS. Такие языки программирования, как Enterprise COBOL и Enterprise PL/I в z/OS содержат интерфейсы для программ, написанных на языке Java. Эти языки содержат набор интерфейсов или средств взаимодействия с программами, написанными на языке Java, как описывалось для языка COBOL в разделе 9.4.4 «Связь с Java-методами» и для языка PL/I в разделе 9.6.3 «Использование синтаксического анализатора SAX».

Различные лицензированные программы Java Software Development Kit (SDK) для z/OS позволяют разработчикам приложений использовать Java API для z/OS, создавать или запускать приложения на различных платформах и использовать Java для доступа к данным, расположенным на мэйнфрейме. Некоторые из этих продуктов позволяют Java-приложениям выполняться только в среде 31-разрядной адресации. Однако использование 64-разрядных SDK для z/OS позволяет Java-приложениям, которые ранее были ограничены 31-разрядной адресацией, выполняться в 64-разрядной среде. Кроме того, некоторые мэйнфреймы поддерживают специальный процессор zAAP (zSeries Application Assist Processor) для выполнения Java-приложений. Программы можно запускать интерактивно через z/OS UNIX или в пакете.

9.8.1 Продукты IBM SDK для z/OS

Как и в случаях с Java SDK на других платформах IBM, лицензированные программы z/OS Java SDK разрабатываются для поддержки API промышленного стандарта. Продукты z/OS SDK независимы друг от друга и могут заказываться и обслуживаться отдельно.

На момент написания этой книги были доступны следующие Java SDK для z/OS:

- Продукт Java SDK1.3.1 под названием IBM Developer Kit for OS/390, Java 2 Technology Edition работает в z/OS, а также в более ранней OS/390. Он представляет собой 31-разрядный продукт. Многие клиенты z/OS выполнили перенос (или *миграцию*) своих Java-приложений на более поздние версии Java.
- IBM SDK for z/OS, Java 2 Technology Edition, Version 1.4 представляет собой 31-разрядный вариант продукта Sun Microsystems Java Software Development Kit (SDK) для платформы z/OS и он сертифицирован как полностью совместимый Java-продукт. IBM успешно прошла тесты Java Certification Kit (JCK) 1.4, предоставленный компанией Sun Microsystems, Inc.
- IBM SDK for z/OS, Java 2 Technology Edition, Version 1.4 работает в z/OS Version 1 Release 4 или более поздней версии или в z/OS.e Version 1 Release 4 или более поздней версии. Он является аналогом среды выполнения Java, доступной на любой другой серверной платформе.
- IBM 64-bit SDK for z/OS, Java 2 Technology Edition, Version 1.4 позволяет Java-приложениям выполняться в 64-разрядной среде. Он работает в z/OS Version 1 Release 6 или более поздней версии. Как и 31-разрядная версия продукта, этот продукт позволяет использовать Java SDK1.4 API.

Дополнительные сведения о продуктах Java SDK для z/OS см. в Интернете по адресу

`http://www.ibm.com/servers/eserver/zseries/software/java/`

9.8.2 Использование Java Native Interface (JNI)

Java Native Interface (JNI) представляет собой Java-интерфейс к традиционным языкам программирования и входит в JDK. Если стандартные Java API не содержат нужной вам функции, JNI позволяет Java-коду, выполняющемуся в JVM (Java Virtual Machine) взаимодействовать с приложениями и библиотеками, написанными на других языках, например на PL/I. Кроме того, Invocation API позволяет встраивать JVM в PL/I-приложения.

Java является достаточно полным языком программирования; однако бывают ситуации, когда требуется вызвать программу, написанную на другом языке программирования. В Java для этого используется вызов метода на традиционном языке, называемого *собственным методом* (*native method*). Программирование с использованием JNI позволяет использовать собственные методы для выполнения множества различных операций. Собственный метод позволяет:

- использовать Java-объекты так же, как их использует Java-метод;
- создавать Java-объекты, включая массивы и строки, и затем просматривать и использовать эти объекты для выполнения различных задач;

- просматривать и использовать объекты, созданные кодом Java-приложения;
- изменять Java-объекты, созданные им или переданные ему; эти измененные объекты затем можно сделать доступными для Java-приложения.

И, наконец, собственные методы могут с легкостью вызывать уже существующие Java-методы, используя функциональные возможности, уже реализованные в Java-инфраструктуре. Таким образом, и часть приложения, написанная на трациционном языке, и часть приложения, написанная на Java, могут создавать, изменять и осуществлять доступ к Java-объектам, и затем совместно использовать эти объекты.

9.9 Использование CLIST в z/OS

Язык CLIST является интерпретируемым языком. Подобно программам на других высокоуровневых интерпретируемых языках, CLIST-программы просты в написании и тестировании. Их не требуется компилировать или компоновать. Для тестирования CLIST-программы следует ее просто запустить и исправить возникающие ошибки и так до тех пор, пока программа не отработает без ошибок.

Языки CLIST и REXX представляют собой два командных языка, доступных в TSO/E. Язык CLIST позволяет более эффективно работать с TSO/E.

Название CLIST (произносится «си-лист») означает *command list* (*командный список*); такое название связано с тем, что наиболее простые CLIST-программы представляют собой списки команд TSO/E. При вызове такой CLIST-программы она вызывает команды TSO/E по порядку.

Язык программирования CLIST используется для следующих целей:

- выполнение типовых задач (таких как ввод команд TSO/E);
- вызов других CLIST-программ;
- вызов приложений, написанных на других языках;
- ISPF-приложения (в частности, вывод панелей и управление выполнением приложений).

9.9.1 Типы CLIST-программ

CLIST-программа может выполнять широкий диапазон задач, большинство из которых попадает в одну из трех общих категорий:

- CLIST-программы, выполняющие типовые задачи;
- CLIST-программы, представляющие собой структурированные приложения;
- CLIST-программы, управляющие приложениями, написанными на других языках.

Эти типы программ описываются в этом разделе.

CLIST-программы, выполняющие типовые задачи

Как пользователю TSO/E, вам, скорее всего, придется регулярно выполнять определенные задачи. К таким задачам может относиться ввод команд TSO/E для проверки состояния наборов данных, распределение наборов данных для определенных программ или для распечатывания файлов.

Можно написать CLIST-программы, позволяющие значительно сократить время, требуемое для выполнения этих типовых задач. Сгруппировав все инструкции, требуемые для выполнения задачи в CLIST-программу, можно сократить время, количество нажатий клавиш и количество ошибок при выполнении задачи, что позволяет повысить вашу продуктивность. CLIST-программа может содержать только команды TSO/E или же сочетание команд TSO/E и операторов CLIST.

CLIST-программы, представляющие собой структурированные приложения

Язык CLIST содержит основные инструменты, необходимые для составления полных и структурированных приложений. Любая CLIST-программа может вызвать другую CLIST-программу, называемую *вложенной CLIST-программой* (*nested CLIST*). CLIST-программы могут также содержать отдельные подпрограммы, называемые *подпроцедурами* (*sub-procedures*). Вложенные CLIST-программы и подпроцедуры позволяют разделить CLIST-программы на логические модули и поместить общие функции в единое расположение. Некоторые операторы CLIST позволяют:

- определять общие данные для подпроцедур и вложенных CLIST-программ;
- определять данные, предназначенные для определенных подпроцедур и CLIST-программ;
- передавать определенные данные в подпроцедуру или вложенную CLIST-программу.

В интерактивных приложениях CLIST-программы могут выдавать ISPF-команды для вывода полноэкранных панелей. И наоборот, ISPF-панели могут вызывать CLIST-программы на основе данных, введенных пользователем в панели.

CLIST-программы, управляющие приложениями, написанными на других языках

Предположим, что у вас есть доступ к приложениям, написанным на других языках программирования, однако интерфейсы для этих приложений могут быть сложны в использовании или запоминании. Вместо того, чтобы писать новые приложения, можно написать CLIST-программы, реализующие простые интерфейсы между пользователем и такими приложениями.

CLIST-программа может передавать и принимать сообщения с терминала, позволяющие определить, что нужно пользователю. Затем на основе этой информации CLIST-программа может создать среду и выдавать требуемые команды для вызова программы, выполняющей запрашиваемые задачи.

9.9.2 Выполнение CLIST-программ

Для выполнения CLIST-программы используется команда EXEC. В командной строке ISPF вводится `TSO` перед началом команды. В TSO/E в режиме EDIT или TEST следует использовать подкоманду EXEC, как если бы использовалась команда EXEC (CLIST-программы, выполняемые в режиме EDIT или TEST, могут использовать только подкоманды EDIT или TEST и операторы CLIST, однако можно использовать подкоманду END в CLIST-программе для выхода из режима EDIT или TEST, что позволит CLIST-программе использовать команды TSO/E).

9.9.3 Другие варианты использования языка CLIST

Помимо ввода команд TSO/E, CLIST-программы могут выполнять более сложные задачи программирования. Язык CLIST содержит инструменты программирования, необходимые для написания больших структурированных приложений. CLIST-программы могут выполнять любые сложные задачи, от вывода набора полноэкранных панелей до управления программами, написанными на других языках.

Язык CLIST включает следующие возможности:

- широкий набор арифметических и логических операторов для обработки численных данных;
- функции работы со строками, осуществляющие обработку символьных данных;
- операторы CLIST, позволяющие структурировать программы, осуществлять ввод-вывод, определять и изменять переменные и обрабатывать ошибки и прерывания для привлечения внимания (attention interrupts).

9.10 Использование REXX в z/OS

Язык REXX (Restructured Extended Executor) представляет собой процедурный язык, позволяющий писать программы и алгоритмы четким и структурированным образом. Он является и интерпретируемым, и компилируемым языком. Отличие интерпретируемого языка от других языков программирования, таких как COBOL, состоит в том, что в нем не нужно компилировать командный список REXX перед его выполнением. Однако для сокращения времени обработки можно компилировать командный список REXX перед его выполнением.

Язык программирования REXX обычно используется для следующих целей:

- Выполнение типовых задач, таких как ввод команд TSO/E.
- Вызов других исполняемых программ REXX.
- Вызов приложений, написанных на других языках.
- ISPF-приложения (вывод панелей и управление выполнением приложений).
- Быстрое однократное разрешение проблем.
- Системное программирование.
- Все, для чего можно использовать другой высокоуровневый компилируемый язык.

REXX также используется в среде Java, например, диалект REXX под названием NetRexx™ напрямую работает с Java. NetRexx-программы могут непосредственно использовать любые Java-классы, а также могут использоваться при написании любого Java-класса. Это привносит безопасность и быстродействие, свойственные Java, в REXX-программы, а также арифметику и простоту REXX в Java. Таким образом, один язык NetRexx может использоваться как для написания скриптов, так и для разработки приложений.

Структура REXX-программы довольно проста. Она содержит стандартный набор управляющих конструкций. Это, в частности, включает IF... THEN... ELSE... для простой условной обработки, SELECT... WHEN... OTHERWISE... END для выбора из различных

вариантов и несколько вариантов DO... END для группировки и циклов. Отсутствует инструкция GOTO, однако существует инструкция SIGNAL для аварийной передачи управления, например, при возникновении ошибок и вычисляемых переходах .

Связь между JCL и программными файлами в REXX осуществляется так же, как в COBOL и других высокоуровневых языках. См. рис. 9.3 и Пример 9.5.

9.10.1 Компиляция и выполнение командных списков REXX

REXX-программа, скомпилированная в среде z/OS, может выполняться в среде z/VM. Подобным образом REXX-программа, скомпилированная в среде z/VM, может выполняться в среде z/OS. REXX-программа, скомпилированная в среде z/OS или z/VM, может выполняться в среде z/VSE, если установлен REXX/VSE.

Процесс компиляции исходной программы на языке REXX и последующей компоновки объектного модуля в загрузочный модуль практически ничем не отличается от соответствующего процесса в COBOL. См. Пример 9.2, раздел 10.3.7 «Как используется редактор связей?» и рис. 9.3.

Существует три основных компонента языка REXX, используемых при компиляции:

- IBM Compiler for REXX on zSeries. Компилятор преобразует исходные программы REXX в компилированные программы.
- IBM Library for REXX on zSeries. Библиотека содержит подпрограммы, вызываемые компилированными программами во время выполнения.
- Alternate Library. Библиотека Alternate Library содержит языковой обработчик, преобразующий компилированные программы и вызывающий их из интерпретатора. Она может использоваться пользователями z/OS и z/VM, не имеющими IBM Library for REXX on zSeries для запуска компилированных программ.

Компилятор и библиотека работают на системах z/OS с TSO/E и на системах z/VM с CMS. IBM Library for REXX в REXX/VSE работает под управлением z/VSE.

Компилятор может генерировать выходные данные в следующих форматах:

- Компилированные EXEC-файлы

  Такие программы выполняются точно так же, как и интерпретируемые REXX-программы. Они таким же образом вызываются системным обработчиком EXEC-файлов и имеют такую же последовательность поиска. Простейший способ замены интерпретируемых программ компилированными программами состоит в генерировании компилированных EXEC-файлов. Пользователям необязательно знать, являются ли используемые ими REXX-программы компилированными EXEC-файлами или интерпретируемыми программами. Компилированные EXEC-файлы могут быть переданы в z/VSE и выполняться там.
- Объектные модули под управлением z/OS или TEXT-файлы под управлением z/VM.
- TEXT-файл, представляющий собой файл объектного кода с неразрешенными внешними обращениями (этот термин используется только в z/VM). Такие файлы необходимо преобразовать в исполняемый формат (загрузочные модули), прежде чем их можно будет использовать. Загрузочные модули и MODULE-файлы вызываются таким же образом, как и загрузочные модули, сгенериро-

ванные другими компиляторами, и имеют такую же последовательность поиска. Однако последовательность поиска отличается от последовательности поиска интерпретируемых REXX-программ и компилированных EXEC-файлов. Эти загрузочные модули могут использоваться как команды или как части пакетов функций REXX. Объектные модули или MODULE-файлы могут передаваться в z/VSE на компоновку .

- IEXEC-вывод.

  Эти выходные данные содержат расширенный источник компилируемой REXX-программы. Под расширенным подразумевается, что главная программа и все части, включаемые во время компиляции директивой %INCLUDE, находятся в IEXEC-выводе. IEXEC-вывод содержит только текст в определенных границах. Обратите внимание, однако, на то, что заданное по умолчанию значение MARGINS включает весь текст во входных записях.

Дополнительные сведения. Дополнительные сведения о REXX см. в следующих публикациях:

- *The REXX Language*, 2nd Ed., Cowlishaw, ZB35-5100
- *Procedures Language Reference (Level 1)*, C26-4358 SAA® CPI
- *REXX on zSeries V1R4.0 User's Guide and Reference*, SH19-8160
- *Creating Java Applications Using NetRexx* , SG24-2216

Также рекомендуем посетить веб-сайт:

`http://www.ibm.com/software/awdtools/REXX/language/REXXlinks.html`

9.11 Компилируемые и интерпретируемые языки

Во время проектирования приложения может потребоваться решить, какой язык использовать для написания исходного кода приложения: компилируемый или интерпретируемый. Языки обоих типов имеют свои преимущества и недостатки. Обычно решение использовать интерпретируемый язык принимается по причине ограничений времени разработки или из-за простоты последующего внесения изменений в программу. Использование интерпретируемого языка является компромиссным вариантом. При этом уменьшается время разработки, однако увеличивается стоимость выполнения. Так как каждая строка интерпретируемой программы транслируется при каждом ее выполнении, возникают высокие непроизводительные издержки. Таким образом, интерпретируемый язык в целом больше подходит для незапланированных, чем для предопределенных запросов.

9.11.1 Преимущества компилируемых языков

Ассемблер, COBOL, PL/I, C/C++ транслируются путем прогона исходного кода через компилятор. Это генерирует очень эффективный код, который можно многократно выполнять. Выполнение трансляции происходит только один раз при компиляции исходного кода; после этого требуется только лишь загрузить и выполнить программу.

В интерпретируемых языках, с другой стороны, синтаксический анализ, интерпретация и выполнение выполняется при каждом выполнении программы, что зна-

чительно увеличивает стоимость выполнения программы. По этой причине интерпретируемые программы обычно менее эффективны, чем компилируемые.

Некоторые языки программирования, такие как REXX и Java, допускают как интерпретацию, так и компиляцию.

9.11.2 Преимущества интерпретируемых языков

В разделе «Преимущества компилируемых языков» мы рассмотрели основания для использования компилируемых языков. В разделах «Использование языка CLIST в z/OS» и «Использование языка REXX в z/OS» обсуждались сильные стороны интерпретируемых языков. Не существует ответа на вопрос, какой язык «лучше» – все зависит от приложения. Даже в одном приложении может использоваться несколько различных языков. Например, одно из преимуществ такого языка как CLIST состоит в простоте кодирования, тестирования и внесения изменений. Однако он не очень эффективен. В данном случае компромисс состоит в увеличении требуемых машинных ресурсов взамен уменьшения времени разработки.

Учитывая это, мы можем увидеть, что имеет смысл использовать компилируемый язык при написании интенсивных частей приложения (с высоким показателем использования ресурсов), тогда как интерфейсы (вызывающие приложение) и менее интенсивные части можно написать на интерпретируемом языке. Интерпретируемый язык может хорошо подходить для незапланированных запросов или даже для разработки прототипа приложения.

Одна из задач проектировщика состоит в том, чтобы сопоставить преимущества и недостатки каждого языка и решить, какой язык лучше использовать для той или иной части приложения.

9.12 Что такое z/OS Language Environment?

Как говорилось в главе 8 «Проектирование и разработка приложений для z/OS», приложение представляет собой набор из одной или нескольких программ, взаимодействующих друг с другом для достижения определенных целей, таких как управление запасами или ведение ведомостей. Цели разработки приложений включают модульность и совместное использование кода, а также разработку приложений на рабочей станции.

В z/OS Language Environment обеспечивает общую среду для всех соответствующих высокоуровневых языков. Высокоуровневый язык представляет собой язык программирования более высокого уровня, чем ассемблер, но более низкого уровня, чем в генераторах программ и языки запросов. z/OS Language Environment обеспечивает общую языковую среду разработки и выполнения для программистов приложений в z/OS. Если функции были предварительно реализованы в отдельных языковых продуктах, Language Environment устраняет необходимость поддерживать отдельные языковые библиотеки.

В прошлом языки программирования имели ограниченные возможности обращения друг к другу и согласованной работы в различных операционных системах. Это ограничивало возможности использования нескольких языков в приложении.

Языки программирования имели различные правила реализации структур данных и обработки ситуаций, а также взаимодействия с системными службами и библиотечными подпрограммами.

В Language Environment благодаря возможности обращения из одного языка в другой, программисты приложений для z/OS могут использовать функции и возможности в каждом языке.

9.12.1 Как используется Language Environment

Language Environment создает общую среду выполнения для всех используемых высокоуровневых языков. Объединяются основные службы времени выполнения, в частности подпрограммы обработки сообщений времени выполнения, обработки ситуаций и управления памятью. Доступ к этим службам осуществляется через набор интерфейсов, согласованных между языками программирования. Приложение может либо вызывать эти интерфейсы напрямую, либо использовать внутренние службы языка, вызывающие интерфейсы.

Language Environment позволяет использовать одну среду выполнения для приложений, независимо от языка программирования приложения и его требований к системным ресурсам.

На рис. 9.5 представлены компоненты Language Environment, в том числе:

- Основные подпрограммы, поддерживающие запуск и остановку программ, выделение памяти, связь с программами, написанными на других языках, а также индикацию и обработку ситуаций.
- Общие библиотечные службы, в частности математические службы и службы времени, часто требуемые программам, запускаемым в системе. Эти функции поддерживаются через библиотеку вызываемых служб.
- Фрагменты библиотеки времени выполнения, ориентированные на определенный язык.

Рис. 9.5. Компоненты z/OS Language Environment

Language Environment представляет собой обязательную среду выполнения для приложений, сгенерированных следующими компиляторами IBM:

- z/OS C/C++;
- C/C++ Compiler for z/OS;
- AD/Cycle® C/370™ Compiler;

- VisualAge® for Java, Enterprise Edition for OS/390;
- Enterprise COBOL for z/OS and OS/390;
- COBOL for z/OS;
- Enterprise PL/I for z/OS and OS/390;
- PL/I for MVS and VM (прежде назывался AD/Cycle PL/I for MVS and VM);
- VS FORTRAN и FORTRAN IV (в режиме совместимости).

Во многих случаях можно запускать компилированный код, сгенерированный предыдущими версиями вышеперечисленных компиляторов. Также существует набор макросов ассемблера, позволяющих ассемблерным подпрограммам выполняться в Language Environment.

9.12.2 Подробнее о Language Environment

Компоненты Language Environment, ориентированные на определенный язык, обеспечивают языковые интерфейсы и определенные службы, поддерживаемые для каждого отдельного языка, которые могут вызываться через общий интерфейс вызовов. В этом разделе некоторые из этих интерфейсов и служб описываются более подробно.

На рис. 9.6 представлена общая среда выполнения, создаваемая Language Environment.

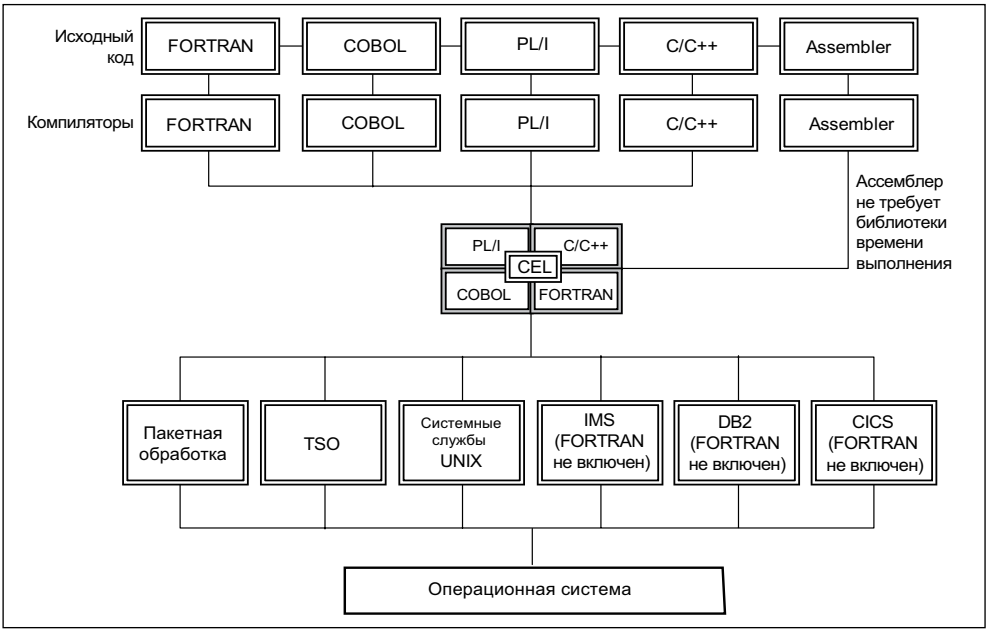


Рис 9.6. Общая среда выполнения Language Environment

Архитектура среды Language Environment построена на следующих моделях:

- модель управления программой;
- модель обработки ситуаций;
- модель служб сообщений;
- модель управления памятью.

Модель управления программой

Модель управления программой Language Environment обеспечивает инфраструктуру, в которой выполняется приложение. Она является основой для всех моделей компонентов (модели обработки ситуаций, модели обработки сообщений времени выполнения и модели управления памятью), составляющих архитектуру Language Environment.

Модель управления программой определяет влияние семантики языка программирования в приложениях на разных языках, интегрируя обработку транзакций и многопоточную обработку.

Некоторые термины, используемые для описания модели управления программой являются общими терминами программирования; другие термины в других языках имеют другое значение. Важно понимать значения терминов в контексте Language Environment в сравнении с другими контекстами.

Управление программой

Управление программой определяет используемые в приложении конструкции выполнения программы, а также семантику, связанную с интеграцией различных компонентов управления такими конструкциями.

В основе модели управления программой в Language Environment лежат три сущности: *процесс* (*process*), *анклав* (*enclave*) и *поток* (*thread*).

Процессы

Компонентом наивысшего уровня в модели программы Language Environment является *процесс* (*process*). Процесс состоит по меньшей мере из одного анклава и логически отделен от других процессов. Language Environment обычно не допускает совместного использования файлов между анклавами и не обеспечивает возможность доступа к наборам внешних данных.

Анклавы

Основной сущностью в модели управления программой является *анклав* (*enclave*), представляющий собой набор подпрограмм, составляющих приложение. Анклав является аналогом следующих понятий:

- выполняемого модуля в COBOL;
- программы, содержащей функцию main и ее подфункции в C и C++;
- процедуры main и всех ее подпрограмм в PL/I;
- программы и ее подпрограммы в Fortran.

В Language Environment под «средой» обычно понимается среда выполнения высокоуровневого языка на уровне анклава. Анклав содержит одну процедуру main и ни одной или несколько подпрограмм. Процедура main является первым выполняемым блоком в анклаве; все остальные процедуры называются подпрограммами.

Потоки

Каждый анклав содержит по меньшей мере один *поток* (*thread*), представляющий простейшую сущность определенной подпрограммы. Поток создается во время инициализации анклава и имеет собственный стек времени выполнения, используемый

при выполнении потока, а также собственный счетчик инструкций, регистры и механизмы обработки ситуаций. Каждый поток представляет независимый экземпляр подпрограммы, работающий в рамках ресурсов анклава.

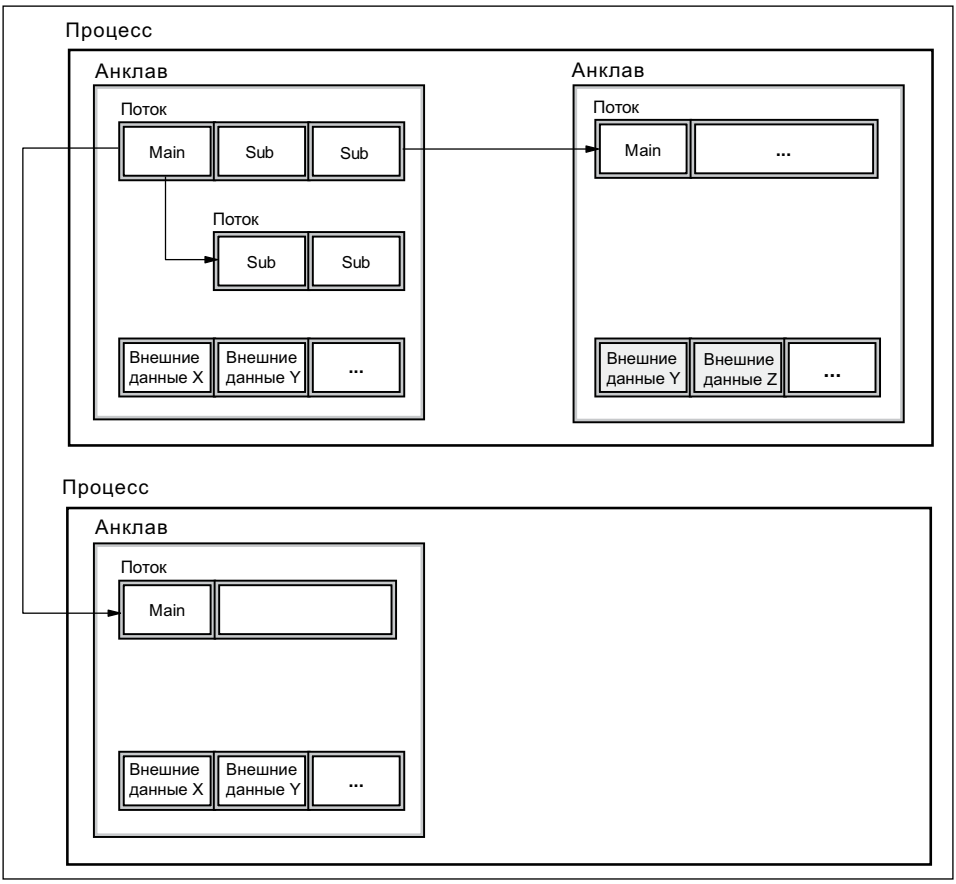


Рис. 9.7. Полная модель программ Language Environment

На рис. 9.7 представлена полная модель программ Language Environment со множеством процессов, анклавов и потоков. Как показано на рисунке, каждый процесс находится в собственном адресном пространстве. Анклав содержит одну процедуру main и некоторое количество подпрограмм.

Потоки могут создавать анклавы, которые могут создавать другие потоки, и т. д.

Модель обработки ситуаций

Для одноязычных и многоязычных приложений библиотека времени выполнения Language Environment предоставляет согласованное и предсказуемое средство обработки ситуаций. Оно не призвано заменить текущую обработку ситуаций в высокоуровневых языках, а позволяет каждому языку реагировать на ситуации в собственной среде, а также на ситуации в многоязычной среде.

Управление ситуациями в Language Environment обеспечивает гибкость непосредственного реагирования на ситуации с помощью вызываемых служб, сигнализирующих о ситуациях и запрашивающих информацию об этих ситуациях. Также оно содержит функции диагностики ошибок, генерирования отчетов об ошибках и восстановления после ошибок.

Модель обработки сообщений и поддержка национальных языков

Language Environment содержит набор общих служб обработки сообщений, создающих и передающих информационные и диагностические сообщения времени выполнения. Используя службы обработки сообщений, можно использовать токен ситуации, возвращаемый из вызываемой службы или из другой сигнализируемой ситуации, преобразовывать его в сообщение и передавать в заданное устройство вывода или в буфер. Вызываемые службы поддержки национальных языков позволяют задать национальный язык для вывода сообщений об ошибках, а также названия дней и месяцев. Они также позволяют изменять параметры страны, что влияет на формат даты, формат времени, символ валюты, символ разделителя между целой и дробной частями и разделитель тысяч.

Модель управления памятью

Существуют общие службы управления памятью для всех языков программирования, соответствующих Language Environment; среда Language Environment контролирует стек и кучу (heap), используемые во время выполнения. Она позволяет одноязычным и многоязычным приложениям осуществлять доступ к центральному набору средств управления памятью и реализует модель памяти с несколькими кучами для языков, не обеспечивающих их. Общая модель памяти устраняет необходимость поддержки каждым языком отдельного менеджера памяти и позволяет избежать несовместимости между различными механизмами хранения.

9.12.3 Выполнение программы в Language Environment

После компиляции программы можно выполнить следующие действия:

- скомпоновать и запустить существующий объектный модуль и принять заданные по умолчанию опции времени выполнения Language Environment;
- скомпоновать и запустить существующий объектный модуль и задать новые опции времени выполнения Language Environment;
- вызвать службу Language Environment.

Применение заданных по умолчанию опций времени выполнения

Для запуска существующего объектного модуля в пакетной обработке и применения всех заданных по умолчанию опций времени выполнения Language Environment можно использовать каталогизированную процедуру компоновки и запуска CEEWLG, входящую в Language Environment (каталогизированные процедуры обсуждаются в разделе 6.7 «JCL-процедуры (PROC)». Процедура CEEWLG идентифицирует библиотеки Language Environment, которые для объектного модуля требуется скомпоновать и запустить.

Службы библиотеки времени выполнения

Библиотеки Language Environment находятся в наборах данных, идентифицируемых старшим квалификатором, характерным для инсталляции. Например, SCEERUN содержит подпрограммы библиотеки времени выполнения, необходимые при выполнении приложений, написанных на языках C/C++, PL/I, COBOL и FORTRAN. SCEERUN2 содержит подпрограммы библиотеки времени выполнения, необходимые при выполнении приложений, написанных на языках C/C++ и COBOL.

Приложения, которым требуется библиотека времени выполнения, обеспечиваемая средой Language Environment, могут осуществлять доступ к наборам данных SCEERUN и SCEERUN2 с использованием одного или обоих нижеперечисленных методов:

- LNKLST;
- STEPLIB.

> **Важно!** Подпрограммы библиотеки Language Environment разделены на две категории: резидентные подпрограммы и динамические подпрограммы. Резидентные подпрограммы связаны с приложением и содержат подпрограммы инициализации и завершения процесса, а также указатели на вызываемые службы. Динамические подпрограммы не являются частью приложения и динамически загружаются во время выполнения.

Существуют некоторые аспекты, о которых нужно знать, прежде чем выполнять компоновку и запуск приложений в Language Environment.

Вызываемые службы Language Environment

Согласованные службы обработки ситуаций, реализованные в Language Environment, будут особенно полезны разработчикам приложений на языке COBOL. Для всех языков то же относится и к общим математическим службам, равно как и к службам даты-времени.

Вызываемые службы Language Environment разделены на следующие группы:

- Службы ситуаций связи.
- Службы обработки ситуаций.
- Службы даты-времени.
- Службы динамической памяти.
- Общие вызываемые службы.
- Службы инициализации и завершения процессов.
- Вызываемые службы региональных параметров.
- Математические службы.
- Службы обработки сообщений.
- Службы поддержки национальных языков.

Дополнительные сведения. Вызываемые службы более подробно описываются в публикации *z/OS Language Environment Programming Reference*, SA22-7562.

Соглашения вызовов в Language Environment

Службы Language Environment могут вызываться подпрограммами библиотек высокоуровневых языков, другими службами Language Environment и пользовательскими вызовами на высокоуровневых языках. Во многих случаях службы вызываются подпрограммами библиотек высокоуровневых языков в результате работы пользовательской функции. Ниже представлены примеры вызовов вызываемой математической службы из трех языков, рассматривавшихся в этой главе. Также см. примеры в разделе 9.9.3 «Другие варианты использования языка CLIST».

В Примере 9.8 представлен вызов вызываемых математических служб CEESDLG1 для логарифма по основанию10 из COBOL-программы.

Пример 9.8. Образец вызова математической вызываемой службы из COBOL-программы

```
77     ARG1RL COMP-2.
77     FBCODE PIC X(12).
77     RESLTRL COMP-2.
       CALL «CEESDLG1» USING ARG1RL , FBCODE ,
       RESLTRL.
```

9.13 Заключение

В этой главе рассматриваются решения, которые вам, возможно, придется принимать при проектировании и разработке приложения для z/OS. Выбор языка программирования является важным шагом на этапе проектирования приложения. Проектировщик приложения должен иметь представление о преимуществах и недостатках каждого языка, чтобы сделать наилучший выбор, исходя из требований приложения.

Критическим фактором при выборе языка является определение того, какой из языков является наиболее используемым на данной инсталляции. Если в большинстве приложений инсталляции используется COBOL, скорее всего этот язык будет выбран и для новых приложений этой инсталляции.

Важно понимать, что даже если основной язык уже выбран, это не означает, что вы должны привязываться к этому языку при написании всех программ в приложении. В некоторых случаях можно применять несколько языков, что позволяет использовать преимущества того или иного языка в различных частях приложения. В частности, может иметь смысл написать часто вызываемые подпрограммы на ассемблере, чтобы максимально увеличить эффективность приложения, даже если остальная часть приложения написана на языке COBOL или другом высокоуровневом языке.

Многие z/OS-узлы содержат библиотеку подпрограмм, совместно используемых на предприятии. Библиотека, например, может включать подпрограммы преобразования дат. Если эти подпрограммы написаны в соответствии со стандартными соглашениями компоновки, их можно вызывать из других языков, независимо от того, на каком языке написаны подпрограммы.

Каждый язык имеет свойственные ему преимущества, и проектировщики должны использовать эти преимущества. Если приложение заслуживает того, чтобы пойти на

дополнительные сложности и написать его с использованием нескольких языков, проектировщику следует использовать преимущества возможностей каждого языка. Однако следует помнить, что когда придет время вносить изменения в приложения, другие люди также должны будут уметь программировать на этих языках. Это является важнейшим правилом программирования. Первый программист мог уже давным-давно уйти из компании, а приложению еще жить и жить.

Таким образом, сложность в проектировании должна всегда рассматриваться с точки зрения простоты обслуживания.

Основные термины в этой главе

Ассемблер	Компоновщик	Компилятор
Отладка	Динамическая библиотека	Генерирование
Ввод-вывод	Интерпретатор	Загрузочные модули
Препроцессор	Язык программирования	Переменная

9.14 Контрольные вопросы

Чтобы проверить понимание материала этой главы, ответьте на следующие вопросы:

1. Почему некоторые программы пишут на ассемблере?
2. Продолжают ли компании совершенствовать компиляторы для COBOL и PL/I?
3. Почему CLIST и REXX называются интерпретируемыми языками?
4. Каковы основные сферы применения CLIST и REXX?
5. Какой интерпретируемый язык также поддерживает компиляцию?
6. Является ли использование Language Environment обязательным при разработке приложений для z/OS?
7. Какой из вариантов организации файлов данных является приемлемым для оперативных приложений? Какой приемлем для пакетных приложений?
8. Что такое высокоуровневый язык? Какие преимущества составления программ на высокоуровневом языке в сравнении с ассемблером?
9. Предположим, что программа PROG1 запущена с использованием следующего JCL-кода:

```
//job JOB
//STEP010 EXEC PGM=PROG1
//STEPLIB DD DSN=MY.PROGLIB,DISP=SHR
//INPUT1 DD DSN=A.B.C,DISP=SHR
//OUTPUT1 DD DSN=X.Y.Z,DISP=SHR
```

Если изменить карту INPUT1 DD таким образом, чтобы использовался набор данных A1.B1.C1, смогли бы мы использовать ту же программу для его обработки? Предполагается, что новый набор имеет такие же характеристики, что и старый набор данных.

9.15 Темы для дальнейшего обсуждения

1. Если важно обеспечить высокую производительность, какой язык следует использовать для написания программы: компилируемый или интерпретируемый?
2. Если требуется разработать систему обработки транзакций, какой из следующих языков будет наилучшим вариантом выбора?
   а) COBOL или PL/I на CICS.
   б) C/C++ на CICS.
   в) Комбинация из вышеперечисленного.
3. Какой язык вы бы использовали для написания приложения, вычисляющего премии в страховой политике? Предположим, что приложение будет вызываться множеством других приложений.
4. Может ли COBOL-программа вызвать программу на ассемблере? Чем может быть полезна такая возможность?

10

Компиляция и компоновка программ в z/OS

Цель. Как новому программисту приложений для z/OS, вам нужно будет создавать новые программы для выполнения в z/OS. Это потребует знания компиляции, компоновки и выполнения программы.

После завершения работы над этой главой вы сможете:

- объяснить назначение компилятора;
- скомпилировать исходную программу;
- объяснить различие между редактором связей и компоновщиком;
- создать исполняемый код из скомпилированной программы;
- объяснить различие между объектным модулем и загрузочным модулем;
- запустить программу в z/OS.

10.1 Исходный, объектный и загрузочный модули

Исходный модуль – входные данные языкового транслятора (компилятора)

Программу можно разделить на логические блоки, выполняющие определенные функции. Логический блок кода, выполняющий функцию или несколько связанных функций, называется *модулем* (*module*). Отдельные функции разрабатываются в отдельных модулях; такой процесс называется *модульным программированием* (*modular programming*). Каждый модуль может быть написан на символьном языке, который лучше всего подходит для выполняемой функции.

Объектный модуль – выходные данные языкового транслятора

Каждый модуль ассемблируется или компилируется каким-либо языковым транслятором. На вход языкового транслятора подается *исходный модуль* (*source module*); на выход языкового транслятора выводится *объектный модуль*. Прежде чем можно будет выполнить объектный модуль, он должен пройти обработку компоновщиком (или редактором связей). На выход компоновщика выводится *загрузочный модуль* (*load module*) (рис. 10.1).

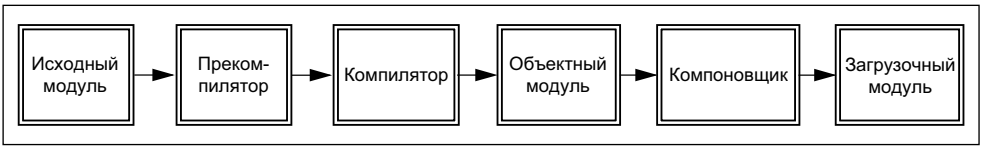


Рис. 10.1. Исходный, объектный и загрузочный модули

В зависимости от типа модуля (исходный, объектный или загрузочный), его можно сохранить в библиотеке. Библиотека представляет собой секционированный набор данных (partitioned data set, PDS) или расширенный секционированный набор данных (partitioned data set extended, PDSE) на устройстве хранения с прямым доступом. PDS и PDSE разделены на секции, называемые разделами. В библиотеке каждый раздел содержит программу или часть программы.

10.2 Что такое исходные библиотеки?

Исходные программы (или *исходный код*) представляют собой набор операторов, написанных на компьютерном языке, как описывается в главе 9 «Использование языков программирования в z/OS». Исходные программы после исправления ошибок сохраняются в секционированный набор данных, называемый *исходной библиотекой* (*source library*). Исходные библиотеки содержат исходный код для передачи в процесс компиляции или для извлечения в целях изменения программистом приложений.

Copybook представляет собой исходную библиотеку, содержащую предварительно написанный текст. Она используется для копирования текста в исходную программу во время компиляции в виде ссылки, что позволяет избежать многократного ввода одних и тех же операторов. Обычно она представляет собой разделяемую библиотеку, в которой программисты хранят часто используемые сегменты программ для

последующего включения в свои исходные программы. Ее не следует путать с подпрограммой или программой. Раздел copybook представляет собой всего лишь текстовый фрагмент; он может и не содержать настоящие операторы языка программирования.

Copybook – разделяемая библиотека, в которой программисты хранят часто используемые сегменты программ

Подпрограмма (*subroutine*) представляет собой часто вызываемую процедуру, выполняющую определенную функцию. Раздел copybook и подпрограмма имеют практически одинаковое назначение; они позволяют избежать многократного ввода одинакового кода. Однако подпрограмма представляет собой небольшую программу (скомпилированную, скомпонованную и исполняемую), вызываемую и возвращающую результат, исходя из информации, которая была ей передана. Раздел copybook представляет только лишь текст, включаемый в исходную программу при создании исполняемой программы. Слово copybook является термином языка COBOL, однако подобные понятия используются в большинстве языков программирования.

При использовании copybook в компилируемой программе можно извлекать их из исходной библиотеки, передавая оператор DD для SYSLIB или других библиотек, указанных в операторах COPY. В Примере 10.1 выполняется вставка текста раздела INPUTRCD из библиотеки DEPT88.BOBS.COBLIB в компилируемую исходную программу.

Пример 10.1. Copybook в исходном коде на языке COBOL

```
//COBOL.SYSLIB  DD  DISP=SHR,DSN=DEPT88.BOBS.COBLIB
//SYSIN  DD  *
       IDENTIFICATION DIVISION.
    . . .
       COPY  INPUTRCD
    . . .
```

Библиотеки должны находиться на устройствах хранения с прямым доступом (DASD). Они не могут находиться в файловой системе HFS при компиляции с использованием JCL или TSO.

10.3 Компиляция программ в z/OS

Редактор связей – преобразует объектные модули в исполняемые загрузочные модули

Функция компилятора состоит в том, чтобы преобразовать исходный код в объектный модуль, который затем должен пройти обработку компоновщиком (или редактором связей), прежде чем его можно будет выполнять. В процессе компиляции исходного модуля компилятор присваивает относительные адреса всем инструкциям, элементам данных и меткам, начиная с нуля.

Адреса состоят из базового адреса и смещения. Это позволяет изменять расположение программ; другими словами, их необязательно загружать в одно и то же место памяти при каждом выполнении (дополнительные сведения о перемещаемых программах см. в разделе 10.4 «Создание загрузочных модулей для исполняемых про-

грамм»). Любые ссылки на внешние программы и подпрограммы остаются неразрешенными. Эти ссылки либо разрешаются при компоновке объектного модуля, либо динамически разрешаются при выполнении программы.

Для компиляции программ в z/OS можно использовать пакетное задание или осуществлять компиляцию в TSO/E, используя команды, CLIST-программы или ISPF-панели. Программы на языке C можно компилировать в оболочке z/OS UNIX командой **c89**. Программы на языке COBOL можно компилировать в оболочке z/OS UNIX командой **cob2**.

Для компиляции в пакетном задании z/OS включает набор каталогизированных процедур, позволяющих избежать ввода JCL-кода, который иначе нужно было бы набирать. Если ни одна из каталогизированных процедур не соответствует вашим требованиям, потребуется написать весь JCL-код для компиляции.

На этапе компиляции нужно определить наборы данных, требуемые для компиляции, любые опции компилятора, необходимые для вашей программы, и желательный формат вывода.

Набор данных (библиотека), содержащая исходный код, задана оператором SYSIN DD, как показано в Примере 10.2.

Пример 10.2. Оператор SYSIN DD для исходного кода

```
//SYSIN  DD  DSNAME=dsname,
//  DISP=SHR
```

Можно поместить исходный код непосредственно во входной поток. В этом случае нужно использовать следующий оператор SYSIN DD:

```
//SYSIN  DD  *
```

При использовании обозначения DD * после оператора должен идти исходный код. Если за компиляцией идет еще один шаг задания, тогда за оператором /* или последним оператором исходного кода следует оператор EXEC.

10.3.1 Что такое прекомпилятор?

Некоторые компиляторы имеют прекомпилятор или предобработчик, предназначенный для обработки операторов, не являющихся частью компьютерного языка программирования. Если ваша исходная программа содержит операторы EXEC CICS или операторы EXEC SQL, она должна сначала пройти предобработку для преобразования этих операторов в операторы языка COBOL, PL/I или ассемблера, в зависимости от того, на каком языке написана программа.

10.3.2 Компиляция с использованием каталогизированных процедур

Простейший способ компиляции программы в операционной системе z/OS состоит в использовании пакетного задания с *каталогизированной процедурой* (*cataloged procedure*). Каталогизированная процедура представляет собой набор управляющих операторов задания, помещенных в секционированный набор данных (PDS), называемый библиотекой процедур (procedure library, PROCLIB). z/OS поставляется с библиотекой процедур SYS1.PROCLIB. Эта системная библиотека более подробно рассмат-

ривается в разделе 16.3.7 «SYS1.PROCLIB». Простейший способ понять использование каталогизированных процедур состоит в том, чтобы рассматривать их как copybooks. Однако вместо операторов исходного языка каталогизированные процедуры содержат JCL-операторы. Необязательно вводить JCL-оператор, чтобы сообщить системе, где их искать, так как они находятся в системной библиотеке, в которой поиск осуществляется автоматически при выполнении JCL-кода, указывающего на процедуру.

Каталогизированная процедура – набор операторов управления заданиями в PDS, называемом библиотекой процедур

Для выполнения компиляции необходимо включить в JCL-код следующую информацию:

- описание задания;
- оператор выполнения для вызова компилятора;
- определения наборов данных, требуемых, но не предоставляемых процедурой.

Процедура компиляции в COBOL

JCL в Примере 10.3 выполняет процедуру IGYWC, которая представляет собой одношаговую процедуру компиляции исходной программы. Она генерирует объектный модуль, сохраняемый в наборе данных SYSLIN, как показано в Примере 10.4.

Пример 10.3. Простой JCL-код для компиляции исходной программы на языке COBOL

```
//COMP JOB
//COMPILE EXEC IGYWC
//SYSIN DD *
    IDENTIFICATION DIVISION (исходная программа)
.
.
.
/*
//
```

Оператор SYSIN DD указывает расположение исходной программы. В этом случае звездочка (*) указывает на тот же входной поток.

В PL/I-программах, помимо замены исходной программы, следует заменить оператор компиляции EXEC на:

```
//compile EXEC IBMZC
```

Операторы, представленные в Примере 10.4, составляют каталогизированную процедуру IGYWC, используемую в Примере 10.3. Как говорилось выше, скомпилированная программа, генерируемая в результате процесса компиляции, помещается в набор данных, заданный оператором SYSLIN DD.

Пример 10.4. Процедура компиляция IGYWC в COBOL

```
//IGYWC PROC LNGPRFX='IGY.V3R2M0',SYSLBLK=3200
//*
//* COMPILE A COBOL PROGRAM
```

```
//*
//* PARAMETER    DEFAULT VALUE
//* SYSLBLK      3200
//* LNGPRFX      IGY.V3R2M0
//*
//* CALLER MUST SUPPLY //COBOL.SYSIN DD . . .
//*
//COBOL EXEC PGM=IGYCRCTL,REGION=2048K
//STEPLIB  DD    DSNAME=&LNGPRFX..SIGYCOMP,
//               DISP=SHR
//SYSPRINT DD    SYSOUT=*
//SYSLIN   DD    DSNAME=&&LOADSET,UNIT=SYSDA,
//               DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//               DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1   DD    UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD    UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD    UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD    UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD    UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT6   DD    UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD    UNIT=SYSDA,SPACE=(CYL,(1,1))
```

Процедура прекомпиляции, компиляции и компоновки в COBOL

JCL-код в Примере 10.5 выполняет процедуру DFHEITVL, представляющую собой трехшаговую процедуру прекомпиляции исходной программы на языке COBOL, компиляции выходных данных этапа прекомпиляции и последующей их компоновки в загрузочную библиотеку. На первом шаге создается прекомпилированный исходный код во временных наборах данных SYSPUNCH с преобразованием всех CICS-вызовов в операторы языка COBOL. На втором шаге этот временный набор данных используется в качестве входных данных и генерируется объектный модуль, сохраняемый во временном наборе данных SYSLIN, как показано в Примере 10.6. На третьем шаге временный набор данных SYSLIN и остальные модули, которые требуется включить, используются в качестве входных данных, и создается загрузочный модуль в наборе данных, на который указывает оператор SYSLMOD DD.

В Примере 10.5 показано, что JCL-код несколько более сложен, чем в простой задаче компиляции (см. Пример 10.3). При переходе от одношагового к многошаговому процессу необходимо сообщить системе, к какому шагу идет обращение при предоставлении JCL-замещений.

В JCL-коде в Примере 10.6 мы видим, что первый шаг (каждый шаг представляет собой оператор EXEC, и имя шага представляет имя в той же строке, что и оператор EXEC) имеет имя TRN, поэтому необходимо включить в оператор SYSIN DD имя TRN, чтобы обеспечить его использование на шаге TRN.

Подобным образом, четвертый шаг имеет имя LKED, поэтому необходимо включить в оператор SYSIN DD имя LKED, чтобы обеспечить его применение к шагу LKED. Дополнительные сведения о замещении каталогизированной процедуры см. в разделе 6.7.1 «Замещение оператора JCL-процедуры».

Конечным результатом выполнения JCL-кода из Примера 10.5 (при условии отсутствия ошибок) должна быть прекомпиляция и компиляция встроенной в поток исходной программы, компоновка объектного модуля и последующее сохранение загрузочного модуля PROG1 в наборе данных MY.LOADLIB.

Пример 10.5. Простой JCL-код для прекомпиляции, компиляции и компоновки исходной программы на языке COBOL

```
//PPCOMLNK JOB
//PPCL EXEC DFHEITVL,PROGLIB='MY.LOADLIB'
//TRN.SYSIN DD *
    IDENTIFICATION DIVISION (исходная программа)
      EXEC CICS ...
...
      EXEC CICS ...
...
//LKED.SYSIN DD *
NAME PROG1(R)
/*
```

Операторы, представленные в Примере 10.6, составляют каталогизированную процедуру DFHEITVL, используемую в Примере 10.5. Как и в случае с другими процедурами компиляции и компоновки, результат этапов прекомпиляции, компиляции и компоновки, представляющий собой загрузочный модуль, помещается в набор данных, указанный в операторе SYSLMOD DD.

Пример 10.6. Процедура DFHEITVL – прекомпиляция, компиляция и компоновка в COBOL

```
//DFHEITVL PROC SUFFIX=1$, Суффикс модуля транслятора
//*
//* Эта процедура была изменена по сравнению с CICS/ESA Version 3
//*
//* Параметр INDEX2 был удален
//*
//    INDEX='CICSTS12.CICS', Квалификатор(ы) для библиотек CICS
//    PROGLIB=&INDEX..SDFHLOAD, Имя выходной библиотеки
//    DSCTLIB=&INDEX..SDFHCOB,  Имя личной библиотеки макросов/DSECT
//    COMPHLQ='SYS1',           Квалификатор(ы) компилятора COBOL
//    OUTC=A,                   Класс вывода на печать
//    REG=2M,                   Размер региона для всех шагов
//    LNKPARM='LIST,XREF',      Параметры компоновки
```

```
//    STUB='DFHEILIC', INCLUDE компоновки для DFHECI
//    LIB='SDFHCOB', Библиотека
//    WORK=SYSDA Устройство для рабочих наборов данных
//*   Эта процедура содержит 4 шага
//* 1.     Вызов транслятора COBOL
//*        (с использованием заданного суффикса 1$)
//* 2.     Вызов компилятора COBOL II
//* 3.     Повторное разбиение &LIB(&STUB) на блоки для
//         использования на шаге компоновки
//* 4.     Компоновка выходных данных в наборе данных &PROGLIB
//*
//* Следующий JCL-код следует использовать
//* для выполнения этой процедуры
//*
//* //APPLPROG EXEC DFHEITVL
//* //TRN.SYSIN DD *
//*   .
//*   . Приложение
//*   .
//* /*
//* //LKED.SYSIN DD *
//*   NAME anyname(R)
//* /*
//*
//* Где anyname - имя вашего приложения.
//*   (Подробные сведения см. в руководстве по системным
//*   определениям, включая сведения о том, что делать, если
//*   программа содержит обращения к общим интерфейсам
//    программирования)
//*
//TRN      EXEC   PGM=DFHECP&SUFFIX,
//             PARM='COBOL2',
//             REGION=&REG
//STEPLIB  DD  DSN=&INDEX..SDFHLOAD,DISP=SHR
//SYSPRINT DD  SYSOUT=&OUTC
//SYSPUNCH DD  DSN=&&SYSCIN,
//             DISP=(,PASS),UNIT=&WORK,
//             DCB=BLKSIZE=400,
//             SPACE=(400,(400,100))
//*
```

```
//COB  EXEC  PGM=IGYCRCTL,REGION=&REG,
//         PARM='NODYNAM,LIB,OBJECT,RENT,RES,APOST,MAP,XREF'
//STEPLIB  DD  DSN=&COMPHLQ..COB2COMP,DISP=SHR
//SYSLIB   DD  DSN=&DSCTLIB,DISP=SHR
//         DD  DSN=&INDEX..SDFHCOB,DISP=SHR
//         DD  DSN=&INDEX..SDFHMAC,DISP=SHR
//         DD  DSN=&INDEX..SDFHSAMP,DISP=SHR
//SYSPRINT DD  SYSOUT=&OUTC
//SYSIN    DD  DSN=&&SYSCIN,DISP=(OLD,DELETE)
//SYSLIN   DD  DSN=&&LOADSET,DISP=(MOD,PASS),
//             UNIT=&WORK,SPACE=(80,(250,100))
//SYSUT1   DD  UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT2   DD  UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT3   DD  UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT4   DD  UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT5   DD  UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT6   DD  UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT7   DD  UNIT=&WORK,SPACE=(460,(350,100))
//*
//COPYLINK EXEC  PGM=IEBGENER,COND=(7,LT,COB)
//SYSUT1   DD  DSN=&INDEX..&LIB(&STUB),DISP=SHR
//SYSUT2   DD  DSN=&&COPYLINK,DISP=(NEW,PASS),
//             DCB=(LRECL=80,BLKSIZE=400,RECFM=FB),
//             UNIT=&WORK,SPACE=(400,(20,20))
//SYSPRINT DD  SYSOUT=&OUTC
//SYSIN    DD  DUMMY
//*
//LKED EXEC  PGM=IEWL,REGION=&REG,
//    PARM='&LNKPARM',COND=(5,LT,COB)
//SYSLIB   DD  DSN=&INDEX..SDFHLOAD,DISP=SHR
//         DD  DSN=&COMPHLQ..COB2CICS,DISP=SHR
//         DD  DSN=&COMPHLQ..COB2LIB,DISP=SHR
//SYSLMOD  DD  DSN=&PROGLIB,DISP=SHR
//SYSUT1   DD  UNIT=&WORK,DCB=BLKSIZE=1024,
//             SPACE=(1024,(200,20))
//SYSPRINT DD  SYSOUT=&OUTC
//SYSLIN   DD  DSN=&&COPYLINK,DISP=(OLD,DELETE)
//         DD  DSN=&&LOADSET,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
```

Процедура компиляции и компоновки в COBOL

JCL-код в Примере 10.7 выполняет процедуру IGYWCL, которая представляет собой двухшаговую процедуру компиляции исходной программы и ее компоновки в загрузочную библиотеку. Первый шаг генерирует объектный модуль, сохраняемый во временном наборе данных SYSLIN, как показано в Примере 10.8. На втором шаге в качестве входных данных принимается временный набор данных SYSLIN, а также любые другие модули, которые может потребоваться включить, и создается загрузочный модуль в наборе данных, заданном оператором SYSLMOD DD.

Конечным результатом выполнения JCL-кода из Примера 10.7 (при условии отсутствия ошибок) должна быть компиляция встроенной в поток исходной программы, компоновка объектного модуля и последующее сохранение загрузочного модуля PROG1 в наборе данных MY.LOADLIB.

Пример 10.7. Простой JCL-код для компиляции и компоновки исходной программы на языке COBOL

```
//COMLNK   JOB
//CL EXEC IGYWCL
//COBOL.SYSIN DD *
    IDENTIFICATION DIVISION (исходная программа)
..
.
/*
//LKED.SYSLMOD DD DSN=MY.LOADLIB(PROG1),DISP=OLD
```

Операторы, представленные в Примере 10.8, составляют каталогизированную процедуру IGYWCL, используемую в Примере 10.7. Как говорилось выше, результатом шагов компиляции и компоновки является загрузочный модуль, помещаемый в набор данных, заданный оператором SYSLMOD DD.

Пример 10.8. Процедура IGYWCL – компиляция и компоновка в COBOL

```
// IGYWCL PROC   LNGPRFX='IGY.V2R1M0',SYSLBLK=3200,
//               LIBPRFX='CEE',
//               PGMLIB='&&GOSET',GOPGM=GO
//*
//* COMPILE AND LINK EDIT A COBOL PROGRAM
//*
//* PARAMETER    DEFAULT VALUE
//* LNGPRFX      IGY.V2R1M0
//* SYSLBLK      3200
//* LIBPRFX      CEE
//* PGMLIB       &&GOSET     DATA SET NAME FOR LOAD MODULE
//* GOPGM        GO          MEMBER NAME FOR LOAD MODULE
//*
//* CALLER MUST SUPPLY //COBOL.SYSIN DD ...
```

```
//*
//COBOL  EXEC  PGM=IGYCRCTL,REGION=2048K
//STEPLIB  DD  DSNAME=&LNGPRFX..SIGYCOMP,
//  DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,UNIT=VIO,
//             DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//  DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT4   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT6   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT7   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//LKED  EXEC  PGM=HEWL,COND=(8,LT,COBOL),REGION=1024K
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&PGMLIB(&GOPGM),
//             SPACE=(TRK,(10,10,1)),
//             UNIT=VIO,DISP=(MOD,PASS)
//SYSUT1  DD  UNIT=VIO,SPACE=(TRK,(10,10))
```

Процедура компиляции, компоновки и запуска в COBOL

JCL-код в Примере 10.9 выполняет процедуру IGYWCLG, которая представляет собой трехшаговую процедуру компиляции исходной программы, ее компоновки в загрузочную библиотеку и последующего выполнения загрузочного модуля. Первые два шага не отличаются от приведенных в примере с компиляцией и компоновкой (Пример 10.7). Однако, если в Примере 10.7 выполняется замещение оператора SYSLMOD DD для перманентного сохранения загрузочного модуля, то в Примере 10.9 нам не требуется его сохранять для выполнения. Поэтому замещение оператора SYSLMOD DD в Примере 10.9 заключено в квадратные скобки, указывая, что оно является необязательным.

Если оно используется, тогда загрузочный модуль PROG1 будет перманентно сохранен в MY.LOADLIB. Если оно не используется, тогда загрузочный модуль будет сохранен во временном наборе данных и удален после этапа запуска (GO).

В Примере 10.9 мы видим, что JCL-код очень похож на JCL-код, использовавшийся в задании простой компиляции (см. Пример 10.3). По сравнению с JCL-кодом в Примере 10.8, единственное различие в JCL-коде в Примере 10.10 состоит в том, что был

добавлен шаг запуска (GO). Конечным результатом выполнения JCL-кода из Примера 10.9 (при условии отсутствия ошибок) должна быть компиляция встроенной в поток исходной программы, компоновка объектного модуля, сохранение (временное или перманентное) загрузочного модуля и последующее выполнение загрузочного модуля.

Пример 10.9. Простой JCL-код для компиляции, компоновки и выполнения исходной программы на языке COBOL

```
//CLGO JOB
//CLG EXEC IGYWCLG
//COBOL.SYSIN DD *
IDENTIFICATION DIVISION (исходная программа)
...
/*
[//LKED.SYSLMOD DD DSN=MY.LOADLIB(PROG1),DISP=OLD]
```

Операторы, представленные в Примере 10.10, составляют каталогизированную процедуру IGYWCLG, используемую в Примере 10.9.

Пример 10.10. Процедура IGYWCLG – компиляция, компоновка и выполнение в COBOL

```
//IGYWCLG PROC LNGPRFX='IGY.V2R1M0',SYSLBLK=3200,
// LIBPRFX='CEE',GOPGM=GO
//*
//* COMPILE, LINK EDIT AND RUN A COBOL PROGRAM
//*
//* PARAMETER  DEFAULT VALUE USAGE
//* LNGPRFX    IGY.V2R1M0
//* SYSLBLK    3200
//* LIBPRFX    CEE
//* GOPGM      GO
//*
//* CALLER MUST SUPPLY //COBOL.SYSIN DD ...
//*
//COBOL EXEC PGM=IGYCRCTL,REGION=2048K
//STEPLIB DD DSNAME=&LNGPRFX..SIGYCOMP,
// DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,UNIT=VIO,
//             DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//             DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=VIO,SPACE=(CYL,(1,1))
```

```
//SYSUT4   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT6   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT7   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//LKED EXEC PGM=HEWL,COND=(8,LT,COBOL),REGION=1024K
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&&GOSET(&GOPGM),SPACE=(TRK,(10,10,1)),
//             UNIT=VIO,DISP=(MOD,PASS)
//SYSUT1   DD  UNIT=VIO,SPACE=(TRK,(10,10))
//GO EXEC PGM=*.LKED.SYSLMOD,COND=((8,LT,COBOL),(4,LT,LKED)),
//             REGION=2048K
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
```

10.3.3 Компиляция объектно-ориентированных приложений

Если используется пакетное задание или TSO/E для компиляции объектно-ориентированной COBOL-программы или определения класса, сгенерированный объектный модуль записывается, как обычно, в набор данных, заданный DD-именем SYSLIN или SYSPUNCH.

Если COBOL-программа или определение класса использует структуру окружения JNI[1] для доступа к вызываемым службам JNI, следует скопировать файл JNI.cpy из HFS в раздел PDS или PDSE с названием JNI, указать эту библиотеку в операторе SYSLIB DD и использовать оператор COPY вида COPY JNI в исходной программе на языке COBOL.

В Примере 10.11 представлено использование DD-имени SYSJAVA для записи сгенерированного исходного Java-файла в файл в HFS. Например:

Пример 10.11. DD-имя SYSJAVA для исходного Java-файла

```
//SYSJAVA DD PATH='/u/userid/java/Classname.java',
//  PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//  PATHMODE=SIRWXU,
//  FILEDATA=TEXT
```

[1] Java Native Interface (JNI) представляет собой Java-интерфейс к традиционным языкам программирования и входит в JDK. Разработка программ с использованием JNI позволяет обеспечить переносимость кода между различными платформами.

10.3.4 Что такое объектный модуль?

Объектный модуль представляет собой набор из одного или нескольких единиц компиляции, сгенерированных ассемблером, компилятором или другим языковым транслятором, и используемых в качестве входных данных для компоновщика (или редактора связей).

Объектный модуль имеет перемещаемый формат и содержит неисполняемый машинный код. Загрузочный модуль также имеет перемещаемый формат, однако содержит исполняемый машинный код. Загрузочный модуль имеет формат, который позволяет менеджеру программ (программе, подготавливающей загрузочные модули к выполнению, загружая их в определенные участки памяти) загружать его в виртуальную память и перемещать.

Объектные модули и загрузочные модули имеют одинаковую логическую структуру, состоящую из следующих элементов:

- управляющие словари, содержащие информацию для разрешения символических перекрестных ссылок между управляющими секциями разных модулей и перемещения адресных констант;
- текст, содержащий инструкции и данные программы;
- указатель конца модуля, представленный оператором END в объектном модуле или указателем конца модуля в загрузочном модуле.

Объектные модули хранятся в секционированном наборе данных, заданном оператором SYSLIN или SYSPUNCH DD, который подается на вход следующего процесса – компоновки.

10.3.5 Что такое объектная библиотека?

Для хранения объектных модулей можно использовать объектную библиотеку. Объектные модули, подлежащие компоновке, извлекаются из объектной библиотеки и преобразуются в исполняемую или загружаемую программу.

Использование опции компилятора OBJECT позволяет сохранить объектный модуль либо на диск в виде традиционного набора данных или UNIX-файла, либо на магнитную ленту. Параметр DISP оператора SYSLIN DD указывает, что следует сделать с объектным модулем:

- передать в компоновщик (или в редактор связей) после компиляции (DISP=PASS);
- каталогизировать в существующей объектной библиотеке (DISP=OLD);
- сохранить (DISP=KEEP);
- добавить в новую объектную библиотеку, каталогизируемую в конце этапа (DISP=CATLG).

Можно определить объектный модуль в качестве основных входных данных компоновщика, указав имя набора данных и имя раздела в операторе SYSLIN DD. В следующем примере в качестве основных входных данных задан раздел TAXCOMP объектной библиотеки USER.LIBROUT. USER.LIBROUT представляет собой каталогизированный секционированный набор данных:

```
//SYSLIN DD DSNAME=USER.LIBROUT(TAXCOMP),DISP=SHR
```

Раздел библиотеки обрабатывается таким образом, как если бы он представлял собой последовательный набор данных.

10.3.6 Каким образом осуществляется управление программами?

Несмотря на то, что компоненты управления программами содержат множество служб, они используются, главным образом, для преобразования объектных модулей в исполняемые программы, их сохранения в программных библиотеках и их загрузки в виртуальную память для выполнения.

Для выполнения этих задач можно использовать компоновщик и загрузчик программ. Эти компоненты можно также использовать в сочетании с редактором связей. Загрузочный модуль, сгенерированный редактором связей, может быть использоваться в качестве входных данных компоновщика, либо может быть загружен в память для выполнения загрузчиком программ. Редактор связей может также обрабатывать загрузочные модули, генерируемые компоновщиком.

На рис. 10.2 показано, как осуществляется совместная работа компонентов управления программами, и каким образом каждый из них используется для подготовки исполняемой программы. Мы уже обсудили некоторые из этих компонентов (исходные модули и объектные модули), так что теперь можно рассмотреть остальные компоненты.

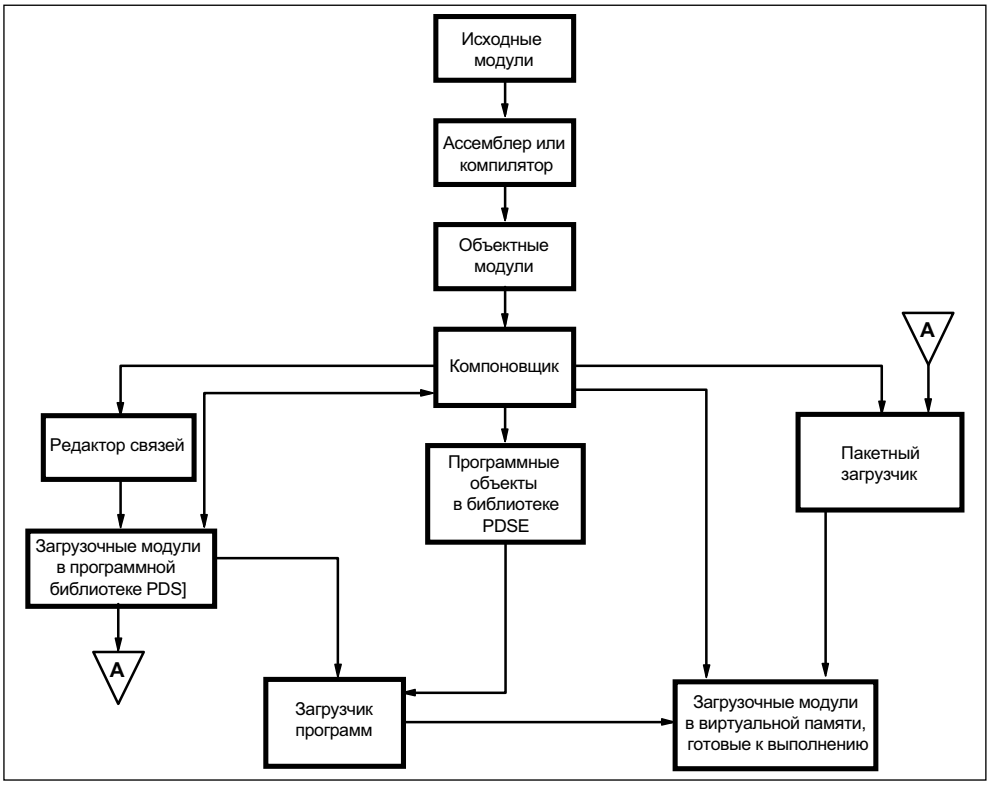


Рис. 10.2. Использование компонентов управления программами для создания и загрузки программ

10.3.7 Как используется редактор связей?

Обработка в редакторе связей осуществляется после ассемблирования или компиляции любой программы. *Редактор связей* (*linkage editor*) представляет собой как обрабатывающую, так и сервисную программу, используемую в сочетании с языковыми трансляторами.

Обрабатывающие программы редактора связей и загрузчика готовят выходные данные языковых трансляторов к выполнению. Редактор связей подготавливает загрузочный модуль перед его переносом в память к выполнению менеджером программ.

Редактор связей принимает два основных вида входных данных:

- Основные входные данные, содержащие объектные модули и управляющие операторы редактора связей.
- Дополнительные пользовательские входные данные, которые могут содержать либо объектные модули с управляющими операторами, либо загрузочные модули. Эти входные данные либо задаются как данные для ввода, либо автоматически включаются редактором связей из библиотеки вызовов.

Существует два вида выходных данных редактора связей:

- Загрузочный модуль, помещаемый в библиотеку (секционированный набор данных) как именованный раздел.
- Диагностические выходные данные, генерируемые в виде последовательного набора данных.

Загрузчик подготавливает исполняемую программу в памяти и передает управление непосредственно ей.

10.3.8 Как создается загрузочный модуль

При обработке объектных модулей и загрузочных модулей редактор связей присваивает последовательные относительные адреса виртуальной памяти управляющим секциям и разрешает ссылки между управляющими секциями. Объектные модули, сгенерированные разными языковыми трансляторами, можно использовать для создания одного загрузочного модуля.

Выходной загрузочный модуль состоит из всех входных объектных и загрузочных модулей, обрабатываемых редактором связей. Поэтому управляющие словари выходного модуля представляют собой смесь всех управляющих словарей, передаваемых на вход редактора связей. Управляющие словари загрузочного модуля называются композитным внешним системным словарем (composite external symbol dictionary, CESD) и словарем перемещений (relocation dictionary, RLD). Загрузочный модуль также содержит текст из каждого входного модуля и указатель конца модуля.

Рисунок 10.3 иллюстрирует процесс компиляции двух исходных программ: PROGA и PROGB. PROGA представляет собой COBOL-программу, а PROGB представляет собой программу на ассемблере. PROGA вызывает PROGB. На этом рисунке мы видим, что после компиляции ссылка на PROGB в программе PROGA является неразрешенной. Процесс редактирования связей двух объектных модулей разрешает ссылку та-

ким образом, что при выполнении программы PROGA обращение к PROGB будет работать должным образом. Управление будет передано в PROGB, она выполнится, после чего управление возвратиться к PROGA в точку, следующую за вызовом PROGB.

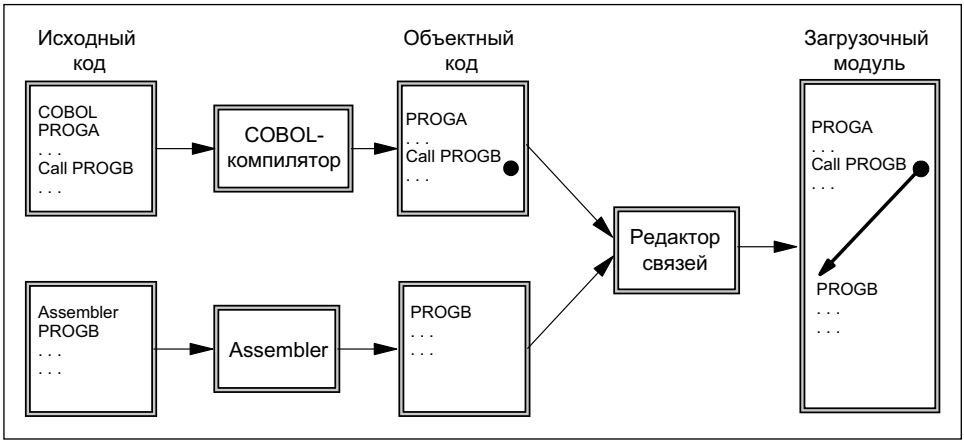


Рис. 10.3. Разрешение ссылок при создании загрузочного модуля

Использование компоновщика

Компоновщик (*binder*), входящий в z/OS, выполняет все функции редактора связей. Компоновщик осуществляет редактирование связей (объединение и редактирование) отдельных объектных модулей, загрузочных модулей и программных объектов, составляющих приложение, и генерирует единый объектный или загрузочный модуль программы, который можно загрузить для выполнения. Когда нужен раздел программной библиотеки, загрузчик переносит его в виртуальную память и подготавливает к выполнению.

Программная библиотека – набор данных, содержащий загрузочные модули и программные объекты

Компоновщик можно использовать в следующих целях:

- Для преобразования объектного или загрузочного модуля в программный объект и его сохранения в программной библиотеке в расширенном секционированном наборе данных (PDSE) или в файле z/OS UNIX.
- Для преобразования объектного модуля или программного объекта в загрузочный модуль и его сохранение в программной библиотеке в секционированном наборе данных (PDS). Это аналогично тому, что делает редактор связей с объектными и загрузочными модулями.
- Для преобразования объектных или загрузочных модулей, либо программных объектов в исполняемую программу в виртуальной памяти с последующим выполнением программы. Это аналогично тому, что делает пакетный загрузчик с объектными и загрузочными модулями.

Компоновщик обрабатывает объектные, загрузочные модули и программные объекты, выполняя редактирование связей или компоновку нескольких модулей

в единый загрузочный модуль или программный объект. Управляющие операторы определяют, каким образом осуществлять комбинирование входных данных в одном или нескольких загрузочных модулях или программных объектах с последовательными адресами виртуальной памяти. Каждый объектный модуль может обрабатываться компоновщиком отдельно, так что только измененные модули потребуют повторной компиляции или ассемблирования. Компоновщик может создавать программы в 24-разрядном, 31-разрядном и 64-разрядном режимах адресации.

Назначение режима адресации (AMODE) выполняется для того, чтобы указать, какой аппаратный режим адресации должен быть активным при выполнении программы. Существуют следующие режимы адресации:

- 24 – указывает, что необходимо использовать 24-разрядную адресацию;
- 31 – указывает, что необходимо использовать 31-разрядную адресацию;
- 64 – указывает, что необходимо использовать 64-разрядную адресацию;
- ANY – указывает, что можно использовать 24-, 31- или 64-разрядную адресацию;
- MIN – указывает, что значение AMODE для программного модуля должно назначаться компоновщиком.

Компоновщик выбирает наиболее ограничительное значение AMODE из всех управляющих секций входных данных программного модуля. *Наиболее* ограничительное значение AMODE равно 24; *наименее* ограничительным значением AMODE является значение ANY.

Компоновщик может выполнять все те же задачи, что и редактор связей. Дополнительные сведения о структуре адреса и о том, какие области адресного пространства имеют 24-, 31- и 64-разрядную адресацию см. в разделе 3.4.9 «Краткая история виртуальной памяти и 64-разрядной адресуемости».

Компоновщик и редактор связей

Компоновщик ослабляет или устраняет многие ограничения, свойственные редактору связей. Компоновщик устраняет ограничение в 64 алиаса, свойственное редактору связей, что позволяет загрузочному модулю или программному объекту использовать любое требуемое количество алиасов. Компоновщик поддерживает использование любого размера блока основного входного набора данных (SYSLIN), что устраняет максимальный предел размера блока, равный 3200 байтам. Кроме того, компоновщик не ограничивает количество внешних имен, тогда как редактор связей устанавливает предел в 32 767 имен.

Кроме того, предварительный редактор связей (prelinker), входящий в z/OS Language Environment, представляет собой еще одно средство объединения объектных модулей в единый объектный модуль. После предварительного связывания можно осуществлять редактирование связей объектного модуля и его запись в загрузочный модуль (сохраняемый в PDS), либо его компоновку в загрузочный модуль или программный объект (сохраняемый в PDS, PDSE или файл zFS). Однако при использовании компоновщика программистам приложений z/OS больше не требуется выполнять предварительное связывание, так как компоновщик выполняет все функции

предварительного редактора связей. Выбор между компоновщиком и редактором связей является вопросом предпочтения. Компоновщик является более современным способом создания загрузочного модуля.

Основные входные данные, требуемые для каждого шага задания компоновщика, определены в операторе DD с указанием DD-имени SYSLIN. Основные входные данные могут представлять собой:

- Последовательный набор данных.
  Раздел секционированного набора данных (PDS).
  Раздел расширенного секционированного набора данных (PDSE).
  Сцепленные последовательные наборы данных или разделы секционированных наборов данных или PDSE, или их сочетание.
- Файл z/OS UNIX.
  Основной набор данных может содержать объектные модули, управляющие операторы, загрузочные модули и программные объекты. Все модули и управляющие операторы обрабатываются последовательно, и их порядок определяет порядок обработки компоновщиком. Однако порядок секций после обработки может не соответствовать входной последовательности.

Пример вызова компоновщика

В Примере 10.12 представлено задание, которое можно использовать для компоновки объектного модуля. Выходные данные шага LKED помещаются в личную библиотеку, определенную оператором SYSLMOD DD. Входные данные передаются с предыдущего шага задания на шаг компоновщика в том же задании (например, выходные данные компилятора являются непосредственными входными данными компоновщика).

Пример 10.12. Пример JCL-кода компоновщика

```
//LKED   EXEC PGM=IEWL,PARM='XREF,LIST', IEWL - алиас IEWBLINK
//         REGION=2M,COND=(5,LT,prior-step)
//*
//*      Определение дополнительных входных данных
//*
//SYSLIB   DD  DSN=language.library,DISP=SHR необязательно
//PRIVLIB  DD  DSN=private.include.library,DISP=SHR необязательно
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1)) игнорируется
//*
//*      Определение библиотеки выходного модуля
//*
//SYSLMOD  DD  DSN=program.library,DISP=SHR обязательно
//SYSPRINT DD  SYSOUT=* обязательно
//SYSTERM  DD  SYSOUT=* необязательно
//*
```

```
//*      Определение основных входных данных
//*
//SYSLIN   DD     DSN=&&OBJECT,DISP=(MOD,PASS)  обязательно
//          DD *   встраиваемые управляющие операторы
          INCLUDE  PRIVLIB(membername)
          NAME     modname(R)
/*
```

Ниже представлено описание JCL-операторов:

EXEC	Выполняет компоновку программного модуля и сохраняет его в программной библиотеке. Альтернативными именами IEWBLINK являются IEWL, LINKEDIT, EWL и HEWLH096. Опция PARM запрашивает вывод таблицы перекрестных ссылок и схемы модулей в диагностическом выходном наборе данных.
SYSUT1	Определяет временный набор данных прямого доступа для использования в качестве промежуточного набора данных.
SYSLMOD	Определяет временный набор данных для использования в качестве выходной библиотеки модулей.
SYSPRINT	Определяет диагностический выходной набор данных, назначаемый классу вывода A.
SYSLIN	Определяет основной входной набор данных &&OBJECT, содержащий входной объектный модуль; этот набор данных был передан с предыдущего шага задания и предназначен для передачи в конце данного шага задания.
INCLUDE	Определяет последовательные наборы данных, разделы библиотеки или файлы z/OS UNIX, предназначенные для использования в качестве источников дополнительных входных данных компоновщика (в данном случае, раздела личной библиотеки PRIVLIB).
NAME	Задает имя программного модуля, созданного из предыдущих входных модулей, и используется в качестве разделителя входных данных програмного модуля. (R) указывает, что этот программный модуль заменяет модуль с таким же именем в выходной библиотеке модулей.

10.4 Создание загрузочных модулей для исполняемых программ

Загрузочный модуль (*load module*) представляет собой исполняемую программу, сохраненную в программной библиотеке в секционированном наборе данных. Создание загрузочного модуля только для выполнения потребует использования пакетного загрузчика или загрузчика программ. Создание загрузочного модуля, который можно сохранить в программной библиотеке, требует использования компоновщика или редактора связей. В любом случае, загрузочный модуль является перемещаемым. Это означает, что он может размещаться по любому адресу в виртуальной памяти в пределах, заданных режимом размещения (RMODE).

Перемещаемый модуль – загрузочный модуль, который может размещаться по любому адресу в виртуальной памяти

После загрузки программы ей передается управление с некоторым значением базового регистра. Он сообщает программе ее начальный адрес, по которому она была загружена, так что все адреса можно разрешать как сумму базового адреса

и смещения. Перемещаемые программы позволяют загрузить идентичные копии программы в различные адресные пространства, каждая из которых загружается по разным начальным адресам. Дополнительные сведения о перемещаемых программах см. в разделе 10.3, «Компиляция программ в z/OS».

10.4.1 Пакетный загрузчик

Пакетный загрузчик (batch loader) объединяет возможности простого редактирования и загрузки (которые также могут быть реализованы в редакторе связей и менеджере программ) в один этап. Пакетный загрузчик принимает объектные и загрузочные модули и загружает их в виртуальную память для выполнения. В отличии от компоновщика и редактора связей, пакетный загрузчик не генерирует загрузочные модули, которые можно было бы сохранить в программных библиотеках. Пакетный загрузчик подготавливает исполняемую программу в памяти и передает ей управление.

Обработка пакетным загрузчиком выполняется на этапе загрузки, который соответствует этапам компоновки и запуска в компоновщике или редакторе связей. Пакетный загрузчик может использоваться в заданиях компиляции-загрузки и загрузки . Он может включать модули из библиотеки вызовов (SYSLIB) и/или из области загрузки модулей (link pack area, LPA). Подобно другим компонентам управления программами, пакетный загрузчик поддерживает атрибуты адресации и режимов размещения в 24-, 31- и 64-разрядных режимах адресации. Программа пакетного загрузчика является реентерабельной и поэтому может размещаться в резидентной области загрузки модулей.

> **Примечание.** В последних версиях z/OS пакетный загрузчик был заменен компоновщиком.

10.4.2 Загрузчик программ

Загрузчик программ (program management loader) расширяет возможности компонента менеджера программ посредством реализации поддержки загрузки программных объектов. Загрузчик считывает как программные объекты, так и загрузочные модули в виртуальную память и подготавливает их к выполнению. Он разрешает любые адресные константы в программе таким образом, чтобы они указывали на определенные области виртуальной памяти, и поддерживает 24-, 31- и 64-разрядный режимы адресации.

При обработке объектных и загрузочных модулей редактор связей назначает последовательные относительные адреса виртуальной памяти управляющим секциям и разрешает ссылки между управляющими секциями. Объектные модули, генерируемые различными языковыми трансляторами, можно использовать для создания одного загрузочного модуля.

В Примере 10.13 необходимо выполнить компиляцию, компоновку и выполнение, в данном случае, для программы на ассемблере.

Пример 10.13. JCL-код компиляции, компоновки и выполнения

```
//USUAL  JOB  A2317P,'COMPLGO'
//ASM    EXEC PGM=IEV90,REGION=256K, ИСПОЛНЕНИЕ ПРОГРАММЫ НА АССЕМБЛЕРЕ
//           PARM=(OBJECT,NODECK,'LINECOUNT=50')
```

```
//SYSPRINT  DD  SYSOUT=*,DCB=BLKSIZE=3509 ПЕЧАТЬ ЛИСТИНГА НА АССЕМБЛЕРЕ
//SYSPUNCH DD  SYSOUT=B ПЕРФОРИРОВАНИЕ ЛИСТИНГА НА АССЕМБЛЕРЕ
//SYSLIB   DD  DSNAME=SYS1.MACLIB,DISP=SHR БИБЛИОТЕКА МАКРОСОВ
//SYSUT1   DD  DSNAME=&&SYSUT1,UNIT=SYSDA, РАБОЧИЙ НАБОР ДАННЫХ
//         SPACE=(CYL,(10,1))
//SYSLIN   DD  DSNAME=&&OBJECT,UNIT=SYSDA, ВЫХОДНОЙ ОБЪЕКТНЫЙ МОДУЛЬ
//         SPACE=(TRK,(10,2)),DCB=BLKSIZE=3120,DISP=(,PASS)
//SYSIN    DD  * встроенный в поток ИСХОДНЫЙ КОД
               .
               код
               .
/*
//LKED     EXEC   PGM=HEWL, ЗАПУСК РЕДАКТОРА СВЯЗЕЙ
//         PARM='XREF,LIST,LET',COND=(8,LE,ASM)
//SYSPRINT DD  SYSOUT=* ПЕЧАТЬ MAP-ФАЙЛА РЕДАКТОРА СВЯЗЕЙ
//SYSLIN   DD  DSNAME=&&OBJECT,DISP=(OLD,DELETE) ВХОДНОЙ ОБЪЕКТНЫЙ МОДУЛЬ
//SYSUT1   DD  DSNAME=&&SYSUT1,UNIT=SYSDA, РАБОЧИЙ НАБОР ДАННЫХ
//            SPACE=(CYL,(10,1))
//SYSLMOD  DD  DSNAME=&&LOADMOD,UNIT=SYSDA, ВЫХОДНОЙ ЗАГРУЗОЧНЫЙ
                                              МОДУЛЬ
//            DISP=(MOD,PASS),SPACE=(1024,(50,20,1))
//GO  EXEC  PGM=*.LKED.SYSLMOD,TIME=(,30), ИСПОЛНЕНИЕ ПРОГРАММЫ
//            COND=((8,LE,ASM),(8,LE,LKED))
//SYSUDUMP DD  SYSOUT=* ВЫВОД ДАМПА ЛИСТИНГА ПРИ ОТКАЗЕ
//SYSPRINT DD  SYSOUT=*, ВЫВОД ЛИСТИНГА
//            DCB=(RECFM=FBA,LRECL=121)
//OUTPUT   DD  SYSOUT=A, ВЫВОД ПРОГРАММНЫХ ДАННЫХ
//            DCB=(LRECL=100,BLKSIZE=3000,RECFM=FBA)
//INPUT    DD  * ВВОД ПРОГРАММНЫХ ДАННЫХ
.
.
данные
.
/*
//
```

Примечание.

- На шаге ASM (компиляция) оператор SYSIN DD соответствует встраиваемому исходному коду, а оператор SYSLIN DD – выходному объектному модулю.
- На шаге LKED (редактирование связей) оператор SYSLIN DD соответствует входному объектному модулю, а оператор SYSLMOD DD – выходному загрузочному модулю.
- На шаге GO (запуск программы) оператор EXEC JCL указывает, что он будет выполнять программу, определенную оператором SYSLMOD DD на предыдущем этапе.
- В этом примере не используется каталогизированная процедура, как в примерах на языке COBOL; вместо этого, весь JCL-код был записан как встраиваемый. Мы можем либо использовать существующую JCL-процедуру, либо составить ее, а затем только вводить замещения, например, оператором INPUT DD.

10.4.3 Что такое загрузочная библиотека?

Загрузочная библиотека (*load library*) содержит программы, готовые для выполнения. Загрузочная библиотека может иметь один из следующих типов:

- системная библиотека;
- личная библиотека;
- временная библиотека.

Системная библиотека

Если задание или шаг не указывает личную библиотеку, система осуществляет поиск программы в системных библиотеках при вводе следующей команды:

```
//stepname EXEC PGM=program-name
```

Система просматривает библиотеки, осуществляя поиск раздела с именем или алиасом, соответствующим program-name (имени программы). Наиболее используемой системной библиотекой является библиотека SYS1.LINKLIB, содержащая исполняемые программы, обработанные редактором связей. Дополнительные сведения о системных библиотеках см. в разделе 16.3.1 «Системные библиотеки z/OS».

Личная библиотека

Каждая исполняемая программа, написанная пользователем, является разделом личной библиотеки. Для того чтобы сообщить системе о том, что программа находится в личной библиотеке, можно использовать оператор DD, определяющий эту библиотеку, одним из следующих способов:

- Используя оператор DD с DD-именем JOBLIB после оператора JOB и до первого оператора EXEC в задании;
- Если вы намереваетесь использовать библиотеку только на одном шаге – тогда используя оператор DD с DD-именем STEPLIB на данном шаге.

Для запуска программы из личной библиотеки, следует ввести:

```
//stepname EXEC PGM=program-name
```

Если используется JOBLIB или STEPLIB, система осуществляет поиск выполняемой программы в библиотеке, определенной оператором JOBLIB или STEPLIB DD, до поиска в системных библиотеках.

Если предыдущий оператор DD в задании определяет программу как раздел личной библиотеки, для выполнения программы следует использовать следующий оператор DD:

```
//stepname EXEC PGM=*.stepname.ddname
```

Личные библиотеки особенно полезны для программ, используемых слишком редко для того, чтобы помещать их в системную библиотеку. Например, программы, подготавливающие квартальные налоговые отчеты о продажах, вполне могут размещаться в личной библиотеке.

Временная библиотека

Временные библиотеки (*temporary libraries*) представляют собой секционированные наборы данных, созданные для хранения программы до ее использования на последующем шаге *того же* задания. Создание и удаление временной библиотеки выполняется в задании.

При тестировании новой программы временная библиотека особенно полезна для сохранения загрузочного модуля, сгенерированного редактором связей, до его выполнения на следующем шаге задания. Так как модуль не будет нужен другим заданиям до его полного тестирования, его не следует сохранять в личной библиотеке или системной библиотеке. В Примере 10.13 на этапе LKED создается временная библиотека &&LOADMOD оператором SYSLMOD DD. На шаге запуска программы (GO) выполняется обращение к тому же временному набору данных командой:

```
//GO EXEC PGM=*.LKED.SYSLMOD,....
```

10.5 Обзор этапов от компиляции до запуска

На рис. 10.4 представлена связь между объектным и загрузочным модулем, сохраненном в загрузочной библиотеке и загружаемом в основную память для выполнения.

Мы начинаем с двух программ, A и B, компилируемых в два объектных модуля. Затем два объектных модуля компонуются в один загрузочный модуль MYPROG, который сохраняется в загрузочной библиотеке в хранилище с прямым доступом. Затем загрузочный модуль MYPROG загружается в основную память загрузчиком программ, после чего ему передается управление для выполнения.

10.6 Использование процедур

В целях экономии времени и во избежание ошибок можно подготовить наборы операторов управления заданием и поместить их в секционированный набор данных (PDS) или расширенный секционированный набор данных (PDSE), называемый *библиотекой процедур* (*procedure library*). Его можно использовать, например, для компиляции, ассемблирования, компоновки и выполнения программы, как показано в Примере 10.13. Дополнительные сведения о JCL-процедурах см. в разделе 6.7 «JCL-процедуры (PROC)».

Библиотека процедур представляет собой библиотеку, содержащую процедуры. Набор операторов управления заданием в библиотеке системных процедур SYS1.

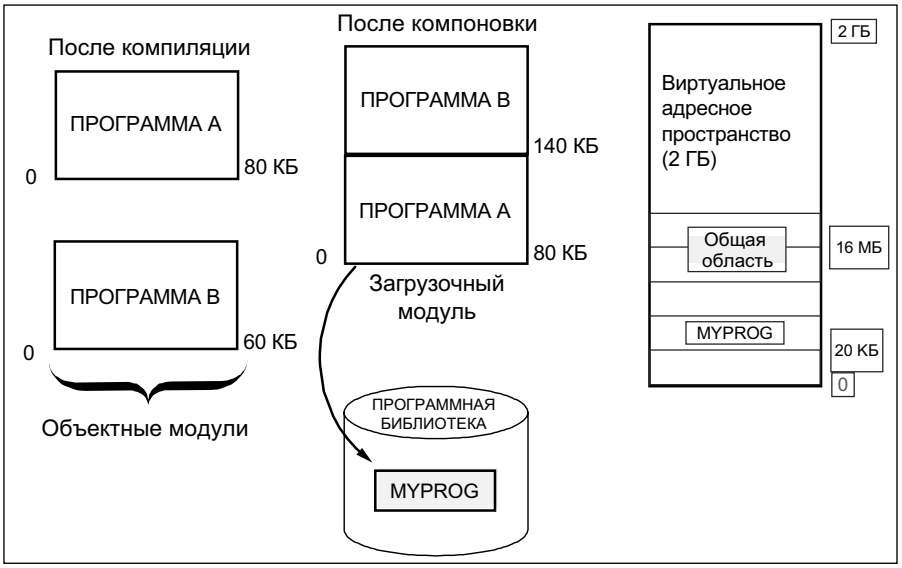


Рис. 10.4. Компиляция, компоновка и выполнение программы

PROCLIB (или в библиотеке процедур, определенной при инсталляции) называется *каталогизированной процедурой* (*cataloged procedure*; библиотека SYS1.PROCLIB представлена в разделе 6.10 «Системные библиотеки»)

Для тестирования процедуры перед ее сохранением в библиотеке процедур следует добавить процедуру во входной поток и выполнить ее; процедура во входном потоке называется *встраиваемой процедурой* (*inline procedure*). В любом задании максимальное количество встраиваемых процедур равно 15. Для того чтобы протестировать процедуру во входном потоке, она должна завершаться оператором конца процедуры (PEND). Оператор PEND обозначает конец процедуры (PROC). Он нужен только когда процедура является встраиваемой. В библиотеке процедур не требуется использовать оператор PEND.

Встраиваемая процедура должна находиться в том же задании до вызывающего ее оператора EXEC.

Пример 10.14. Образец определения процедуры

```
//DEF      PROC   STATUS=OLD,LIBRARY=SYSLIB,NUMBER=777777
//NOTIFY   EXEC   PGM=ACCUM
//DD1      DD     DSNAME=MGMT,DISP=(&STATUS,KEEP),UNIT=3400-6,
//           VOLUME=SER=888888
//DD2      DD     DSNAME=&LIBRARY,DISP=(OLD,KEEP),UNIT=3390,
//           VOLUME=SER=&NUMBER
```

В каталогизированной процедуре, представленной в Примере 10.14, определены три символических параметра: &STATUS, &LIBRARY и &NUMBER. Значения присваиваются символическим параметрам в операторе PROC. Эти значения используются

в том случае, если процедура вызвана, но значения символических параметров в вызывающем операторе EXEC не присвоены.

В Примере 10.15 выполняется тестирование процедуры DEF. Обратите внимание на то, что эта процедура заключена в операторы PROC и PEND. Оператор EXEC, следующий за процедурой DEF, обращается к вызываемой процедуре. В данном случае, так как имя DEF соответствует составленной выше встраиваемой процедуре, система будет использовать эту процедуру, не выполняя поиск в других местах.

Пример 10.15. Тестирование встраиваемой процедуры

```
//TESTJOB  JOB ....
//DEF      PROC   STATUS=OLD,LIBRARY=SYSLIB,NUMBER=777777
//NOTIFY   EXEC   PGM=ACCUM
//DD1      DD     DSNAME=MGMT,DISP=(&STATUS,KEEP),UNIT=3400-6,
//                VOLUME=SER=888888
//DD2      DD     DSNAME=&LIBRARY,DISP=(OLD,KEEP),UNIT=3390,
//                VOLUME=SER=&NUMBER
//         PEND
//*
//TESTPROC EXEC   DEF
//
```

10.7 Заключение

В этой главе описывается процесс трансляции исходной программы в исполняемый загрузочный модуль и выполнение загрузочного модуля. Основными этапами такой трансляции являются компиляция и компоновка, хотя может использоваться и третий этап прекомпиляции исходного кода до его компиляции. Этап прекомпиляции требуется, если исходная программа выдает вызовы на командном языке CICS или SQL-вызовы. Выходные данные этапа прекомпиляции затем передаются на этап компиляции.

Цель этапа компиляции состоит в том, чтобы проверить и преобразовать исходный код в перемещаемый машинный язык в виде объектного кода. Несмотря на то, что объектный код написан на машинном языке, он еще не является исполняемым. Он должен быть обработан редактором связей, компоновщиком или загрузчиком до его выполнения.

Редактор связей, компоновщик и загрузчик в качестве входных данных принимают объектный код и другие загрузочные модули, а затем генерируют исполняемый загрузочный модуль, а в случае загрузчика и исполняет его. Этот процесс разрешает любые неразрешенные ссылки в объектном коде и обеспечивает включение всего, что нужно для выполнения этой программы, в итоговый загрузочный модуль. Загрузочный модуль теперь готов к выполнению.

Для выполнения загрузочного модуля его необходимо загрузить в основную память. Компоновщик или менеджер программ загружает модуль в память и передает

туда управление, чтобы начать его выполнение. При передаче управления модулю ему назначается адрес запуска программы в памяти. Так как адресация инструкций и данных программы осуществляется с использованием базового адреса и смещения, этот начальный адрес обеспечивает адресуемость инструкций и данных в пределах диапазона смещения[1].

Основные термины в этой главе

Компоновщик	Copybook	Редактор связей
Загрузочный модуль	Объектный модуль	Объектно-ориентированный код
Процедура	Библиотека процедур	Программная библиотека
Перемещаемый модуль	Исходный модуль	

10.8 Контрольные вопросы

Чтобы проверить понимание материала этой главы, ответьте на следующие вопросы:

1. Какие шаги нужны для выполнения исходной программы?
2. Как можно изменить объектный модуль? Загрузочный модуль?
3. Сколько разных типов загрузочных библиотек может иметь система?
4. Что такое библиотека процедур, и для чего она используется?
5. В чем различие между редактором связей и компоновщиком?
6. В чем сходство между copybook и каталогизированными библиотеками процедур?
7. Каково назначение компилятора? Каковы его входные и выходные данные?
8. Что означает термин «перемещаемый модуль»?
9. В чем различие между объектным и загрузочным модулем?
10. Для чего используется оператор SYSLMOD DD?
11. Почему использование оператора PEND является обязательным во встраиваемой процедуре (PROC), и не является обязательным в каталогизированной процедуре (PROC)?

10.9 Упражнения

Практические упражнения в этой главе помогают развить навыки подготовки программ к запуску в z/OS. Эти навыки необходимы для выполнения практических упражнений в остальной части книги.

Для выполнения практических упражнений вам или вашей подгруппе нужен идентификатор (yourid) и пароль пользователя TSO (для этого следует обратиться к преподавателю).

Упражнения посвящены следующим темам:

- «Упражнение: компиляция и компоновка программы».
- «Упражнение: выполнение программы».

[1] Максимальное смещение для каждого базового регистра составляет 4096 байт (4 КБ). Любая программа размером больше 4 КБ должна иметь больше одного базового регистра, чтобы обеспечить адресуемость для всей программы.

10.9.1 Упражнение: компиляция и компоновка программы

В этом разделе используется по меньшей мере два языка программирования для компиляции и компоновки; см. JCL-код в:

```
yourid.LANG.CNTL(language)
```

где language может принимать значения ASM, ASMLE, C, C2, COBOL, COBOL2, PL1, PL12.

Выполните это упражнение, прежде чем переходить к упражнению из раздела 10.9.2, «Упражнение: выполнение программы». Результатом успешного выполнения каждого задания в этом упражнении будет создание загрузочных модулей, запускаемых в следующем упражнении.

Примечание. Необходимо изменить JCL-код, чтобы задать старший квалификатор (HLQ) студента, выполняющего задания. Кроме того, может потребоваться изменить все задания, обращающиеся к наборам данных Language Environment. Дополнительные сведения см. в комментариях.
Для передачи на выполнение заданий введите SUBMIT в командной строке ISPF. После завершения задания вам потребуется использовать SDSF для просмотра выходных данных задания.

1. Передайте на выполнение следующий набор данных для компиляции и компоновки сложной программы на ассемблере:

```
yourid.LANG.CNTL(ASMLE)
```

Примечание. Студенту может потребоваться изменить JCL-код для наборов данных, начинающихся с CEE. Спросите у своего системного программиста, какой старший квалификатор (HLQ) соответствует наборам данных Language Environment. JCL-код, который может потребоваться изменить, выделен ниже:

```
//C.SYSLIB DD DSN=SYS1.MACLIB,DISP=SHR
// DD DSN=CEE.SCEEMAC,DISP=SHR
//C.SYSIN DD DSN=ZUSER##.LANG.SOURCE(ASMLE),DISP=SHR
//L.SYSLMOD DD DSN=ZUSER##.LANG.LOAD(ASMLE),DISP=SHR
//L.SYSLIB DD DSN=CEE.SCEELKED,DISP=SHR
// DD DSN=CEE.SCEELKEX,DISP=SHR
```

2. Передайте на выполнение следующий набор данных для компиляции и компоновки простой программы на ассемблере:

```
yourid.LANG.CNTL(ASM)
```

3. Передайте на выполнение следующий набор данных для компиляции и компоновки сложной программы на языке C:

```
yourid.LANG.CNTL(C)
```

Примечание. Студенту может потребоваться изменить JCL-код для наборов данных, начинающихся с CEE и CBC. Спросите у своего системного программиста, какие старшие квалификаторы (HLQ) соответствуют наборам данных Language Environment и C. JCL-код, который может потребоваться изменить, выделен ниже:

```
//STEP1 EXEC PROC=EDCCB,LIBPRFX=CEE,LNGPRFX=CBC,
// INFILE='ZUSER##.LANG.SOURCE(C)',
// OUTFILE='ZUSER##.LANG.LOAD(C),DISP=SHR'
```

4. Передайте на выполнение следующий набор данных для компиляции и компоновки простой программы на языке C:

```
yourid.LANG.CNTL(C2)
```

Примечание. Студенту может потребоваться изменить JCL-код для наборов данных, начинающихся с CEE и CBC. Спросите у своего системного программиста, какие старшие квалификаторы (HLQ) соответствуют наборам данных Language Environment и C. JCL-код, который может потребоваться изменить, выделен ниже:

```
//STEP1 EXEC PROC=EDCCB,LIBPRFX=CEE,LNGPRFX=CBC,
// INFILE='ZUSER##.LANG.SOURCE(C2)',
// OUTFILE='ZUSER##.LANG.LOAD(C2),DISP=SHR'
```

5. Передайте на выполнение следующий набор данных для компиляции и компоновки сложной программы на языке COBOL:

```
yourid.LANG.CNTL(COBOL)
```

Примечание. Студенту может потребоваться изменить JCL-код для наборов данных, начинающихся с CEE. Спросите у своего системного программиста, какой старший квалификатор (HLQ) соответствует наборам данных Language Environment. JCL-код, который может потребоваться изменить, выделен ниже:

```
//SYSIN DD DSN=ZUSER##.LANG.SOURCE(COBOL),DISP=SHR
//COBOL.SYSLIB DD DSN=CEE.SCEESAMP,DISP=SHR
//LKED.SYSLMOD DD DSN=ZUSER##.LANG.LOAD(COBOL),DISP=SHR
```

6. Передайте на выполнение следующий набор данных для компиляции и компоновки простой программы на языке COBOL:

```
yourid.LANG.CNTL(COBOL2)
```

7. Передайте на выполнение следующий набор данных для компиляции и компоновки сложной программы на языке PL/I:

```
yourid.LANG.CNTL(PL1)
```

Примечание. Студенту может потребоваться изменить JCL-код для наборов данных, начинающихся с CEE. Спросите у своего системного программиста, какой старший квалификатор (HLQ) соответствует наборам данных Language Environment. JCL-код, который может потребоваться изменить, выделен ниже:

```
//SYSIN DD DSN=ZUSER##.LANG.SOURCE(PL1),DISP=SHR
//PLI.SYSLIB DD DSN=CEE.SCEESAMP,DISP=SHR
//BIND.SYSLMOD DD DSN=ZUSER##.LANG.LOAD(PL1),DISP=SHR
```

8. Передайте на выполнение следующий набор данных для компиляции и компоновки простой программы на языке PL/I:

yourid.LANG.CNTL(PL12)

10.9.2 Упражнение: выполнение программы

В этом разделе выбранные вами примеры на разных языках были скомпилированы и скомпонованы в упражнении 10.9.1 «Упражнение: компиляция и компоновка программы». Не пытайтесь запускать какое-либо из следующих заданий, если вы не выполнили предыдущее упражнение, так как при этом возникнут ошибки.

Следующее упражнение содержит действия, которые нужно выполнить в примерах на каждом языке для выполнения загрузочного модуля, ранее сохраненного при выполнении задания компиляции и компоновки. Для интерпретируемых языков потребуется выполнять исходные разделы непосредственно из:

yourid.LANG.SOURCE(language)

где language может принимать значение CLIST или REXX.

Примечание. Может потребоваться изменить JCL-код таким образом, чтобы задать HLQ студента, передающего задания. Для передачи на выполнение заданий введите SUBMIT в командной строке ISPF. После завершения задания потребуется использовать SDSF для просмотра выходных данных задания.
Для успешного выполнения этих заданий нужно выполнить задания компиляции и компоновки, приведенные в упражнении 10.9.1 «Упражнение: компиляция и компоновка программы», чтобы создать загрузочные модули в:

```
ZPROF.LANG.LOAD
```

Если эти задания не были успешно выполнены, в журнале задания, выводимом в SDSF, могут выдаваться ошибки, подобные следующим:

```
CSV003I REQUESTED MODULE ASM NOT FOUND
CSV028I ABEND806-04 JOBNAME=ZPROF2 STEPNAME=STEP1
IEA995I SYMPTOM DUMP OUTPUT 238
SYSTEM COMPLETION CODE=806 REASON CODE=00000004
```

Имя модуля, JOBNAME и STEPNAME соответствуют переданному на выполнение заданию.

1. Передайте на выполнение следующий набор данных для выполнения сложной программы на ассемблере:

   yourid.LANG.CNTL(USEASMLE)

   Этот пример осуществляет доступ к z/OS Language Environment и выводит сообщение:

   “IN THE MAIN ROUTINE”.

2. Передайте на выполнение следующий набор данных для выполнения простой программы на ассемблере:

   yourid.LANG.CNTL(USEASM)

   Этот пример устанавливает код завершения 15 и выполняет выход.

3. Передайте на выполнение следующий набор данных для выполнения сложной программы на языке C:

   yourid.LANG.CNTL(USEC)

   Этот пример выводит местную дату и время.

4. Передайте на выполнение следующий набор данных для выполнения простой программы на языке C:
   yourid.LANG.CNTL(USEC2)
   Этот пример выводит сообщение «HELLO WORLD».
5. Передайте на выполнение следующий набор данных для выполнения сложной программы на языке COBOL:
   yourid.LANG.CNTL(USECOBOL)
   Этот пример выводит местную дату и время.
6. Передайте на выполнение следующий набор данных для выполнения простой программы на языке COBOL:
   yourid.LANG.CNTL(USECOBO2)
   Этот пример выводит сообщение «HELLO WORLD».
7. Передайте на выполнение следующий набор данных для выполнения сложной программы на языке PL/I:
   yourid.LANG.CNTL(USEPL1)
   Этот пример выводит местную дату и время.
8. Передайте на выполнение следующий набор данных для выполнения простой программы на языке PL/I:
   yourid.LANG.CNTL(USEPL12)
   Этот пример выводит сообщение «HELLO WORLD».
9. Выполните следующую сложную программу на языке CLIST:
   yourid.LANG.SOURCE(CLIST)
   Этот пример запрашивает у пользователя старший квалификатор (HLQ), после чего создает форматированный листинг каталога для этого HLQ.
   В командной строке ISPF введите:
   TSO EX '*yourid*.LANG.SOURCE(CLIST)'
   При запросе введите HLQ *yourid*
10. Выполните следующую простую программу на языке CLIST:
    yourid.LANG.SOURCE(CLIST2)
    Этот пример выводит сообщение «HELLO WORLD».
    В командной строке ISPF введите:
    TSO EX '*yourid*.LANG.SOURCE(CLIST2)'
11. Выполните следующую сложную программу на языке REXX:
    yourid.LANG.SOURCE(REXX)
    Этот пример запрашивает у пользователя старший квалификатор (HLQ), после чего создает форматированный листинг каталога для этого HLQ.
    В командной строке ISPF введите:
    TSO EX '*yourid*.LANG.SOURCE(REXX)'
    При запросе введите HLQ *yourid*
12. Выполните следующую простую программу на языке REXX:
    yourid.LANG.SOURCE(REXX2)
    Этот пример выводит сообщение «HELLO WORLD».
    В командной строке ISPF введите:
    TSO EX '*yourid*.LANG.SOURCE(REXX2)'

Часть 3

Оперативная рабочая нагрузка в z/OS

В этой части мы рассмотрим основные категории оперативных или *интерактивных* задач, выполняемых в z/OS, таких как обработка транзакций, управление базами данных и веб-обработка. Следующие главы посвящены обсуждению сетевых коммуникаций и нескольких популярных программных продуктов промежуточного уровня, включая DB2, CICS и WebSphere.

11

Системы управления транзакциями в z/OS

Цель. Для того чтобы расширить свои знания о задачах мэйнфреймов, необходимо понимать роль мэйнфреймов в мире современных информационных технологий. В этой главе рассматриваются понятия и терминология обработки транзакций, а также представлен обзор основных типов системного программного обеспечения, используемого в обработке оперативных задач на мэйнфрейме. Эта глава сконцентрирована на двух наиболее часто используемых продуктов управления транзакциями для z/OS: CICS и IMS.

После завершения работы над этой главой вы сможете:

- описать роль больших систем в информационном бизнесе;
- перечислить общие атрибуты большинства систем обработки транзакций;
- описать роль CICS в обработке оперативных транзакций;
- описать CICS-программы, CICS-транзакции и CICS-задачи;
- объяснить, что такое диалоговое и псевдодиалоговое программирование;
- описать CICS и технологии обеспечения доступа через веб;
- описать компоненты IMS.

11.1 Оперативная обработка на мэйнфрейме

В предыдущих главах рассматривались средства пакетной обработки, однако они – не единственные приложения, работающие в z/OS и на мэйнфрейме. Как мы покажем в этой главе, в z/OS также выполняются оперативные приложения. Мы также рассмотрим, что такое оперативные (или интерактивные) приложения, и обсудим основные их элементы в мэйнфрейм-среде.

Мы рассмотрим базы данных, являющиеся общим способом хранения данных приложения. Базы данных облегчают разработку, особенно при использовании реляционной системы управления базами данных (РСУБД), освобождая программиста от задачи организации и управления данными. Далее в этой главе мы рассмотрим несколько широко используемых систем управления транзакциями для предприятий, использующих мэйнфреймы.

Начнем с примера туристического агентства с требованием, общим для многих клиентов мэйнфреймов: обеспечить немедленный доступ клиентов к службам и использовать преимущества Интернет-коммерции.

11.2 Пример глобальной оперативной обработки – новая большая картина

Крупное туристическое агентство использовало систему пакетной обработки на основе мэйнфрейма на протяжении многих лет. Многие годы клиенты агентства получали прекрасное обслуживание, и агентство постоянно совершенствовало свои системы.

В самом начале специалисты по информационным системам разработали несколько приложений для поддержки внутренних и внешних процессов агентства: информации о сотрудниках, информации о клиентах, контактов с компаниями, занимающимися прокатом автомобилей, гостиницах по всему миру, бронирования авиабилетов и т. д. Сначала эти приложения периодически обновлялись посредством пакетной обработки.

Однако эти данные не были статичными и подвергались частым изменениям. Так как цены, например, часто меняются, со временем стало намного сложнее поддерживать актуальность информации. Клиенты агентства хотели получить информацию сразу, а это было не всегда возможно из-за фиксированных интервалов пакетных обновлений (учитывая разницу во времени между Азией, Европой и Америкой).

Выполнение обработки этих задач посредством традиционных пакетных заданий мэйнфреймов означало бы некоторую задержку во времени между получением изменения и фактическим обновлением. Агентству требовался способ обновления небольших объемов данных, передаваемых фрагментами по телефону, факсу или электронной почте, одновременно с возникновением изменений (рис. 11.1).

Таким образом, работающие в агентстве специалисты по информационным технологиям создали несколько новых приложений. Так как изменения требуется немедленно передавать конечным пользователям приложений, новые приложения по существу являются транзакционными. Приложения называются транзакционными или интерактивными, так как изменения в системных данных сразу же вступают в действие.

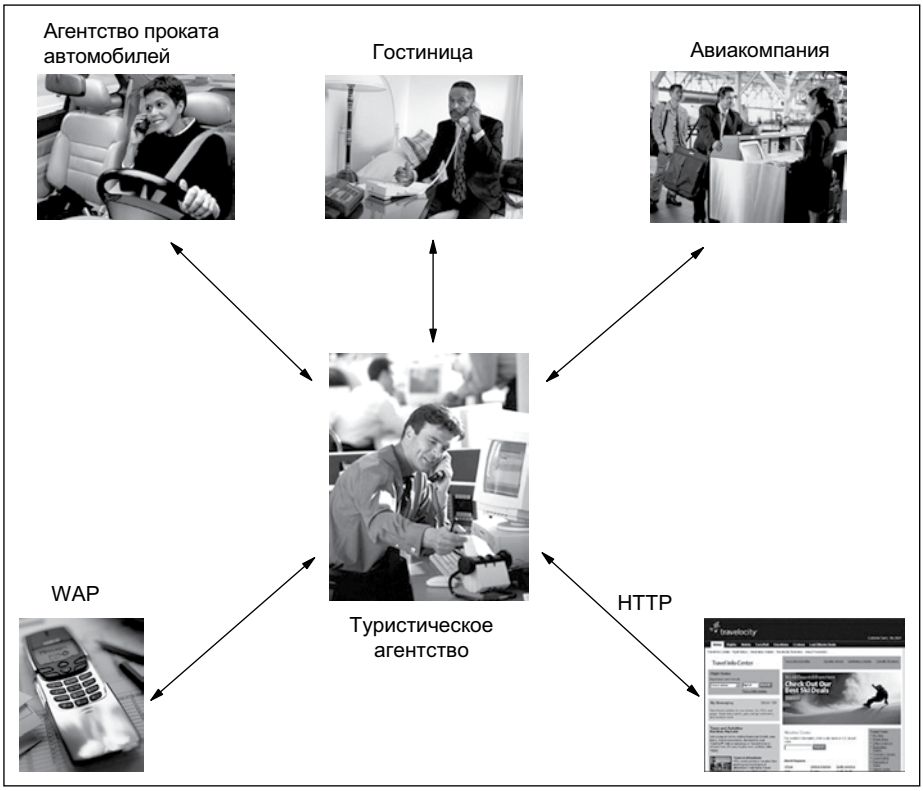


Рис. 11.1. Практический пример

Туристическое агентство связалось со своими поставщиками, чтобы узнать, что можно сделать. Необходимо было реализовать способ взаимодействия между компьютерами. Некоторые авиакомпании также работали с мэйнфреймами, тогда как другие с ними не работали, и все хотели сохранить свои приложения.

В конечном итоге решение было найдено. Связь стала простой: нужно было просто задать вопрос и через несколько секунд получить результат – отлично!

Дополнительные инновации потребовались в связи с тем, что клиенты тоже изменились. В их домах появились персональные компьютеры, и им захотелось иметь возможность просматривать предлагаемые варианты через Интернет. Некоторые клиенты начали использовать для доступа свои мобильные телефоны.

11.3 Транзакционные системы мэйнфрейма

Мы сталкиваемся с транзакциями постоянно, например, при обмене денег на товары и услуги или при выполнении поиска в Интернете. Транзакция представляет своего рода обмен, обычно включающий запрос и ответ, происходящий как типичное событие при выполнении повседневных операций организации.

Транзакции имеют следующие свойства:

- небольшой объем данных, обрабатываемых и передаваемых за одну транзакцию;
- большое количество пользователей;
- выполняются в больших количествах.

11.3.1 Что такое транзакционные программы?

Бизнес-транзакция представляет собой автономную коммерческую операцию. Некоторые транзакции включают короткий диалог (например, транзакция изменения адреса). Другие транзакции включают множество действий, выполняемых в течение длительного периода времени (например, регистрация заказа тура, включая прокат автомобиля, бронирование номера в гостинице и бронирование авиабилетов).

Одна транзакция может состоять из нескольких приложений, выполняющих определенную обработку. Крупномасштабные транзакционные системы (такие как IBM CICS) используют многозадачные и многопоточные возможности z/OS, позволяющие обрабатывать больше одной задачи одновременно, где каждая задача сохраняет свои переменные данные и отслеживает инструкции, выполняемые каждым пользователем.

Многозадачность является важным свойством любой среды, в которой тысячи пользователей могут быть подключены одновременно. Когда многозадачная транзакционная система получает запрос на выполнение транзакции, она может запустить новую задачу, связанную с *одним экземпляром* выполнения транзакции; другими словами, с одним сеансом выполнения транзакции с определенным набором данных, обычно осуществляемым от имени определенного пользователя определенного терминала. Задачу можно рассматривать как аналог потока в UNIX. По завершении транзакции задача завершается.

Многопоточность позволяет осуществлять обработку одной копии приложения несколькими транзакциями одновременно. Многопоточность требует, чтобы все транзакционные приложения были реентерабельными; другими словами, они должны допускать последовательное многократное использование между точками входа и выхода. В языках программирования реентерабельность обеспечивается *обновленной* копией рабочей области памяти, получаемой при каждом вызове программы.

Многопоточность – одна копия приложения может одновременно обрабатываться несколькими транзакциями

11.3.2 Что такое транзакционная система?

На рис. 11.2 представлены основные свойства транзакционной системы. До появления Интернета каждая транзакционная система обслуживала сотни или тысячи *терминалов*, обрабатывая десятки или сотни транзакций в секунду. Такая нагрузка была достаточно предсказуемой с точки зрения количества и состава транзакций.

Транзакция – единица работы, выполняемая одной или несколькими транзакционными программами, включающими определенный набор входных данных и инициирующими определенный процесс или задание

Рис. 11.2. Свойства транзакционной системы

Транзакционные системы должны быть способны поддерживать большое количество одновременно работающих пользователей и типов транзакций.

Одно из основных свойств транзакционной или оперативной системы состоит в том, что взаимодействия между пользователем и системой очень кратковременные. Большинство транзакций выполняются за короткий промежуток времени – в некоторых случаях, за одну секунду. Пользователь выполняет полную бизнес-транзакцию как серию коротких взаимодействий, и каждое взаимодействие требует немедленного времени реагирования. Эти приложения являются критически важными; поэтому необходимо обеспечивать непрерывную доступность, высокую производительность, а также защиту и целостность данных.

Обработка оперативных транзакций (online transaction processing, OLTP) представляет собой обработку транзакций, выполняемую интерактивно; она требует:

- немедленного времени реагирования;
- непрерывной доступности транзакционного интерфейса для конечного пользователя;
- безопасности;
- целостности данных.

Оперативные транзакции знакомы большинству людей. Приведем следующие примеры:

- Транзакции при работе с банкоматом, в частности депонирование, снятие денег, проверка состояния счета и перевод денег.
- Оплата в супермаркете дебетовой или кредитной картой.
- Покупка товаров через Интернет.

  В действительности, оперативная система во многом выполняет те же функции, что и операционная система:
- управление и распределение задач;
- управление полномочиями доступа пользователей к системным ресурсам;
- управление использованием памяти;
- управление и контроль над одновременным доступом к файлам данных;
- обеспечение независимости устройств.

11.3.3 Каковы типичные требования транзакционной системы?

В транзакционной системе транзакции должны соответствовать четырем основным требованиям (на английском языке их первые буквы составляют слово «ACID»):

- Атомарность (**A**tomicity). Процессы, выполняемые транзакцией, либо выполняются полностью, либо не выполняются вообще.
- Согласованность (**C**onsistency). Транзакция должна работать только с согласованной информацией.
- Изоляция (**I**solation). Процессы, вызываемые двумя или более транзакциями, должны быть изолированы друг от друга.
- Долговечность (**D**urability). Изменения, внесенные транзакцией, должны быть постоянными.

Обычно транзакции инициируются конечным пользователем, взаимодействующим с транзакционной системой через терминал. В прошлом транзакционные системы поддерживали только терминалы и устройства, подключенные через сеть телеобработки. В настоящее время транзакционные системы могут обслуживать запросы, переданные любым из следующих способов:

- с веб-страницы;
- из программы на удаленной рабочей станции;
- из приложения на другой транзакционной системе;
- автоматический вызов в заданное время.

11.3.4 Что такое фиксация и откат?

В транзакционной системе термины *фиксация* (*commit*) и *откат* (*rollback*) относятся к набору действий, обеспечивающих либо внесение приложением *всех* изменений в ресурсах, представленных одной единицей восстановления (unit of recovery, UR), либо невнесение каких-либо изменений вообще. Протокол двухфазовой фиксации

обеспечивает фиксацию и откат. Он контролирует внесение всех изменений или невнесение каких-либо изменений даже при отказе одного из элементов (например, приложения, системы или менеджера ресурсов). Протокол позволяет выполнять перезапуск или восстановление после отказа системы или подсистемы.

Протокол двухфазовой фиксации инициируется, когда приложение готово выполнить фиксацию или откат вносимых им изменений. В этот момент координирующий менеджер восстановления, также называемый *менеджер точки синхронизации* (*syncpoint manager*), дает каждому менеджеру ресурсов, задействованному в единице восстановления, возможность сообщить, находится ли его часть единицы восстановления в согласованном состоянии, и можно ли ее зафиксировать. Если все участники сообщат «да», менеджер восстановления даст указание всем менеджерам ресурсов зафиксировать изменения. Если какой-то из участников сообщит «нет», менеджер восстановления даст указание выполнить откат изменений. Этот процесс обычно представляется в виде двух фаз.

На фазе 1 приложение передает запрос на создание точки синхронизации или откат координатору точки синхронизации. Координатор передает команду PREPARE для отправки первоначального потока создания точки синхронизации всем UR агентам менеджеров ресурсов. В ответ на команду PREPARE, каждый менеджер ресурсов, участвующий в транзакции, отвечает координатору точки синхронизации, готов ли он к фиксации или нет.

Когда координатор точки синхронизации получает все ответы от всех своих агентов, инициируется фаза 2. На этой фазе координатор точки синхронизации выдает команду фиксации или отката в зависимости от предыдущих ответов. Если какой-либо из агентов дал отрицательный ответ, инициатор точки синхронизации сообщает *всем* агентам точки синхронизации, что нужно выполнить откат изменений.

Момент, в который координатор регистрирует факт своего намерения сообщить всем менеджерам ресурсов о необходимости фиксации или отката называется *атомарным моментом* (*atomic instant*). Независимо от того, произойдет ли какой-либо отказ после этого момента, координатор предполагает, что произошла фиксация либо откат всех изменений. Координатор точки синхронизации обычно регистрирует решение на этом этапе. В случае аварийного завершения какого-либо участника после атомарного момента менеджер ресурсов, выполнивший аварийное завершение, после перезапуска должен, взаимодействуя с координатором точки синхронизации, завершить все операции фиксации или отката, выполнявшиеся на момент аварийного завершения.

В z/OS главный координатор точки синхронизации имеет название Resource Recovery Services (RRS). Кроме того, менеджер транзакций CICS производства компании IBM содержит собственный встроенный координатор точки синхронизации.

Во время первой фазы протокола агенты не знают, будет ли координатор точки синхронизации выполнять фиксацию или откат изменений. Этот промежуток времени называется *периодом неоднозначного состояния* (*in-doubt*). Единица восстановления (UR) имеет определенное состояние, зависящее от того, на каком этапе двухфазового процесса она находится.

- До того как UR внесет какие-либо изменения в ресурс, она называется *отключенной* (*In-reset*).

- После того как UR запросит изменения в ресурсах, она называется *начатой* (*In-flight*).
- После запроса фиксации (фаза 1), она называется *готовящейся* (*In-prepare*).
- После принятия менеджером точки синхронизации решения о фиксации (фаза 2 процесса двухфазовой фиксации), она называется находящейся в процессе *фиксации* (*In-commit*).
- Если менеджер точки синхронизации решит выполнить откат, она называется находящейся в процессе откатки (*In-backout*).

На рис. 11.3 изображен процесс двухфазовой фиксации.

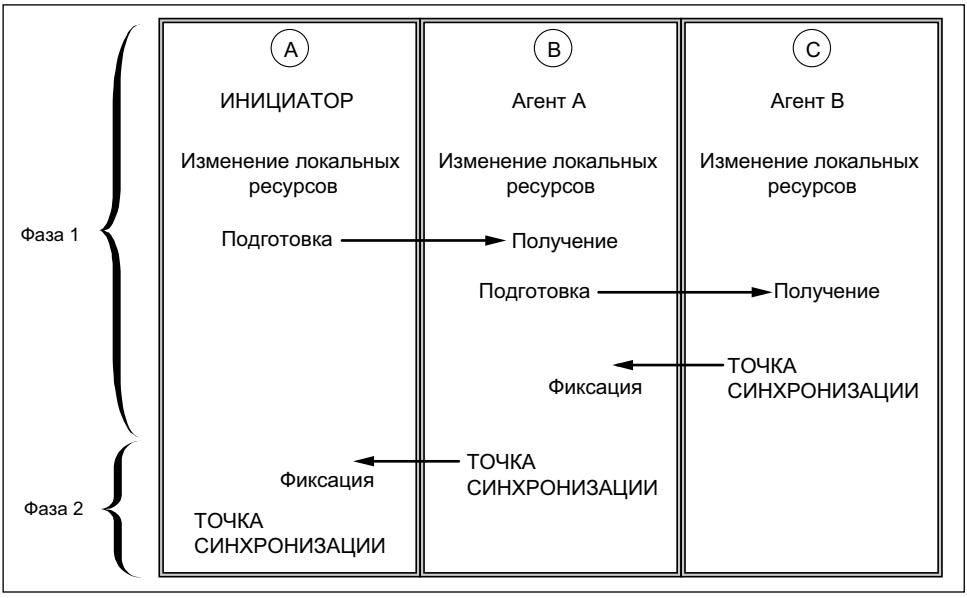


Рис. 11.3. Двухфазовая фиксация

Наиболее часто используемые в z/OS системы управления транзакциями, такие как CICS или IMS, поддерживают протоколы двухфазовой фиксации. Например, CICS поддерживает полную двухфазовую фиксацию транзакций, используя IMS и СУБД DB2, и поддерживает двухфазовую фиксацию в распределенных системах CICS.

Существует множество ограничений, налагаемых на разработчиков приложений, пытающихся создавать новые приложения, требующие обновления в нескольких различных менеджерах ресурсов, возможно, во множестве систем. Многие из этих новых приложений используют такие технологии, как хранимые процедуры DB2 и Enterprise Java Beans, а также используют средства подключения клиентов CICS или IMS, не поддерживающие двухфазовой фиксации. Если какой-либо из этих менеджеров ресурсов используется приложением для изменения ресурсов, невозможно использовать глобальный координатор для точки синхронизации.

Отсутствие глобального координатора точки синхронизации может повлиять на структуру приложения по следующим причинам:

- Приложение не может использовать сложные и распределенные транзакции, если в протоколе двухфазовой фиксации участвуют не все менеджеры ресурсов.
- Приложение нельзя спроектировать как единое приложение (или единицу восстановления) на нескольких системах (кроме CICS).

При программировании программисту приложений необходимо учитывать эти ограничения. Например, программист может ограничить варианты размещения бизнес-данных, чтобы обеспечить передачу всех данных в одной единице восстановления.

Кроме того, эти ограничения могут повлиять на восстанавливаемость защищенных ресурсов или их целостность в случае отказа одного из компонентов, так как менеджеры ресурсов не смогут выполнить ни фиксацию, ни откат изменений.

11.4 Что такое CICS?

CICS является сокращением от Customer Information Control System. Она представляет собой универсальную подсистему обработки транзакций для операционной системы z/OS. CICS обеспечивает выполнение приложения в оперативном режиме по запросу в то же время, когда множество других пользователей передают запросы на запуск тех же приложений, используя те же файлы и программы.

CICS управляет совместным использованием ресурсов, целостностью данных и назначением приоритетов выполнения с быстрым реагированием. CICS осуществляет авторизацию пользователей, выделение ресурсов (основной памяти и циклов процессора) и передачу запросов к базам данных из приложения к соответствующему менеджеру баз данных (например, DB2). Можно сказать, что CICS работает подобно операционной системе z/OS и выполняет во многом такие же функции.

CICS-приложение (*CICS application*) представляет собой набор связанных программ, совместно выполняющих бизнес-операцию, например, обработку заказа тура или подготовку ведомости компании. CICS-приложения выполняются под управлением CICS, используя службы и интерфейсы CICS для доступа к программам и файлам.

CICS-приложения традиционно выполняются путем передачи запроса транзакции. Выполнение транзакции представляет собой выполнение одного или нескольких приложений, реализующих требуемую функцию. В документации по CICS иногда CICS-приложения называются просто «программами», и иногда термин «транзакция» используется для обозначения обработки, выполняемой приложениями.

CICS-приложения могут принимать вид компонентов Enterprise Java Beans. Дополнительные сведения об этой разновидности программирования см. в разделе Java Applications in CICS информационного центра CICS Information Center.

11.4.1 CICS в системе z/OS

В системе z/OS CICS обеспечивает уровень функций управления транзакциями, тогда как операционная система остается конечным интерфейсом с аппаратным обеспечением компьютера. CICS по существу отделяет определенный тип приложений (а именно оперативные приложения) от других приложений в системе и обслуживает эти программы.

Когда приложение осуществляет доступ к терминалу или любому устройству, оно не может взаимодействовать с ним напрямую. Программа выдает команды для связи с системой CICS, которая взаимодействует с требуемыми методами доступа операционной системы. И, наконец, метод доступа взаимодействует с терминалом или устройством.

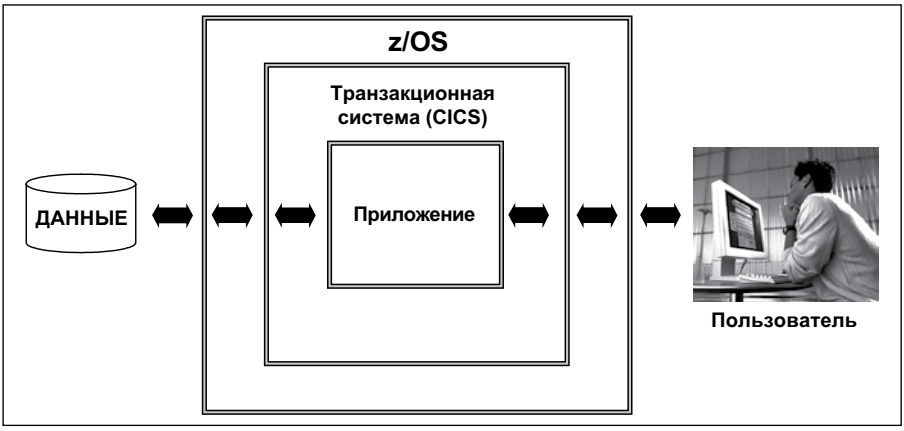


Рис. 11.4. Транзакционная система и операционная система

Система z/OS может содержать несколько копий CICS, выполняющихся одновременно. Каждая копия CICS запускается в отдельном адресном пространстве z/OS. CICS содержит опцию работы с несколькими регионами (multi-region operation, MRO), которая позволяет отделять разные функции CICS в различные CICS-регионы (адресные пространства); поэтому определенное адресное пространство CICS (или несколько адресных пространств) может осуществлять управление терминалами и будет называться регионом-владельцем терминала (terminal owning region, TOR). К прочим опциям относятся регионы-владельцы приложений (application-owning regions, AOR) и регионы-владельцы файлов (file-owning regions, FOR).

11.4.2 CICS-программы, транзакции и задачи

CICS позволяет отделить логику приложения от ресурсов приложения. Для разработки и запуска CICS-приложений необходимо понимать связь между CICS-программами, транзакциями и задачами. Эти термины используются в публикациях по CICS и во многих командах:

- Транзакция.

  Транзакция представляет собой фрагмент обработки, инициируемый одним запросом. Обычно он поступает от конечного пользователя с терминала, но может поступать и с веб-страницы, с программы на удаленной рабочей станции, от приложения в другой CICS-системе или автоматически вызываться в заданное время. Различные способы запуска CICS-транзакций описываются в руководствах *CICS Internet Guide* и *CICS External Interfaces Guide*.

  CICS-транзакция имеет 4-символьное имя, заданное в таблице управления программой (program control table, PCT).

- Приложение.
  Одна транзакция содержит одно или несколько *приложений*, которые при запуске осуществляют требуемую обработку.
  Однако термин *транзакция* используется в CICS для обозначения как одного события, так и для всех остальных транзакций одного типа. В CICS каждый тип транзакций описывается в *определении ресурсов транзакций* (*transaction resource definition*). Это определение назначает имя типа транзакции (идентификатор транзакции; TRANSID) и сообщает CICS некоторые сведения о выполняемой работе, в частности, о том, какую программу следует вызвать первой, и о том, какой тип аутентификации требуется при выполнении транзакции.
  Запуск транзакции осуществляется путем передачи ее TRANSID в CICS. CICS использует информацию, записанную в определении TRANSACTION, для установления требуемой среды выполнения и запускает первую программу.
- Единица работы.
  Термин «транзакция» в настоящее время широко используется в отрасли информационных технологий для описания единицы восстановления, называемой в CICS *единицей работы* (*unit of work*). Обычно этим термином обозначается полная восстанавливаемая операция; ее фиксация или откат может являться результатом программируемой команды или отказа системы. Во многих случаях областью действия CICS-транзакции также является единая единица работы, однако следует помнить о различных значениях этого слова при чтении публикаций, не посвященных CICS.

  Единица работы – транзакция; полная восстанавливаемая операция
- Задача.
  Кроме того, в публикациях, посвященных CICS, часто используется слово *задача* (*task*). Это слово также имеет в CICS особое значение. Когда CICS получает запрос на выполнение транзакции, она может запустить новую задачу, связанную с *одним экземпляром* выполнения транзакции, т. е. с одним сеансом выполнения транзакции с определенным набором данных обычно от имени определенного пользователя определенного терминала. Задачу можно рассматривать как аналог *потока*. По завершении транзакции задача завершается.

11.4.3 Использование языков программирования

Для создания CICS-приложений для z/OS можно использовать COBOL, OO COBOL, C, C++, Java, PL/I или ассемблер. Большая часть логики обработки выражается операторами стандартных языков, но для запроса CICS-операций используются CICS-команды или библиотеки классов на Java и C++.

Большую часть времени используется интерфейс программирования на уровне CICS-команд, EXEC CICS. Это относится к программам на языках COBOL, OO COBOL, C, C++, PL/I и ассемблере. Эти команды подробно определены в руководстве *CICS Application Programming Reference*.

Программирование на языке Java с библиотекой классов JCICS описано в разделе Java Applications in CICS информационного центра CICS Information Center.

Программирование на языке C++ с классами CICS C++ описывается в документации CICS C++ OO Class Libraries.

11.4.4 Диалоговое и псевдодиалоговое программирование

В CICS вход выполняемых программ в режим диалога с пользователем называется *диалоговой транзакцией* (*conversational transaction*, см. также рис. 11.5). В отличии от нее недиалоговая транзакция (non-conversational transaction, см. также рис. 11.6) обрабатывает один блок входных данных, отвечает и завершается (исчезает). Она никогда не приостанавливается для чтения второго блока входных данных с терминала, поэтому настоящего диалога не происходит.

Диалоговая транзакция – программа, выполняющая диалог с пользователем

В CICS существует технология, называемая псевдодиалоговой обработкой, при которой набор недиалоговых транзакций создает (с точки зрения пользователя) видимость единой диалоговой транзакции. Пока ожидается ввод, транзакция не существует; CICS осуществляет чтение входных данных, когда пользователь их отправляет. На рис. 11.5 и 11.6 показаны различные типы диалогов на примере обновления записи банковского счета.

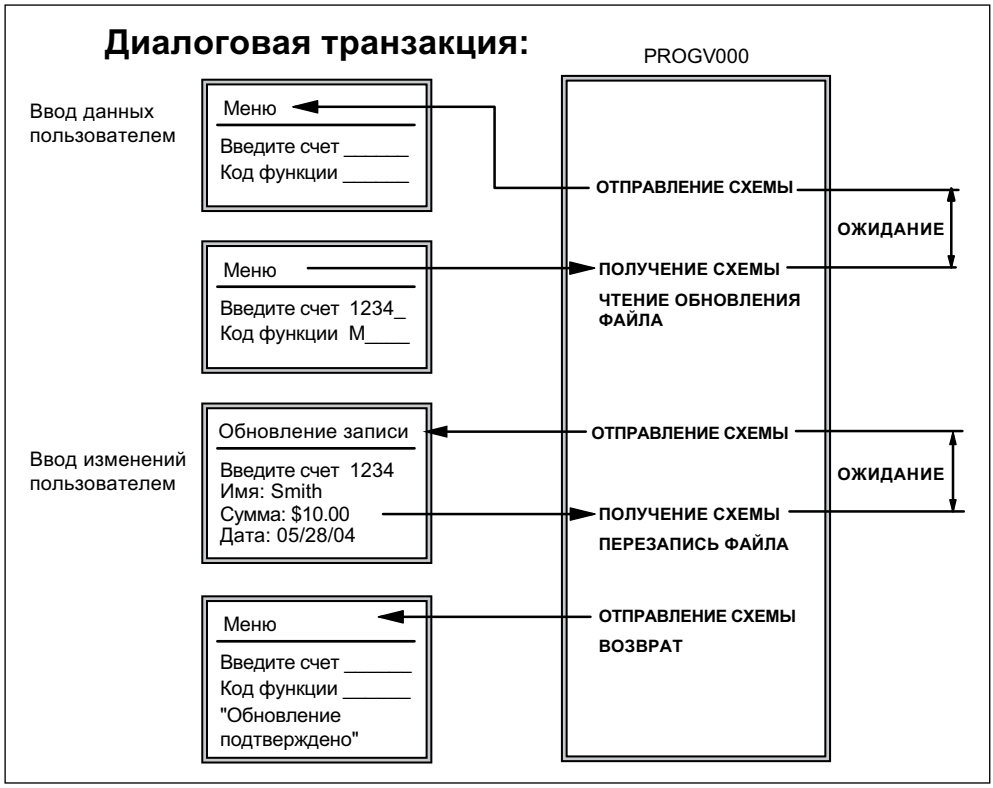


Рис. 11.5. Пример диалоговой транзакции

В диалоговой транзакции программы удерживают ресурсы во время ожидания получения данных. В псевдодиалоговой модели во время ожидания ресурсы не удерживаются (см. рис. 11.6).

Дополнительные сведения по этим вопросам см. в руководстве *CICS Application Programming Guide*.

Псевдодиалоговая транзакция – набор недиалоговых транзакций, создающий видимость диалога

Рис. 11.6. Пример псевдодиалоговой транзакции

11.4.5 Команды программирования CICS

Общий формат CICS-команды: EXECUTE CICS (или EXEC CICS), после чего идет имя команды и, возможно, одна или несколько опций.

Можно разработать множество приложений, используя командный интерфейс CICS, не имея каких-либо знаний или информации по полям управляющих блоков и областей памяти CICS. Однако может потребоваться получать информацию, действительную вне локальной среды приложения.

При необходимости использования системного сервиса CICS, например, при чтении записи из файла, нужно лишь включить CICS-команду в свой код. Например, в COBOL команды CICS имеют следующий вид:

```
EXEC CICS function option option ... END-EXEC.
```

где `function` обозначает требуемое действие. Чтение файла осуществляется функцией READ, запись в терминал – функцией SEND и т. д.

`option` – некоторая спецификация, связанная с функцией. Опции выражаются ключевыми словами. Например, опции команды READ включают FILE, RIDFLD, UPDATE и т. д. FILE сообщает CICS, чтение какого файла требуется выполнить, после чего всегда следует значение, указывающее имя файла. RIDFLD (record identification field – поле идентификации записи, т. е. ключ) сообщает CICS, из какой записи нужно значение. Опция UPDATE, с другой стороны, просто означает намерение изменить запись и не принимает какого-либо значения. Поэтому для того чтобы выполнить чтение с намерением изменения, записи из файла, известного в CICS как ACCTFIL, используя ключ, сохраненный в рабочей памяти как ACCTC, мы применили команду, представленную в Примере 11.1.

Пример 11.1. Пример CICS-команды

```
EXEC CICS
READ FILE('ACCTFIL')
RIDFLD(ACCTC) UPDATE ...
END-EXEC.
```

Для доступа к такой информации можно использовать команды ADDRESS и ASSIGN. Сведения об использовании этих команд в программировании см. в руководстве *CICS Application Programming Reference*. При применении команд ADDRESS и ASSIGN, может выполняться чтение разных полей, однако их не следует задавать или использовать каким-либо другим способом. Это означает, что вам не следует использовать какие-либо поля CICS как аргументы CICS-команд, так как эти поля можно изменить интерфейсными модулями EXEC.

11.4.6 Выполнение CICS-транзакции

Чтобы начать оперативный сеанс с CICS, пользователи обычно начинают с «регистрации» процесса, идентифицирующего их в CICS. Регистрация в CICS дает пользователям полномочия вызывать некоторые транзакции. После регистрации пользователи вызывают определенную транзакцию, которую они намереваются использовать. CICS-транзакция обычно идентифицируется идентификатором транзакции (TRANSID) длиной от 1 до 4 символов, который определяется в таблице, указывающей первоначальную программу, используемую для обработки транзакции.

Приложения хранятся в библиотеке на устройстве хранения с прямым доступом (DASD), подключенном к процессору. Их можно загрузить при запуске системы или по требованию. Если программа находится в памяти и не используется, CICS может освободить пространство для других целей. Когда программа требуется в следующий раз, CICS загружает ее актуальную копию из библиотеки.

За время обработки одной транзакции система может получить сообщения с нескольких терминалов. Для каждого сообщения CICS загружает приложение (если оно еще не загружено) и запускает задачу для его выполнения. Таким образом, несколько CICS-задач могут выполняться одновременно.

Многопоточность (*multithreading*) представляет собой технологию, позволяющую осуществлять обработку одной копии приложения несколькими транзакциями одновременно. Например, одна транзакция может начать выполнять приложение (турист запрашивает информацию). В это время другая транзакция может запустить такую же копию приложения (другой турист запрашивает информацию). Сравните это с *однопоточностью* (*single-threading*), предполагающей выполнение программы до ее завершения, когда требуется, чтобы обработка программы одной транзакцией должна быть завершена, прежде чем другая транзакция сможет использовать эту программу. Многопоточность требует, чтобы все CICS-приложения были квазиреентерабельными; другими словами, они должны допускать последовательное многократное использование между точками входа и выхода. CICS-приложения, использующие CICS-команды, следуют этому правилу автоматически.

CICS поддерживает отдельный поток управления для каждой задачи. Например, когда одна задача ожидает чтения дискового файла или получения ответа от терминала, CICS может передать управление другой задаче. Управление задачами осуществляет программа *управления задачами* CICS.

CICS управляет как многозадачностью, так и запросами обслуживания (операционной системой или CICS) от самих задач. Это позволяет продолжать CICS-обработку, тогда как задача ожидает выполнения запроса операционной системой от ее имени. Каждая транзакция, управляемая CICS, получает контроль над процессором в тот момент, когда эта транзакция имеет наивысший приоритет из всех транзакций, готовых к выполнению.

Во время выполнения транзакции приложение запрашивает различные средства CICS для поддержки передачи сообщений между ним и терминалом и для поддержки доступа к требуемым файлам или базам данных. После завершения работы приложения CICS возвращает терминал в режим ожидания. Чтобы лучше понять, что происходит, см. рис. 11.7, 11.8 и 11.9.

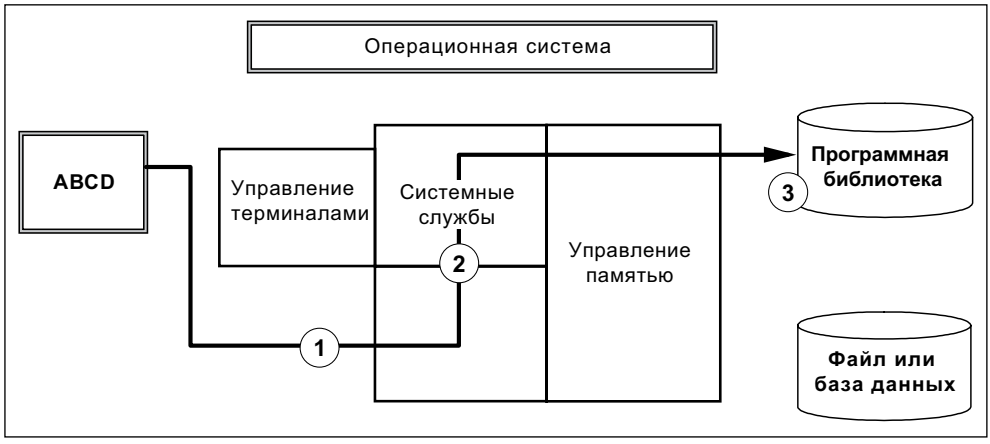


Рис. 11.7. Поток транзакций в CICS (часть 1)

Поток управления во время транзакции (код `ABCD`) представлен последовательностью чисел от 1 до 8 (эта транзакция используется только для того, чтобы показать некоторые задействованные этапы). Эти восемь этапов имеют следующие значения:

1. *Блок управления терминалами* принимает символы `ABCD`, введенные с терминала, и помещает их в рабочую память.
2. *Системные службы* интерпретируют код транзакции `ABCD` как обращение к приложению `ABCD00`. Если оператор терминала имеет полномочия вызова этой программы, она либо уже находится в памяти, либо загружается в память.
3. Модули переносятся из *программной библиотеки* в рабочую память.

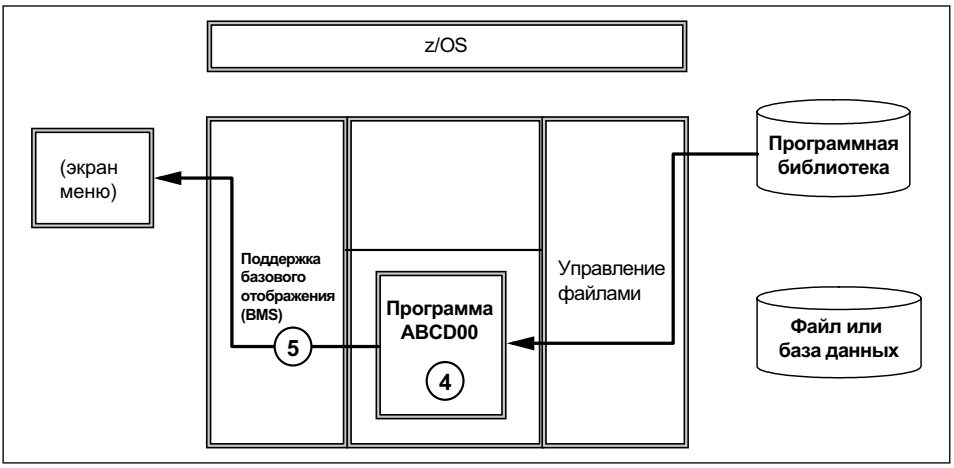


Рис. 11.8. Поток транзакций в CICS (часть 2)

4. Происходит создание *задачи*. Программе `ABCD00` передается управление от ее имени.
5. `ABCD00` вызывает модуль BMS (Basic Mapping Support, базовая поддержка отображения) и модуль управления терминалами для передачи меню на терминал, позволяя пользователю указать, какая именно информация нужна.

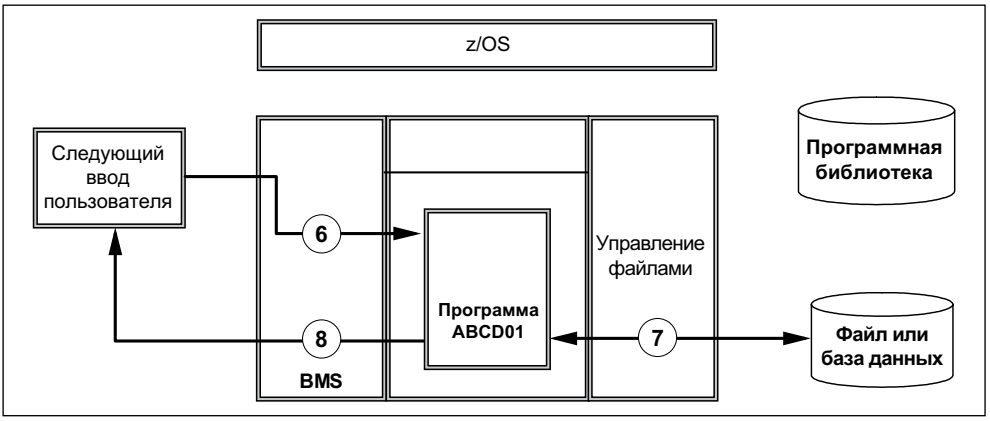


Рис. 11.9. Поток транзакций в CICS (часть 3)

6. BMS и модуль управления терминалами обрабатывают также следующий ввод пользователя, передавая его в программу `ABCD01` (программу, назначенную про-

граммой `ABDC00` для обработки следующего ответа с терминала), которая затем вызывает модуль управления файлами.

7. *Модуль управления файлами* считывает соответствующий файл, запрошенный пользователем терминала для обработки,.
8. И, наконец, `ABCD01` вызывает BMS и модуль управления терминалами для форматирования извлеченных данных и их вывода на терминал.

11.4.7 Службы CICS для приложений

CICS-приложения выполняются под управлением CICS, используя службы и интерфейсы CICS для доступа к программам и файлам.

Интерфейс программирования приложений

Интерфейс программирования приложений (application programming interface, API) используется для доступа к службам CICS из приложения. Составление CICS-программы выполняется во многом так же, как и составление любой другой программы. Большая часть логики обработки выражается элементами стандартных языков, тогда как для доступа к службам CICS используются CICS-команды.

Службы управления терминалами

Эти службы позволяют CICS-приложению осуществлять связь с терминальными устройствами. Посредством этих служб можно передавать информацию на экран терминала и принимать с терминала пользовательский ввод. Работать со *службами управления терминалами* напрямую довольно непросто. Модуль BMS (Basic Mapping Support) позволяет осуществлять связь с терминалом на высокоуровневом языке. Он автоматически выполняет форматирование данных, и программисту необязательно знать подробности потока данных.

Службы управления файлами и базами данных

Можно провести различие между следующими двумя службами управления данными CICS.

1. *Служба управления файлами* (*File control*) позволяет осуществлять доступ к наборам данных, управляемых методами доступа VSAM (Virtual Storage Access Method) или BDAM (Basic Direct Access Method). Служба управления файлами позволяет выполнять чтение, изменение, добавление и просмотр данных в наборах данных VSAM и BDAM, а также удалять данные из наборов данных VSAM.
2. *Служба управления базами данных* (*Database control*) позволяет осуществлять доступ к базам данных DL/I и DB2. Несмотря на то, что CICS имеет два интерфейса программирования для DL/I, рекомендуется использовать высокоуровневый интерфейс EXEC DL/I. CICS имеет один интерфейс для DB2: интерфейс EXEC SQL, содержащий мощные операторы управления наборами таблиц, что упрощает обработку приложений по отдельным записям (или по отдельным сегментам, в случае DL/I).

Прочие службы CICS

Служба управления задачами (*Task control*) может использоваться для управления выполнением задач. Можно приостановить задачу или запланировать использование ресурса задачей, обеспечив поддержку последовательного многократного использования. Кроме того, можно изменить приоритет, назначенный задаче.

Служба управления программами (*Program control*) управляет передачей управления между приложениями и системой CICS. Имя приложения, указанное в команде управления программой, должно быть определено в CICS как программа. Можно использовать команды управления программами для связывания одного приложения с другим и передачи управления от одного приложения к другому, без возврата к запрашивающей программе.

Служба управления временной памятью и динамическими данными (*Temporary Storage (TS) and Transient Data (TD) control*). Средство управления временной памятью CICS позволяет программисту приложений сохранять данные в очередях временной памяти, либо в основной памяти или вспомогательной памяти на устройстве хранения с прямым доступом, либо, в случае временной памяти, на устройстве сопряжения. Средство управления динамическими данными CICS содержит общее средство организации очередей для постановки в очередь (или сохранения) данных для последующей или внешней обработки.

Службы управления интервалами (*Interval control*) содержат функции, связанные со временем. Используя команды управления интервалами, можно среди прочего запустить задачу в заданное время или после определенного интервала, отложить обработку задачи и запросить уведомление по истечении заданного времени.

Служба управления памятью (*Storage control*) управляет запросами к основной памяти на предоставление промежуточных рабочих областей и прочей основной памяти, нужной для обработки транзакции. CICS делает рабочую память доступной в каждой программе автоматически, без запросов из приложения и обеспечивает другие средства промежуточного хранения как внутри задач, так и между задачами. Однако помимо рабочей памяти, автоматически создаваемой CICS, можно использовать другие CICS-команды для получения и освобождения основной памяти.

Службы управления дампами и трассировкой (Dump and trace control). Служба управления дампами создает дамп транзакции в случае аварийного завершения во время выполнения приложения. CICS-трассировка представляет собой вспомогательное средство отладки для программистов приложений, генерирующее трассировочные записи последовательности CICS-операций.

11.4.8 Управление программами

Транзакция (задача) в процессе выполнения своей работы может запускать несколько программ.

Определение программ содержит по одной записи для каждой программы, используемой любым приложением в системе CICS. Каждая запись, среди прочего, содержит сведения о языке, на котором написана программа. Определение транзакций содержит запись для каждого идентификатора транзакции в системе, и среди важной

информации по каждой транзакции хранится идентификатор и имя первой программы, запускаемой от имени транзакции.

Ниже описано взаимодействие этих двух наборов определений, для транзакции и для программы.

- Пользователь вводит идентификатор транзакции на терминале (или он определяется предыдущей транзакцией).
- CICS ищет этот идентификатор в списке установленных определений транзакций.
- Это сообщает CICS, какую программу следует вызвать первой.
- CICS ищет эту программу в списке установленных определений транзакций, определяет ее местонахождение и загружает ее (если она еще не загружена в основную память).
- CICS выполняет создание управляющих блоков, нужных для этого определенного сочетания транзакции и терминала, используя информацию из обоих наборов определений. Для программ на языке COBOL командного уровня это включает создание личной копии рабочей памяти для конкретного сеанса выполнения программы.
- CICS передает управление программе, которая начинает выполнение, используя управляющие блоки для данного терминала. Если нужно, эта программа в процессе выполнения транзакции может передать управление любой другой программе в списке установленных определений программ.

Существует две CICS-команды передачи управления из одной программы в другую. Одной из них является команда LINK, подобная оператору CALL в COBOL. Другой командой является команда XCTL (transfer control), не имеющая аналога в COBOL. Когда одна программа ссылается на другую (командой LINK), первая программа остается в основной памяти. Когда вторая программа (на которую была выполнена ссылка) завершает выполнение, первая программа продолжает выполнение после команды LINK. Программа, на которую выполняется ссылка, считается выполняющейся на один логический уровень ниже программы, выполняющей вызов по ссылке.

С другой стороны, когда одна программа передает управление другой (командой XCTL), первая программа считается завершенной, а вторая выполняется на том же уровне, что и первая. Когда вторая программа завершает выполнение, управление передается не первой программе, а той, которая последней выдала команду LINK.

Некоторые рассматривают саму систему CICS как наивысший программный уровень в этом процессе, а первую программу в транзакции как программу на один уровень ниже и т. д. Эта концепция изображена на рис. 11.10.

Команда LINK имеет следующий вид:

```
EXEC CICS LINK PROGRAM(pgmname)
COMMAREA(commarea) LENGTH(length) END-EXEC.
```

где `pgmname` – имя программы, на которую требуется выполнить ссылку. `Commarea` – имя области, содержащей данные, подлежащие передаче, и/или области, в которую следует возвратить результаты. Интерфейс COMMAREA представляет собой еще один способ вызова CICS-программ.

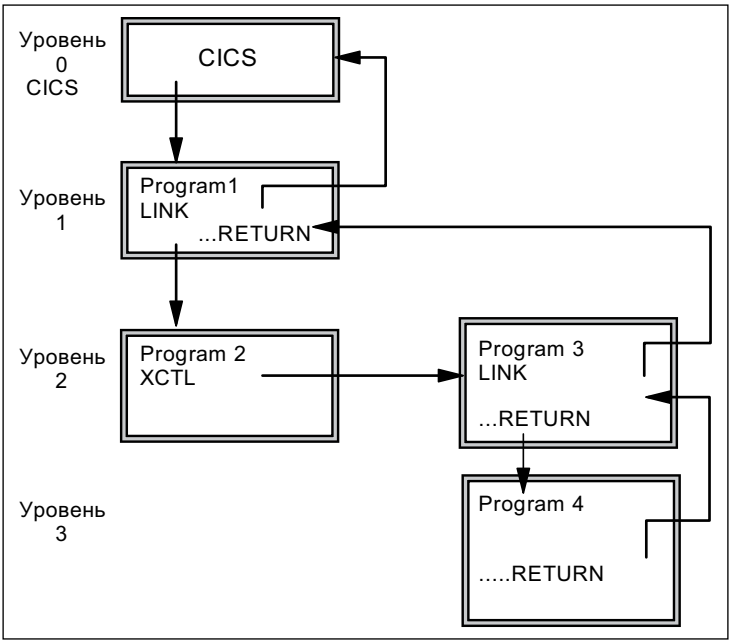


Рис. 11.10. Передача управления между программами (нормальные завершения)

Основной принцип проектирования CICS-приложения состоит в отделении логики представления от бизнес-логики; связь между программами осуществляется с использованием команды LINK, и передача данных между этими программами выполняется через интерфейс COMMAREA. Такая модульная структура обеспечивает не только разделение функций, но и гораздо более высокую гибкость обеспечения доступа к существующим приложениям через веб с использованием новых методов представления.

11.4.9 План программирования в CICS

Ниже перечислены основные этапы разработки CICS-приложения, использующего интерфейс командного программирования EXEC CICS.

1. Спроектируйте приложение, указав ресурсы и используемые службы CICS. См. главу, посвященную проектированию приложений в руководстве *CICS Application Programming Guide*.
2. Напишите программу на выбранном вами языке, используя команды EXEC CICS для запроса служб CICS. Список CICS-команд см. в руководстве *CICS Application Programming Reference*.

   Одним из требуемых компонентов оперативных транзакций является определение экрана, т. е. схема изображения на экране (например, веб-страницы); в CICS такая схема называется *картой* (*map*).
3. В зависимости от компилятора, может потребоваться только лишь скомпилировать программу и установить ее в CICS или же может потребоваться определить

опции транслятора программы, после чего выполнить трансляцию и компиляцию программы. Дополнительные сведения см. в руководстве *CICS Application Programming Guide*.

4. Определите свою программу и связанные транзакции в CICS, используя определения ресурсов PROGRAM и определения ресурсов TRANSACTION, как описано в руководстве *CICS Resource Definition Guide*.
5. Определите все используемые программой ресурсы CICS, в частности, файлы, очереди и терминалы.
6. Сделайте ресурсы доступными из CICS, используя команду CEDA INSTALL, описанную в руководстве *CICS Resource Definition Guide*.

11.4.10 Пример оперативной обработки

Возвращаясь к примеру туристического агентства, приведенному в главе 11 «Системы управления транзакциями в z/OS», можно привести следующие примеры CICS-транзакций:

- добавление, изменение и/или удаление информации о сотруднике;
- добавление, изменение и/или удаление информации о доступных автомобилях компанией по их прокату;
- получение информации о количестве доступных автомобилей компанией по их прокату;
- обновление цен на прокатные автомобили;
- добавление, изменение и/или удаление информации о регулярных рейсах авиакомпанией;
- получение информации о количестве проданных билетов по авиакомпаниям или по пункту назначения.

```
ABCD                              Average salary by department

Type a department number and press enter.

Department number: A02

Average salary($):   58211.58




F3: Exit
```

Рис. 11.11. Экран пользователя CICS-приложения

На рис. 11.11 показано, как пользователь может определить среднюю зарплату по отделу. Пользователь вводит код отдела, после чего транзакция определяет среднюю зарплату.

Обратите внимание на возможность добавления определений PF-клавиш на пользовательских экранах в своих CICS-приложениях.

11.5 Что такое IMS?

Созданная в 1969 году под названием Information Management System/360, система IMS является и менеджером транзакций, и менеджером баз данных для z/OS. IMS состоит из трех компонентов: менеджера транзакций (Transaction Manager, TM), менеджера баз данных (Database Manager, DB) и набора системных служб, реализующих общие функции для двух других компонентов (рис. 11.12).

IMS – продукт компании IBM, осуществляющий поддержку иерархических баз данных, передачу данных, обработку транзакций, а также откат и восстановление баз данных

По мере развития IMS, были добавлены новые интерфейсы, призванные обеспечить соответствие новым бизнес-требованиям. Теперь доступ к ресурсам IMS можно получить, используя множество интерфейсов к компонентам IMS.

В этой главе мы рассмотрим функции IMS как менеджера транзакций; функции, связанные с базами данных, более подробно рассматриваются в главе 12 «Системы управления базами данных в z/OS».

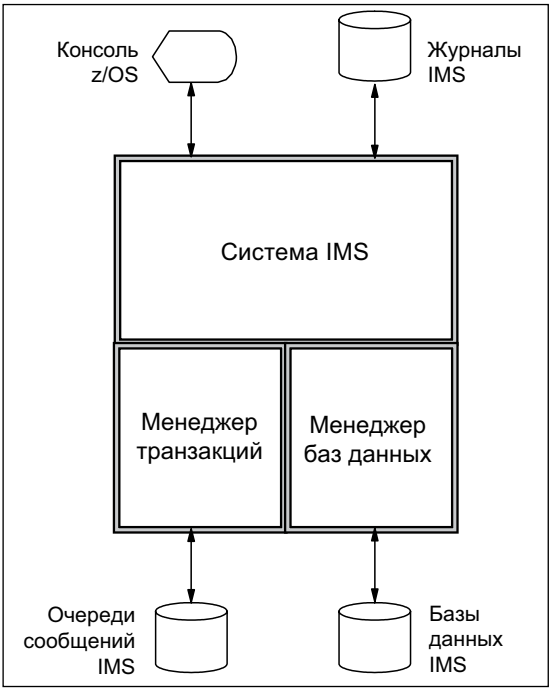


Рис. 11.12. Обзор IMS

Создание IMS-программы во многом выполняется так же, как и создание любой другой программы. Для создания IMS-приложений для z/OS можно использовать COBOL, OO COBOL, C, C++, Java, PL/I или ассемблер. Дополнительные сведения о программировании на языке Java см. в публикации *IMS Java Guide and Reference*.

Менеджер транзакций IMS

Менеджер транзакций IMS (IMS Transaction Manager) предоставляет пользователям сети доступ к приложениям, выполняющимся под управлением IMS. Пользователями могут быть люди, работающие на терминалах или рабочих станциях или же другие приложения, выполняющиеся в той же системе z/OS, на других системах z/OS или на платформах, отличных от z/OS.

Транзакция представляет собой набор входных данных, вызывающий выполнение определенного бизнес-приложения. Сообщение, предназначаемое для приложения, и любые возвращаемые результаты считаются одной транзакцией.

Менеджер баз данных IMS

Менеджер баз данных IMS представляет собой центральный пункт управления и доступа к данным, обрабатываемых IMS-приложениями. Он поддерживает базы данных, использующие иерархическую модель баз данных IMS, и обеспечивает доступ к этим базам данных из приложений, выполняющихся под управлением менеджера транзакций IMS, монитора транзакций CICS (теперь имеющего название Transaction Server for z/OS) и пакетных заданий z/OS.

Менеджер баз данных содержит средства защиты (резервного копирования и восстановления) и обслуживания баз данных. Он позволяет нескольким задачам (пакетным и/или оперативным) осуществлять доступ и изменять данные, сохраняя целостность этих данных. Кроме того, он содержит средства настройки баз данных посредством их реорганизации и реструктуризации. Внутренняя организация баз данных IMS выполнена с использованием множества методов доступа организации баз данных IMS. Эти базы данных хранятся в дисковом хранилище, и доступ к ним осуществляется с использованием обычных методов доступа операционной системы.

Более подробно менеджер баз данных IMS рассматривается в главе 12 «Системы управления базами данных в z/OS».

Системные службы IMS

Существует множество функций, общих для менеджера баз данных и менеджера транзакций:

- перезапуск и восстановление подсистем IMS после отказов;
- безопасность: управление доступом к ресурсам IMS;
- управление приложениями: диспетчеризация работы, загрузка приложений, управление блокировкой;
- предоставление диагностической информации и информации о производительности;
- обеспечение средств работы с подсистемами IMS;
- обеспечение интерфейса к другим подсистемам z/OS, с которыми взаимодействуют IMS-приложения.

11.5.1 IMS в системе z/OS

IMS работает на zSeries и более ранних вариантах архитектуры S/390 или совместимых с ней мэйнфреймов, а также в z/OS и более ранних вариантах операционной системы. Подсистема IMS выполняется в нескольких адресных пространствах в системе z/OS. Используется одно управляющее адресное пространство и несколько зависимых адресных пространств, обеспечивающих IMS-службы и выполняющих IMS-приложения.

По историческим причинам некоторые документы, описывающие IMS, используют для описания адресного пространства z/OS термин *регион* (*region*), например, управляющий регион IMS (IMS Control Region). В этой книге термин «регион» используется в тех случаях, когда он имеет общее употребление. Термин «регион» можно воспринимать как синоним термина «адресное пространство z/OS».

Для того чтобы наилучшим образом использовать уникальные преимущества z/OS, IMS выполняет следующие действия:

- Выполняется в нескольких адресных пространствах – подсистемы IMS (за исключением пакетных приложений и утилит IMS/DB) обычно состоят из адресного пространства управляющего региона, зависимых адресных пространств, обеспечивающих работу системных служб, и зависимых адресных пространств для приложений.
- Выполняет по несколько задач в каждом адресном пространстве IMS, особенно в управляющих регионах, создает несколько подзадач z/OS для различных выполняемых функций. Это позволяет осуществлять диспетчеризацию других подзадач IMS в z/OS, пока одна подзадача IMS ожидает системных служб.
- Использует межпространственные службы z/OS для связи между различными адресными пространствами, составляющими подсистему IMS. Она также использует общую системную область z/OS для хранения управляющих блоков IMS, к которым часто осуществляется доступ из адресных пространств IMS, что сокращает нагрузку, связанную с использованием нескольких адресных пространств.
- Использует интерфейс подсистем z/OS – IMS динамически регистрируется как подсистема z/OS. Это средство используется для обнаружения отказа зависимых адресных пространств и предотвращения удаления зависимых адресных пространств (и для взаимодействия с другими подсистемами, подобными DB2 и WebSphere MQ).
- Может использовать сисплекс z/OS – несколько подсистем IMS могут выполняться в системах z/OS, составляющих сисплекс и осуществляющих доступ к тем же базам данных IMS.

11.5.2 Сообщения менеджера транзакций IMS

Сетевые входные и выходные данные менеджера транзакций IMS имеют вид входных и выходных сообщений IMS и физических терминалов или приложений в сети. Эти сообщения обрабатываются асинхронно (т. е. при получении сообщения IMS не всегда отправляет ответ немедленно или даже когда бы то ни было, и IMS также может отправлять незапрашиваемые сообщения).

Сообщения могут быть четырех типов.

- Транзакции; в этих сообщениях данные передаются в IMS-приложения для обработки.
- Сообщения, предназначенные для других логических пунктов назначения, таких как сетевые терминалы.
- Команды для обработки в IMS.
- Сообщения для обработки в IMS APPC. Так как IMS использует асинхронный протокол для сообщений, а APPC использует синхронные протоколы (т. е. он всегда ожидает ответ после отправления сообщения), интерфейс IMS TM для APPC должен выполнять специальную обработку.
  Если IMS не может обработать входное сообщение сразу или немедленно отправить выходное сообщение, то сообщение сохраняется в очереди сообщений, внешней по отношению к системе IMS. IMS обычно не удаляет сообщения из очереди сообщений до тех пор, пока не получит подтверждение того, что приложение обработало сообщение, или что оно достигло пункта назначения.

11.6 Заключение

В этой главе мы увидели, что транзакционные приложения продолжают изменяться в соответствии с потребностями организации, ее клиентов и поставщиков. Иногда изменения реализуются посредством внедрения новых технологий, тогда как надежные приложения остаются неизменными. Взаимодействие с компьютером осуществляется в оперативном режиме с помощью менеджера транзакций. Существует множество менеджеров транзакций и менеджеров баз данных, однако они имеют одинаковый принцип работы.

CICS представляет собой подсистему обработки транзакций. Это означает, что она выполняет приложения от имени пользователя в оперативном режиме по запросу, в то же время, когда многие другие пользователи могут передавать запросы на запуск тех же приложений, используя те же файлы и программы. CICS управляет совместным использованием ресурсов, целостностью данных и назначением приоритетов выполнения с быстрым реагированием. CICS-приложения традиционно запускаются посредством передачи запроса *транзакции*. Выполнение транзакции состоит в выполнении одного или нескольких *приложений*, реализующих требуемую функцию.

Создание CICS-программы во многом выполняется так же, как и создание любой другой программы. Для создания CICS-приложений можно использовать COBOL, C, C++, Java, PL/I или ассемблер. Большая часть логики обработки выражается операторами стандартных языков, но также можно использовать *CICS-команды*. CICS-команды группируются в соответствии с их функцией, взаимодействием с терминалом, доступом к файлам или связыванием программ. Большая часть ресурсов CICS может определяться и изменяться оперативно посредством CICS-транзакций. Другие транзакции позволяют осуществлять мониторинг системы CICS. Продолжающийся рост Интернета привел к тому, что многие корпорации ищут наилучшие способы сделать свои существующие системы доступными для пользователей в Интернете. Был представлен краткий обзор различных технологий, доступных для обеспечения доступа к существующим приложениям через веб.

Система IMS (Information Management System) состоит из трех компонентов: менеджера транзакций (Transaction Manager, TM), менеджера баз данных (Database Manager, DB) и набора системных служб, реализующих общие функции для двух других компонентов. Создание IMS-программы во многом выполняется так же, как и создание любой другой программы. Для создания IMS-приложений можно использовать COBOL, OO COBOL, C, C++, Java, PL/I или ассемблер.

Основные термины в этой главе

IMS TM	Диалоговая транзакция	Псевдодиалоговая транзакция
IRLM	CICS TS	Information Management System (IMS)
CICS-команда	Регион	Basic Mapping Support (BMS)
Многопоточность	Транзакция	Единица работы

11.7 Контрольные вопросы

Чтобы проверить понимание материала этой главы, ответьте на следующие вопросы:

1. Назовите типичные оперативные транзакции, выполняемые вами часто.
2. Почему многозадачность и многопоточность важны для оперативной обработки транзакций?
3. Каковы общие свойства оперативной транзакционной системы?
4. Опишите двухфазовую фиксацию.
5. Опишите основные фазы плана программирования CICS.
6. В чем различие между терминами «бизнес-транзакция» и «CICS-транзакция»?
7. Как выполняется определение ресурсов в CICS?
8. Какие основные компоненты содержит IMS, и каково их назначение?
9. Каковы четыре типа IMS-сообщений?

11.8 Упражнение: Создание CICS-программы

Ниже представлено пробное упражнение.

Во время работы над упражнением может быть полезным использовать руководство *CICS Application Programming Guide*.

Анализ и изменение учебной программы

- Рассмотрите возможные варианты использования интерфейса COMMAREA. Рассмотрите передачу данных между вызываемыми программами с использованием команд LINK или XCTL. Можно разработать общую программу обработки ошибок; все обращения к ней можно выполнять вместе с передачей требуемых данных об ошибках через COMMAREA. Кроме того, опция COMMAREA команды возврата предназначена для передачи данных между последовательными транзакциями в псевдодиалоговой последовательности.
  Состояние ресурса может быть передано первой транзакцией через интерфейс COMMAREA для сравнения с текущим состоянием второй транзакцией. Прежде чем разрешать обновление, может быть необходимо знать, изменилось ли со-

стояние с момента последнего взаимодействия. В веб-приложениях бизнес-логика в CICS-приложении может быть вызвана с использованием интерфейса COMMAREA.

- Несколько простых обновлений транзакции учебной программы можно выполнить достаточно просто:
  Добавьте одно дополнительное поле выходных данных на экране. Например, можно использовать максимальное значение премии сотрудника.
  Новое поле должно быть определено в исходной карте. Возможно нужно будет изменить некоторые элементы. Составьте карту и сгенерируйте новый копируемый файл. Измените программу таким образом, чтобы она имела еще один столбец в SQL-выражении, и после извлечения переместите его содержимое в соответствующее новое выходное поле на карте. Выполните подготовительное задание для пользовательской программы. Сеанс CICS требует новых копий программы и карты.
  - Создайте транзакцию, которая может быть подобной главному меню; одна из опций запускает текущую программу.
    В карте этой транзакции обязательными являются только два переменных поля: поле опций и строка сообщений. Изначально включается только одна опция, соответствующая текущей транзакции ABCD. Для добавления новой карты может использоваться тот же набор карт . Транзакция ABCD должна быть изменена таким образом, чтобы выполнить команду RETURN TRANSID для новой транзакции. В систему CICS необходимо добавлять только следующие ресурсы: новая транзакция и программы (пользовательская программа и карта).
  - Рассмотрите оператор CICS HANDLE CONDITION и подумайте, где его можно использовать.
    Попытайтесь добавить средства контроля ошибок в команду RECEIVE CICS. Ситуация MAPFAIL возникает, когда после команды RECEIVE с терминала не передаются допустимые данные.

Бизнес-транзакция

Проанализируйте типичную бизнес-транзакцию. Рассмотрите различные CICS-программы и транзакции, которые могут быть нужны для ее выполнения. Нарисуйте диаграмму, чтобы изобразить схему процесса.

Можно использовать пример, представленный в книге *CICS Application Programming Primer*. Универмаг с покупателями, совершающими покупки в кредит, хранит главный файл счетов своих клиентов. Приложение выполняет следующие действия:

- вывод записей счетов клиентов;
- добавление новых записей счетов;
- изменение или удаление существующих записей счетов;
- печать одной копии записи счета клиента;
- доступ к записям по имени.

12

Системы управления базами данных в z/OS

Цель. Вам потребуется хорошее понимание основных типов системного программного обеспечения, используемого для обработки оперативной рабочей нагрузки мэйнфрейма. В этой главе мы сконцентрируемся на двух наиболее часто используемых системах управления базами данных (СУБД) для z/OS: DB2 и IMS DB. После завершения работы над этой главой вы сможете:

- объяснить, как используются базы данных на предприятии, работающем в оперативном режиме;
- описать две модели сетевой связи для больших систем;
- описать роль DB2 в обработке оперативных транзакций;
- перечислить основные структуры данных DB2;
- составить простые SQL-запросы для выполнения в z/OS;
- представить обзор программирования приложений с использованием DB2;
- описать компоненты IMS;
- описать структуру подсистемы IMS DB.

12.1 Системы управления базами данных для мэйнфрейма

В этом разделе представлен обзор основных понятий сферы баз данных, их использования и их преимуществ. Существует множество различных баз данных, однако здесь мы ограничимся рассмотрением двух типов баз данных, чаще всего используемых на мэйнфреймах: иерархических и реляционных баз данных.

12.2 Что такое база данных?

База данных обеспечивает хранение и управление бизнес-данными. Она независима от приложений (но не от их требований обработки). При правильном проектировании и реализации, база данных обеспечивает единое согласованное представление бизнес-данных, что позволяет обеспечить централизованный контроль и управление.

Один из способов описания логического представления этого набора данных состоит в использовании модели взаимоотношений между сущностями. База данных описывает подробности (атрибуты) определенных объектов (сущностей) и взаимоотношения между различными типами сущностей. Например, модуль контроля запасов в приложении описывает изделия, заказы на покупку, клиентов и заказы клиентов (сущности). Каждая сущность имеет атрибуты; в частности, изделие имеет такие атрибуты как номер изделия, название, цена за единицу, количество и т. д.

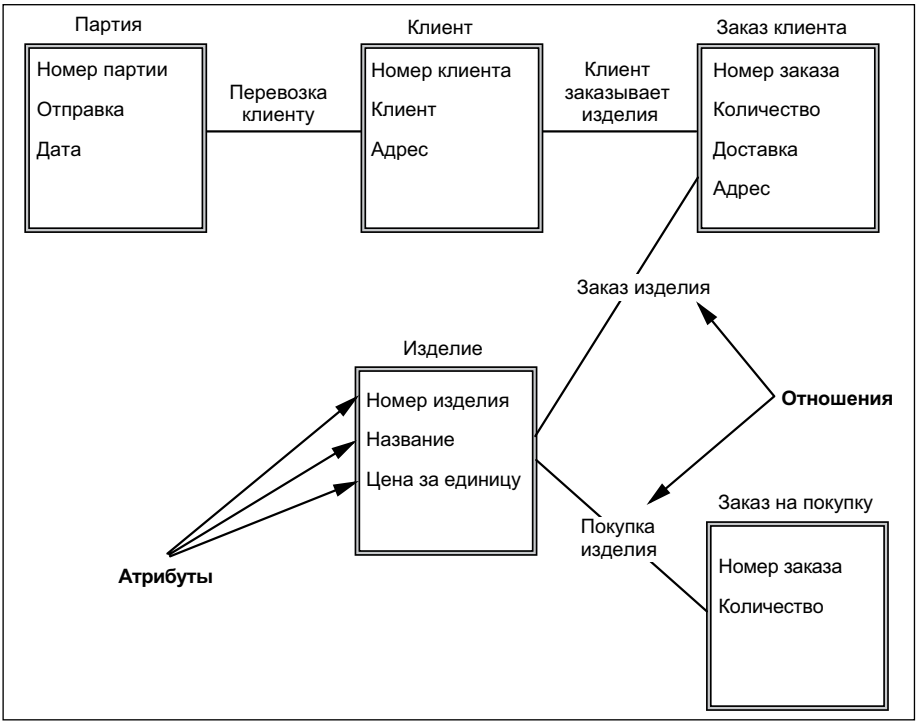


Рис. 12.1. Сущности, атрибуты и отношения

Кроме того, существуют отношения между сущностями, например, клиент связан с размещенными заказами, которые связаны с заказанным изделием и т. д. Модель взаимоотношений между сущностями представлена на рис. 12.1.

Система управления базами данных (СУБД), например компонент IMS Database Manager (IMS/DB) или система DB2, обеспечивает метод хранения и использования бизнес-данных в базе данных.

12.3 Для чего используются базы данных?

Когда компьютеры только появились, данные хранились в отдельных файлах, которые использовались только этим приложением или даже небольшой частью отдельного приложения. Однако надлежащим образом спроектированная и реализованная СУБД имеет множество преимуществ по сравнению с использованием простой файловой системы на основе PDS.

СУБД – система управления базами данных, реализующая метод хранения и использования данных в базе данных

- Она сокращает объем работы, связанной с программированием приложений.
- Она более эффективно управляет созданием, изменением и доступом к данным, чем система, отличная от СУБД. Как известно, для того чтобы добавить в файл новые элементы данных, потребуется переписать все приложения, использующие этот файл, даже если они не используют новый элемент данных. При использовании СУБД это не нужно. Хотя многие программисты используют разные хитрости, позволяющие уменьшить объем работы по программированию приложений, определенный объем работы все еще остается.
- Она обеспечивает более высокий уровень защиты данных и конфиденциальности, чем простая файловая система. В частности, при доступе к логической записи в простом файле приложение может видеть *все* элементы данных, включая любые конфиденциальные и привилегированные данные. Для того чтобы решить эту проблему, многие клиенты прибегают к размещению конфиденциальных данных в отдельном файле и связыванию двух файлов при необходимости. Это может вызвать проблемы с согласованностью данных.

Сегмент – любой раздел, зарезервированная область, частичный компонент или фрагмент более крупной структуры

При использовании СУБД конфиденциальные данные можно изолировать в отдельном сегменте (в IMS/DB) или представлении (в DB2), предотвращающем их просмотр несанкционированными приложениями. Но эти элементы данных являются составной частью логической записи.

Тем не менее, одни и те же сведения могут храниться в нескольких различных местах; например, сведения о клиенте могут находиться и в приложении формирования заказов, и в приложении выставления счетов. Это вызывает ряд проблем.

- Так как информация хранится и обрабатывается независимо, сведения, которые должны быть одинаковыми (например, имя и адрес клиента), могут быть несогласованными в различных приложениях.

- Когда нужно будет изменить общие данные, потребуется выполнить их изменение в нескольких местах, что вызовет большую нагрузку. Если отсутствуют какие-нибудь копии данных, это вызывает проблемы, описанные в предыдущем пункте.
- Отсутствует центральный пункт управления данными, который обеспечивал бы их защиту от потери и несанкционированного доступа.
- Дублирование данных означает напрасное расходование места в хранилище.

Использование системы управления базами данных, такой как IMS/DB или DB2, для реализации базы данных обеспечивает дополнительные преимущества. В частности, СУБД:

- Позволяет нескольким задачам осуществлять одновременный доступ и изменение данных, сохраняя целостность базы данных. Это особенно важно в случаях, когда большое число пользователей осуществляет доступ к данным через оперативное приложение.
- Содержит средства изменения приложением нескольких записей базы данных, и обеспечивает согласованность данных приложения в различных записях, даже при отказе приложения.
- Может поместить конфиденциальные данные в отдельный сегмент (в IMS) или таблицу (в DB2). В отличие от СУБД, при использовании простого файла PDS или VSAM приложение осуществляет доступ к каждому элементу данных в логической записи. Некоторые из этих элементов могут содержать данные, доступ к которым должен быть ограниченным.
- Содержит утилиты, управляющие и реализующие резервное копирование и восстановление данных, не допуская потери критически важных бизнес-данных.
- Содержит утилиты мониторинга и настройки доступа к данным.
- Может изменять структуру логической записи (путем добавления или перемещения полей данных). Такие изменения обычно требуют переассемблирования или перекомпиляции каждого приложения, осуществляющего доступ к файлу VSAM или PDS, даже если оно не использует дополнительные или измененные поля. Правильно спроектированная база данных освобождает программиста приложений от таких изменений.

Однако следует помнить о том, что использование базы данных и системы управления базами данных само по себе не обеспечивает описанные преимущества. Требуется также надлежащее проектирование и администрирование баз данных и разработка приложений.

12.4 Кто такой администратор базы данных?

Администраторы баз данных (database administrators, DBA) главным образом отвечают за определенные базы данных в подсистеме. В некоторых компаниях администраторам баз данных назначаются специальные групповые полномочия, SYSADM, что позволяет им делать почти все, что угодно, в подсистеме DB2, и дает им права доступа ко всем базам данных в подсистеме. В других организациях полномочия администратора базы данных ограничиваются отдельными базами данных.

Администратор базы данных создает иерархию объектов данных от базы данных, табличных пространств и таблиц до любых требуемых индексов и представлений. Он также настраивает определения ссылочной целостности данных и необходимые ограничения.

Администратор базы данных в сущности реализует физическую структуру базы данных. Это включает необходимость определения размеров пространства, а также размеров физических наборов данных для табличных пространств и индексных пространств, а также назначения групп хранения (называемых также *storgroups*).

Существует множество инструментов, которые могут помочь администратору базы данных в выполнении этих задач. DB2, например, содержит Administration Tool и DB2 Estimator. Если размер объектов увеличивается, администратор базы данных может переделать некоторые объекты, чтобы внести изменения.

Администратор базы данных может отвечать за назначение полномочий доступа к объектам базы данных, хотя иногда этим занимается специальная группа безопасности.

Системам управления базами данных свойственна централизация данных и управление доступом к этим данным. Одним из преимуществ централизации является доступность согласованных данных более чем для одного приложения. В результате, это обеспечивает более строгий контроль над данными и их использованием.

Ответственность за точную реализацию управления возлагается на администратора базы данных. В действительности, для того чтобы получить полные преимущества использования централизованной базы данных, необходимо иметь центральный пункт управления ею. Так как действительная реализация функций администратора базы данных зависит от организации, мы ограничимся обсуждением ролей и обязанностей администратора базы данных. Группе, выполняющей роль администратора базы данных, нужен опыт как в прикладном, так и в системном программировании.

В типичной инсталляции администратор базы данных отвечает за следующее:

- разработка стандартов для баз данных, их администрирование и использование;
- разработка, пересмотр и утверждение структуры новых баз данных;
- определение правил доступа к данным и мониторинга их безопасности;
- обеспечение целостности и доступности базы данных, а также мониторинг необходимых действий по реорганизации, резервного копирования и восстановления;
- проверка взаимодействия новых программ с существующими рабочими базами данных на основе результатов тестирования с применением тестовых данных.

В целом администратор базы данных отвечает за поддержку актуальной информации о данных в базе данных. Изначально это можно делать вручную. Однако можно ожидать увеличения области действия и сложности в такой степени, которая оправдывает или требует использования программы словаря данных.

Администратор базы данных не отвечает за содержимое баз данных. За это отвечает пользователь. Со своей стороны, администратор базы данных применяет процедуры точного, полного и своевременного обновления баз данных.

12.5 Как выполняется проектирование баз данных?

Процесс проектирования баз данных вкратце можно описать как структурирование элементов данных для различных приложений таким образом, чтобы:

- каждый элемент данных был доступен для различных приложений на данный момент и в обозримом будущем;
- обеспечивалось эффективное хранение элементов данных;
- применялся контролируемый доступ для элементов данных с особыми требованиями безопасности.

За многие годы было разработано множество различных моделей баз данных (например, иерархическая, реляционная или объектная), поэтому не существует согласованного словаря, который описывал бы используемые понятия.

12.5.1 Сущности

База данных содержит информацию о сущностях. *Сущность* (*entity*) имеет следующие свойства:

- она может быть уникально определена;
- о ней можно собрать основную информацию, в настоящее время или в будущем.

В действительности это определение ограничено контекстом обсуждаемых приложений и предприятий. Примерами сущностей являются изделия, проекты, заказы, клиенты, товары и т. д. Важно понимать, что определение сущностей является основным этапом процесса проектирования базы данных. Информация о сущностях, содержащаяся в базах данных, описывается атрибутами данных.

12.5.2 Атрибуты данных

Атрибут данных (*data attribute*) представляет собой единицу информации, содержащую определенный факт о сущности. Например, допустим, что в качестве сущности рассматривается изделие. Известно три факта об этом изделии: название=шайба, цвет=зеленый, вес=143. Таким образом, имеется три атрибута данных.

Атрибут данных имеет имя и значение. Имя атрибута данных сообщает тип регистрируемого факта; значение представляет собой сам факт. В данном примере «имя», «цвет» и «вес» представляют собой имена атрибутов данных, тогда как «шайба», «зеленый» и «143» представляют значения. Для того чтобы значение было осмысленным, оно должно быть связано с именем.

Экземпляр (*occurrence*) представляет собой значение атрибута данных определенной сущности. Атрибут всегда зависит от сущности. Он не имеет смысла сам по себе. В зависимости от ее использования сущность может быть описана одним или несколькими атрибутами данных. В идеале сущность должна быть уникальным образом определена одним атрибутом данных, например, порядковым номером сущности. Такой атрибут данных называется *ключом* (*key*) сущности. Ключ представляет собой идентификатор определенного экземпляра сущности и является специальным атрибутом сущности. Ключи не всегда являются уникальными. Сущности с одинаковыми значениями ключа называются *синонимами* (*synonym*).

Например, полное имя человека не является уникальным идентификатором. В таких случаях приходится использовать другие атрибуты, такие как полный адрес, день рождения или произвольный порядковый номер. Более распространенный метод состоит в определении нового атрибута, выступающего в качестве уникального ключа, например, номера сотрудника.

12.5.3 Отношения между сущностями

Между идентифицированными сущностями существуют связи, называемые *отношениями* (*relationship*). Например, заказ может состоять из нескольких изделий. Опять же, эти отношения имеют смысл только в контексте приложения и предприятия. Существуют отношения «один-к-одному» (т. е. один экземпляр одной сущности соответствует одному экземпляру другой сущности), «один-ко-многим» (один экземпляр одной сущности соответствует многим экземплярам другой сущности) или «много-ко-многим» (многие экземпляры одной сущности имеют связь со многими экземплярами другой сущности).

Отношения также могут быть *рекурсивными* (*recursive*), когда экземпляр сущности может быть связан с другими экземплярами той же сущности. В частности, некоторое изделие, например крепеж, может состоять из нескольких других изделий: болта, гайки и шайбы.

12.5.4 Функции приложений

Сами по себе данные не являются конечной целью системы управления базами данных. Важна обработка, выполняемая над этими данными приложением. Наилучший способ представить эту обработку состоит в том, чтобы рассмотреть наименьший блок приложения, представляющий пользователя, взаимодействующего с базой данных, например, один заказ, состояние запасов одного изделия. В последующих разделах это называется *функцией приложения* (*application function*).

Функции обрабатываются приложениями. В пакетной системе множество функций объединяется в одной программе (например, все заказы за день), затем обрабатывается относительно базы данных за одно выполнение требуемого приложения . В оперативной системе можно сгруппировать только одну или две функции в одной программе, чтобы обеспечить одну итерацию с пользователем.

Несмотря на то, что функции всегда различимы, даже при пакетной обработке, некоторые люди предпочитают говорить о программах, а не о функциях. Однако четкое понимание функций является обязательным для надлежащего проектирования, особенно в среде баз данных. После определения функциональных требований приложения можно решить, как лучше реализовать их в программах, используя CICS или IMS. Функция, в некотором роде, представляет индивидуальное использование приложения определенным пользователем. По существу она является центральным элементом приложения баз данных.

12.5.5 Пути доступа

Входные данные каждой функции содержат определенную идентификацию используемых сущностей (например, номер изделия при доступе к базе данных изделий). Такая идентификация называется путем доступа данной функции. В целом функции

требуют произвольного доступа, хотя в целях повышения производительности иногда используется последовательный доступ. Это особенно верно для функций, проходящих пакетную обработку, для многочисленных функций относительно размера базы данных, или если нужна информация из большого количества записей базы данных. Для эффективного произвольного доступа каждый путь доступа должен использовать ключ сущности.

12.6 Что такое система управления базами данных?

Система управления базами данных (СУБД) в сущности представляет собой компьютерную систему хранения данных. Пользователи системы имеют доступ к средствам выполнения некоторых операций в такой системе, предполагающих либо управление данными в базе данных, либо управление структурой базы данных. Системы управления базами данных классифицируются в соответствии со структурами или типами данных.

Существует несколько типов баз данных, которые можно использовать на мэйнфрейме для использования z/OS: с инвертированной списковой структурой, иерархические, сетевые или реляционные.

В организациях, использующих мэйнфреймы, часто применяют иерархическую модель, когда *структура* данных (не значения), требуемая приложению, является сравнительно статичной. Например, структура базы данных ведомости материалов всегда содержит номер компонента сборки верхнего уровня и несколько уровней компонентов и подкомпонентов. Структура обычно содержит прогнозные данные, стоимость и цену компонентов и т. д. Структура данных в ведомостях материалов редко изменяется, и новые элементы данных (не значения) редко определяются. Приложение обычно начинается с номера компонента сборки верхнего уровня и переходит к составляющим компонентам нижних уровней.

Корень – верхний уровень иерархии

Обе системы управления базами данных имеют преимущества, перечисленные в разделе 12.3 «Для чего используются базы данных?». Реляционные СУБД имеют дополнительное важное преимущество над иерархической базой данных, состоящее в том, что она не является навигационной. Под *навигационной* подразумевается, что в иерархической базе данных программист приложений должен знать структуру базы данных. Программа должна содержать специальную логику для навигации от корневого сегмента к требуемым дочерним сегментам, содержащим требуемые атрибуты или элементы. Тем не менее, программа должна осуществлять доступ к промежуточным сегментам, даже если они не нужны.

В остальной части раздела рассматривается реляционная структура базы данных.

12.6.1 Какие структуры существуют в реляционной базе данных?

Реляционные базы данных содержат следующие структуры:

- База данных.
  База данных представляет собой логический набор данных. Она содержит набор связанных табличных пространств и индексных пространств. Обычно ба-

за данных содержит все данные, связанные с одним приложением или с набором связанных приложений. Например, может использоваться база данных ведомостей или база данных инвентаризации.

- Таблица.
  Таблица представляет собой логическую структуру, состоящую из строк и столбцов. Строки не имеют фиксированного порядка, поэтому для получения данных может потребоваться их сортировка. Порядок столбцов определяется при создании таблицы администратором базы данных. На пересечениях столбцов и строк находятся элементы данных, называемые значениями или, если точнее, элементарными значениями. Имя таблицы состоит из высокоуровневого квалификатора, соответствующего идентификатору пользователя-владельца, и имени самой таблицы, например TEST.DEPT или PROD.DEPT. Существует три типа таблиц:
  - базовая таблица, создаваемая и содержащая постоянные данные;
  - временная таблица, хранящая промежуточные результаты запросов;
  - таблица результатов, возвращаемая при запросах таблиц.

SQL – Structured Query Language (язык структурированных запросов) – язык, используемый для опрашивания и обработки данных в реляционной базе данных

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00
B01	PLANNING	000020	A00
C01	INFORMATION CENTER	000030	A00
D01	DEVELOPMENT CENTER		A00
E01	SUPPORT SERVICES	000050	A00
D11	MANUFACTURING SYSTEMS	000060	D01
D21	ADMINISTRATION SYSTEMS	000070	D01
E11	OPERATIONS	000090	E01
E21	SOFTWARE SUPPORT	000100	E01

Рис. 12.2. Пример таблицы DB2 (таблица отделов)

Эта таблица содержит:

- столбцы – упорядоченный набор столбцов DEPTNO, DEPTNAME, MGRNO и ADMRDEPT. Все данные в определенном столбце должны иметь одинаковый тип данных;
- строки – каждая строка содержит данные по одному отделу;
- значения – находятся на пересечении столбцов и строк. Например, PLANNING является значением на пересечении столбца DEPTNAME и строки, соответствующей отделу B01.

- Индексы.
  Индекс представляет собой упорядоченный набор указателей на строки таблицы. В отличие от строк таблицы, не имеющих определенного порядка, индекс в DB2 всегда должен поддерживаться в порядке. Индекс имеет следующее назначение:

- обеспечение быстродействия, что позволяет быстрее извлекать данные;
- обеспечение уникальности.

Создание индекса по имени сотрудника позволяет быстрее извлекать данные по этому сотруднику чем посредством сканирования всей таблицы. Кроме того, при создании уникального индекса по номеру сотрудника, DB2 принудительно обеспечивает уникальность каждого значения. Уникальный индекс является единственным способом применения уникальности в DB2.
При создании индекса автоматически создается *индексное пространство* (*index space*) – набор данных, содержащий индекс.

- Ключи.
  Ключ представляет собой один или несколько столбцов, идентифицированных как ключ при создании таблицы или индекса или при определении ссылочной целостности данных.
  - Первичный ключ.

Таблица может иметь только один первичный ключ, так как он определяет сущность. Существует два требования, предъявляемых к первичному ключу:

1. Он должен иметь значение, т. е. он не может содержать пустое значение (null).
2. Он должен быть уникальным, т. е. для него должен быть определен уникальный индекс.
   - Уникальный ключ.
     Мы уже знаем, что первичный ключ должен быть уникальным, однако в таблице может быть больше одного уникального ключа. В нашем примере таблицы EMP [см. раздел «Таблица сотрудников (EMP)»] номер сотрудника определяется как первичный ключ, который, таким образом, является уникальным. Если бы наша таблица содержала номер социального страхования, это значение также было бы уникальным. Для того чтобы обеспечить уникальность, можно создать уникальный индекс по столбцу номера социального страхования.
   - Внешний ключ.
     Внешним ключом является ключ, определенный в ограничении ссылочной целостности данных, чтобы сделать его существование зависимым от первичного или уникального ключа (родительского ключа) в другой таблице .
     В приведенном примере номер рабочего отдела сотрудника связан с первичным ключом, определенным в номере отдела в таблице DEPT. Это ограничение является частью определения таблицы.

12.7 Что такое DB2?

Основные понятия реляционной системы управления базами данных (РСУБД) обсуждаются в главе 11 «Системы управления транзакциями в z/OS». Большинство примеров таблиц, рассматриваемых в этой главе, можно найти в Приложении B «Примеры таблиц DB2». Эти таблицы, в частности, EMP и DEPT, являются частью Sample Database, распространяемой вместе с DB2 на всех платформах. Скриншоты взяты из версии 8. Поэтому владельцем таблиц является DSN8810.

Элементы, управляемые DB2, можно разделить на две категории: структуры данных, используемые для организации пользовательских данных, и системные структуры, управляемые DB2. Структуры данных можно разделить на *базовые структуры* (*basic structures*) и *структуры схемы* (*schema structures*). Структуры схемы являются сравнительно новыми объектами, реализованными на мэйнфреймах для совместимости в семействе продуктов DB2. *Схема* (*schema*) представляет собой логическую группу этих новых объектов.

12.7.1 Структуры данных в DB2

Ранее в этой главе мы обсудили большую часть базовых структур, общих для систем управления базами данных. Теперь мы рассмотрим некоторые структуры, характерные для DB2.

Представления

Представление (*view*) является альтернативным способом просмотра данных из одной или нескольких таблиц. Оно подобно трафарету, накладываемому на изображение на слайде, позволяющему видеть только лишь некоторые аспекты базового изображения. Например, можно создать представление для таблицы отделов, которое бы позволяло пользователям осуществлять доступ к одному определенному отделу для обновления информации о заработной плате. При этом пользователи не должны видеть информацию о зарплате в других отделах. Для этого создается представление таблицы, которое позволяет пользователям видеть только один отдел; при этом представление используется так же, как таблица. Таким образом, представление используется в целях безопасности. В большинстве компаний пользователям не разрешается осуществлять прямой доступ к таблицам; для этого используются представления. Пользователи получают доступ через представление. Представление можно также использовать для упрощения сложного запроса для менее опытных пользователей.

Представление – способ просмотра данных в таблице, позволяющий контролировать видимость информации разными пользователями

Табличное пространство

Таблица представляет собой только лишь логическую конструкцию. Она хранится в физическом наборе данных, называемом *табличным пространством* (*table space*). Табличные пространства представляют собой структуры хранения, способные содержать одну или несколько таблиц. Имя табличного пространства состоит из имени базы данных, за которым следует имя табличного пространства, например, PAYROLL.ACCNT_RECV. Существует три типа табличных пространств: простые, сегментированные и многораздельные. Дополнительные сведения см. в руководстве *DB2 UDB for z/OS: SQL Reference*.

В DB2 используются наборы данных VSAM. Другими словами, каждый сегмент представляет собой набор данных VSAM.

Индексное пространство

Индексное пространство (*index space*) представляет собой еще одну структуру хранения, содержащую один индекс. В действительности при создании индекса DB2 автоматически определяет индексное пространство.

Группы хранения

Группа хранения (*storage group*) представляет собой набор томов на дисках (DASD), содержащий наборы данных, в которых фактически хранятся таблицы и индексы.

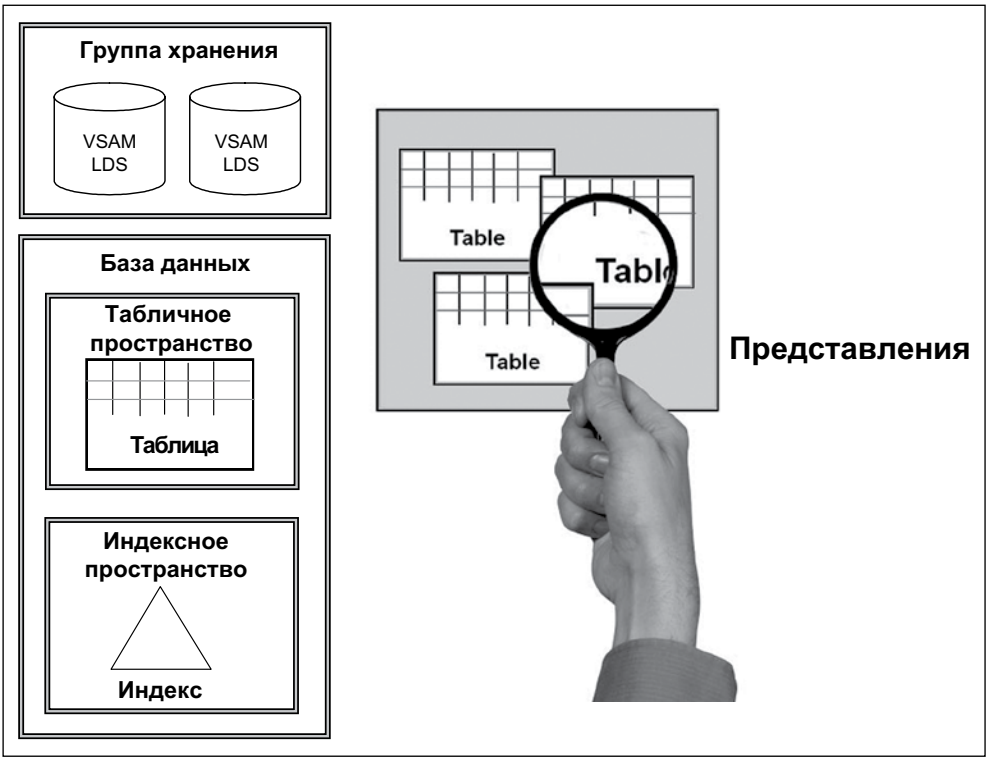


Рис. 12.3. Иерархия объектов в подсистеме DB2

12.7.2 Структуры схемы

Пользовательский тип данных (UDT)

Пользовательский тип данных (user-defined data type, UDT) является способом определения пользователями собственных типов данных вдобавок к обычным символьным и численным типам данных. Однако UDT основаны на уже существующих типах данных DB2. При работе с международными валютами скорее всего потребуется каким-то образом различать разные валюты. Используя определения UDT, можно на основании десятичного типа данных определить тип EURO как отдельный тип данных от YEN и US_DOLLAR. В результате, например, нельзя будет прибавить YEN к EURO, так как это разные типы данных.

Пользовательская функция (UDF)

Пользовательская функция (user-defined function, UDF) может быть достаточно просто определена на основе уже существующей функции DB2, такой как округление или

вычисление среднего, или может быть более сложной и представлять собой приложение с доступом посредством SQL-оператора. В примере с международными валютами можно было бы использовать UDF для конвертации одной валюты в другую, чтобы можно было выполнять арифметические функции.

Триггер

Триггер (*trigger*) определяет набор действий, выполняемых при выполнении операций вставки, изменения или удаления в определенной таблице. Например, предположим, что при каждой вставке сотрудника в таблицу EMP требуется также добавить единицу к счетчику сотрудников, находящемуся в таблице статистики компании. Можно определить триггер, срабатывающий при выполнении вставки в таблицу EMP. При таком срабатывании автоматически прибавляется единица в определенном столбце таблицы COMPANY_STATS.

Большой объект (LOB)

Большой объект (Large Object, LOB) представляет собой тип данных, используемый в DB2 для управления неструктурированными данными. Существует три типа больших объектов LOB:

- Двоичный большой объект (Binary Large Object, BLOB) – используется для фотографий и рисунков, музыкальных и звуковых записей, а также видеоклипов.
- Символьный большой объект (Character Large Object, CLOB) – используется для больших текстовых документов.
- Двухбайтовый символьный большой объект (Double Byte Character Large Object, DBCLOB) – используется для хранения больших текстовых документов, написанных на языках, требующих использования двухбайтовых символов, например, на японском (кандзи).

Большие объекты (LOB) хранятся в специальных вспомогательных таблицах, использующих специальное табличное пространство для больших объектов. Базовая таблица EMP может содержать такой текстовый материал, как резюме сотрудников. Так как резюме составляют большой объем данных, они содержатся в отдельной таблице. Столбец в таблице EMP, определенный как CLOB, содержит указатель на специальную вспомогательную таблицу LOB, хранящуюся в табличном пространстве LOB. Каждый столбец, определенный как LOB, имеет собственную ассоциативную вспомогательную таблицу и табличное пространство LOB.

Хранимая процедура

Хранимая процедура (*stored procedure*) представляет собой программу, написанную пользователем, которая обычно хранится и запускается на сервере (хотя может запускаться для решения локальных задач). Хранимые процедуры были специально разработаны для клиент-серверной среды, где клиенту требуется только лишь сделать одно обращение к серверу, который затем запускает хранимую процедуру, осуществляя доступ к данным DB2 и возвращая результаты. Это устраняет необходимость совершения нескольких сетевых обращений для выполнения отдельных запросов к базе данных, что может быть дорогостоящим.

Хранимую процедуру можно рассматривать как подпрограмму, которую можно вызвать для выполнения набора связанных функций. Она представляет собой приложение, определенное в DB2 и управляемое подсистемой DB2.

Системные структуры

Каталог и оглавление

DB2 управляет набором таблиц, содержащих метаданные или данные обо всех объектах DB2 в подсистеме. *Каталог* (*catalog*) хранит информацию обо всех объектах, в частности, о таблицах, представлениях, индексах, табличных пространствах и т. д., тогда как *оглавление* (*directory*) хранит информацию о приложениях. К каталогу можно направить запрос для просмотра информации об объектах; к оглавлению этого сделать нельзя.

При создании пользовательской таблицы, DB2 автоматически записывает имя таблицы, ее создателя, ее табличное пространство и базу данных в каталог и помещает эту информацию в таблицу каталога с именем SYSIBM.SYSTABLES. Все столбцы, определяемые в таблице, автоматически регистрируются в таблице SYSIBM.SYSCOLUMNS.

Кроме того, чтобы отметить наличие полномочий доступа к таблице у владельца таблицы, строка автоматически вставляется в SYSIBM.SYSTABAUTH. Любые индексы, создаваемые для таблицы, регистрируются в таблице SYSIBM.SYSINDEXES.

Буферные пулы

Буферные пулы (*buffer pools*) представляют области в виртуальной памяти, в которых DB2 временно хранит страницы табличных пространств или индексов. Они выступают в качестве кеша между DB2 и физическим дисковым устройством хранения, на котором находятся данные. Страница данных извлекается с диска и размещается в странице буферного пула. Если требуемые данные уже находятся в буфере, это позволяет избежать дорогостоящих операций ввода-вывода с диском.

Активные и архивные журналы

DB2 регистрирует все изменения данных и прочие значимые события в *журнале* (*log*). Эта информация используется для восстановления данных в случае отказа или для отмены изменений и возврата в прежнее состояние. DB2 записывает каждую запись журнала в набор данных, называемый *активным журналом* (*active log*).

Когда активный журнал заполнен, DB2 копирует его содержимое в набор данных на диске или магнитной ленте, называемый *архивным журналом* (*archive log*). Загрузочный набор данных (bootstrap data set) следит за активными и архивными журналами. DB2 использует эту информацию в сценариях восстановления, для перезапуска систем или для любых операций, требующих чтения журнала. Загрузочный набор данных позволяет осуществлять восстановление состояния на определенный момент времени.

12.7.3 Адресные пространства DB2

DB2 представляет собой подсистему, содержащую несколько адресных пространств, требующую, как минимум, три адресных пространства:

- системных служб;
- служб баз данных;
- служб менеджера блокировок (IRLM).

Кроме того, для связи с другими подсистемами DB2 используется утилита распределенных данных (Distributed Data Facility, DDF). Эти адресные пространства представлены на рис. 12.4.

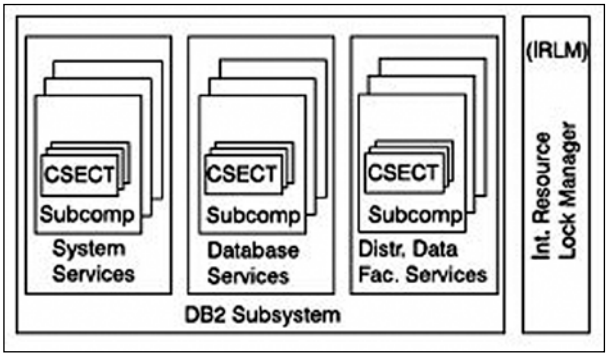


Рис. 12.4. Минимальный набор адресных пространств в DB2

12.7.4 Использование утилит DB2

В z/OS администратор базы данных управляет объектами базы данных, используя набор утилит и программ, передаваемых на выполнение с использованием JCL-заданий. Обычно компания имеет библиотеку наборов данных для этих заданий, копируемую и используемую администраторами баз данных. Однако существуют инструменты, генерирующие JCL-код, такие как Administration Tool и опция Utility на панели DB2I.

Утилиты помогают администраторам баз данных выполнять свою работу. Утилиты можно разделить на две категории.

- Утилиты организации данных.
  После создания таблиц администратор базы данных использует для их заполнения утилиту LOAD, способную сжимать большие объемы данных. Существует также утилита UNLOAD и программа DSNTIAUL, позволяющие администратору базы данных перемещать или копировать данные из одной подсистемы в другую.
  Утилита REORG позволяет хранить данные в определенном порядке. Последующие вставки и загрузки могут нарушить этот порядок, поэтому администратор базы данных должен планировать последующие запуски утилиты REORG на основании отчетов, генерируемых утилитой RUNSTATS, содержащих статистику и информацию о производительности. Можно также запускать RUNSTATS для системного каталога.
- Утилиты резервного копирования и восстановления.
  Крайне важно, чтобы администратор базы данных создавал копии образа данных и индексов утилитой COPY, чтобы можно было выполнить восстановление данных. Администратор базы данных может создать полную копию или инк-

рементальную копию (только для данных). Так как восстановление можно выполнить только с использованием полной копии, используется утилита MERGECOPY, осуществляющая объединение инкрементальных копий с полной копией. Утилита RECOVER позволяет выполнить восстановление на определенный момент времени с использованием копии образа. Чаще она используется для восстановления на момент создания копии образа, после чего применяется информация из журналов, регистрирующих все изменения данных, для восстановления до текущего момента. В случае отсутствия копии образа индекс воссоздается утилитой REBUILD INDEX.

- Утилиты согласованности данных.
  Одной из наиболее важных утилит согласованности данных является утилита CHECK, которую можно использовать для проверки и исправления ссылочной целостности и несогласованностей ограничений, особенно после дополнительного заполнения таблиц или после восстановления.
  Обычно утилиты используются следующим образом: сначала запускается RUNSTATS, затем EXPLAIN, затем снова RUNSTATS.

12.7.5 Использование команд DB2

Как системный администратор, так и администратор базы данных используют команды DB2 для мониторинга подсистемы. Панель DB2I и Administration Tool содержат средства ввода этих команд. Команда DISPLAY DATABASE отображает состояние всех табличных пространств и индексных пространств в базе данных. Например, без копии образа таблица может быть переведена в состояние *отложенного копирования* (*copy pending*), которое требует запуска утилиты COPY. Существует несколько других команд отображения состояния, например, DISPLAY UTILITY, показывающая состояние утилиты, а также команды отображения информации о буферном пуле, потоках и журналах.

Существуют также команды DSN, которые можно вводить из сеанса TSO или пакетного задания. Однако эти команды проще ввести, используя опции панели DB2I: BIND, DCLGEN, RUN и т. д. (в некоторых вычислительных центрах администраторы баз данных отвечают за связывание, хотя обычно оно выполняется программистами в составе задания компиляции).

12.8 Что такое SQL?

Язык SQL (Structured Query Language, язык структурированных запросов) представляет собой высокоуровневый язык, позволяющий сформулировать, какая информация нужна пользователю, не требуя знаний о том, как она извлекается. База данных отвечает за определение пути доступа, необходимого для извлечения данных. SQL работает на уровне множеств; другими словами, он предназначен для извлечения одной или нескольких строк. В сущности он используется для составления запросов к одной или нескольким таблицам и возвращает результаты в виде таблицы результатов.

SQL состоит из трех категорий, реализующих различные функции:

- DML (Data Manipulation Language, язык манипулирования данными) – используется для чтения и изменения данных;

- DDL (Data Definition Language, язык определения данных) – используется для определения, изменения или удаления объектов DB2;
- DCL (Data Control Language, язык управления данными) – используется для назначения и отмены полномочий.

Существует ряд инструментов для ввода и выполнения SQL-операторов. Мы рассмотрим SPUFI (SQL Processing Using File Input). SPUFI входит в панель меню DB2 Interactive (DB2I), которая вызывается из панели ISPF после установки DB2 (это, конечно, зависит от того, как системные администраторы настроят панели меню в системе).

Чаще всего администраторы баз данных используют инструмент SPUFI. Он позволяет создать и сохранить один или несколько SQL-операторов одновременно. Администраторы баз данных используют его для назначения или отмены полномочий, а иногда и для создания объектов, если это нужно сделать срочно. SPUFI также часто используется разработчиками для тестирования своих запросов. Это позволяет убедиться в том, что запрос возвращает точно то, что нужно.

Другим инструментом, который может использоваться на мэйнфрейме, является инструмент Query Management Facility (QMF), позволяющий ввести и сохранить только один SQL-оператор за раз. Основным преимуществом QMF является средство создания отчетов[1]. Оно позволяет разрабатывать гибкие и многократно используемые форматы отчетов, в том числе и графиков. Кроме того, этот инструмент включает функцию Prompted Query (запрос с подсказками), помогающую пользователям, не знакомым с SQL, составлять простые SQL-операторы. Другим инструментом является Administration Tool, который содержит возможности SPUFI, а также средство построения запросов.

На рис. 12.5 показан ввод SQL-операторов с использованием SPUFI. Следует выбрать первый пункт на панели DB2I. Обратите внимание на то, что рассматриваемая подсистема DB2 имеет имя DB8H.

```
                          DB2I PRIMARY OPTION MENU          SSID: DB8H
COMMAND ===> 1_

Select one of the following DB2 functions and press ENTER.

 1  SPUFI                  (Process SQL statements)
 2  DCLGEN                 (Generate SQL and source language declarations)
 3  PROGRAM PREPARATION    (Prepare a DB2 application program to run)
 4  PRECOMPILE             (Invoke DB2 precompiler)
 5  BIND/REBIND/FREE       (BIND, REBIND, or FREE plans or packages)
 6  RUN                    (RUN an SQL program)
 7  DB2 COMMANDS           (Issue DB2 commands)
 8  UTILITIES              (Invoke DB2 utilities)
 D  DB2I DEFAULTS          (Set global parameters)
 X  EXIT                   (Leave DB2I)

 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Рис. 12.5. Ввод SQL с использованием SPUFI

[1] QMF содержит функцию регулятора для ограничения процессорной мощности, используемой плохо построенным или требовательным запросом.

SPUFI осуществляет ввод и вывод файлов, поэтому необходимо наличие двух предварительно распределенных наборов данных.

- Первый набор данных, который может иметь имя ZPROF.SPUFI.CNTL, обычно представляет собой секционированный набор данных, предназначенный для содержания или сохранения пользовательских запросов в виде разделов. При использовании последовательного набора данных запись выполнялась бы поверх существующего SQL.
- Выходной файл, который может иметь имя ZPROF.SPUFI.OUTPUT, должен быть последовательным; это означает, что выходные данные следующего запроса записываются поверх существующего содержимого файла. Если нужно сохранить существующее содержимое, необходимо переименовать файл, используя средства редактирования в меню ISPF.

На рис. 12.6 представлено назначение наборов данных.

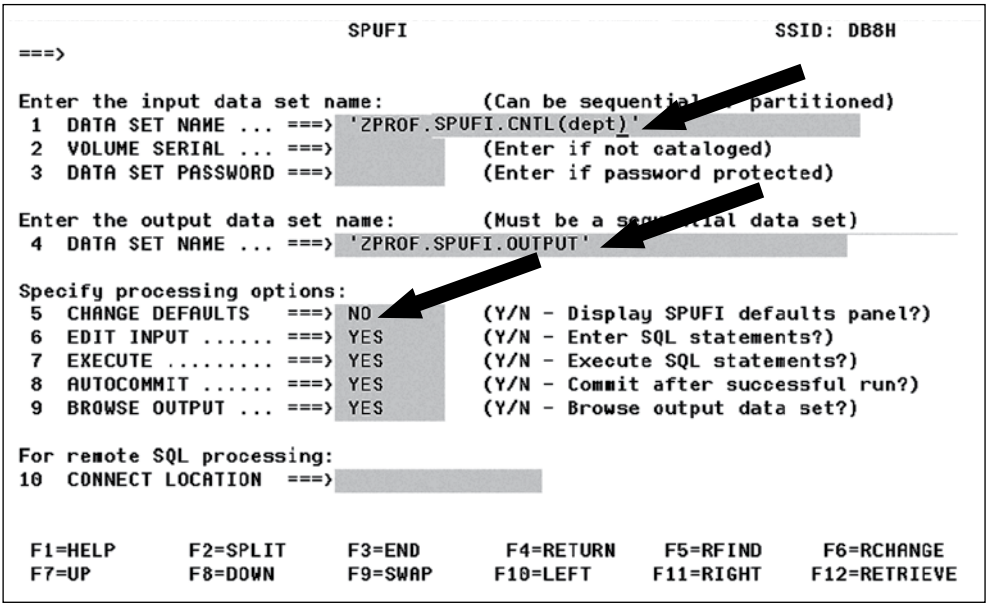


Рис. 12.6. Назначение наборов данных SPUFI

Обратите внимание на опцию 5, которую можно временно изменить на YES, чтобы просмотреть значения по умолчанию. Одним из значений, которое может потребоваться изменить, является максимальное количество извлекаемых строк.

Если установить для опции 5 значение NO и нажать клавишу Enter, SPUFI откроет входной файл ZPROF.SPUFI.CNTL(DEPT), в котором можно выполнить ввод или редактирование SQL-оператора. После ввода команды `recov on` и нажатия Enter предупреждение в верхней части экрана исчезает. Эта опция является частью профиля, описанного ранее в этой книге. Экран представлен на рис. 12.7.

Если в профиле установлено CAPS ON, тогда введенный SQL-оператор при нажатии Enter будет преобразован в верхний регистр. Но это не является обязательным.

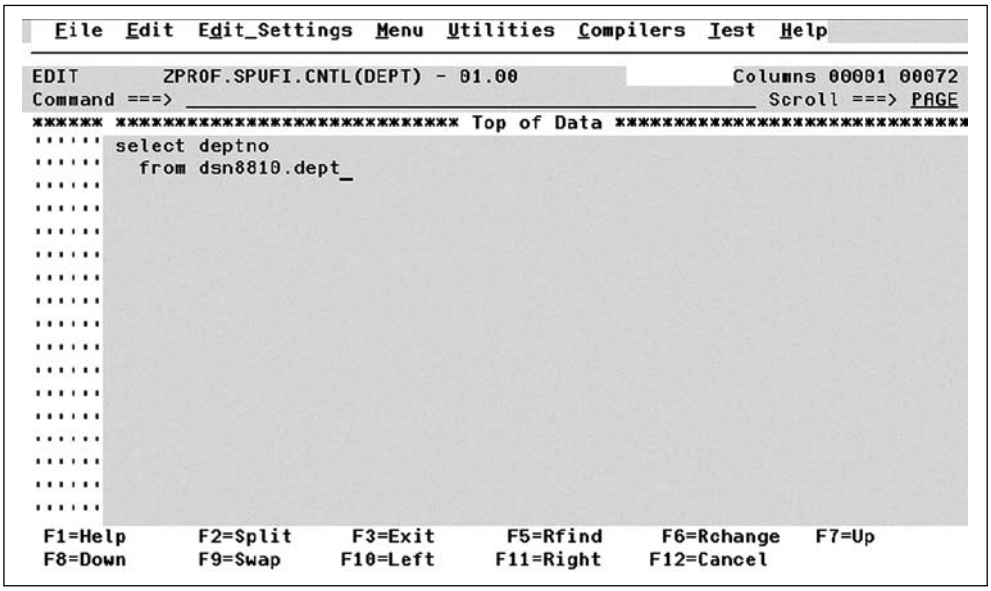


Рис. 12.7. Редактирование входного файла

Обратите внимание на то, что таблица имеет имя DSN8810.DEPT. Оно представляет собой составное имя таблицы, так как мы собираемся использовать примеры таблиц, созданные пользователем DSN8810.

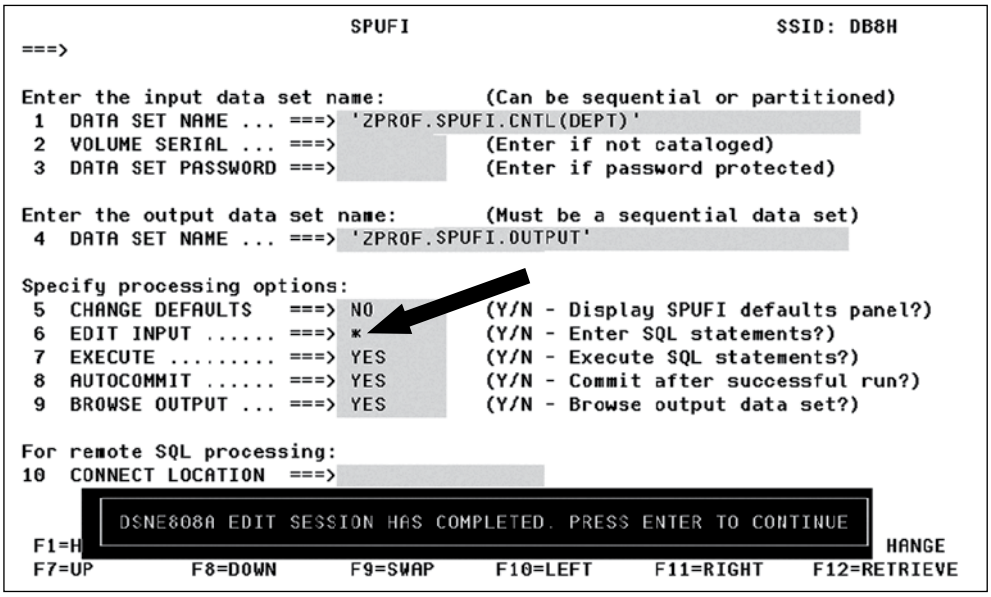


Рис. 12.8. Возврат в первую панель SPUFI

Если ввести только один SQL-оператор, тогда не требуется использовать указатель конца, в качестве которого в SQL используется точка с запятой (;), так как это задано в параметрах по умолчанию (это можно изменить; см. опцию 5 на предыдущем экране). Однако при вводе более одного SQL-оператора требуется использовать точку с запятой в конце каждого оператора, чтобы указать, что имеется несколько операторов.

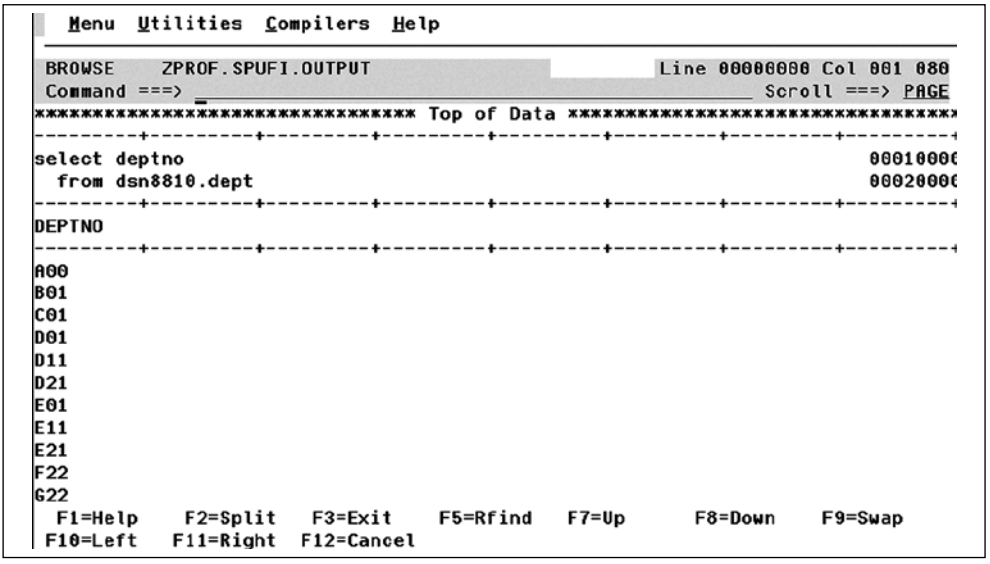


Рис. 12.9. Первая часть результатов запроса в SPUFI

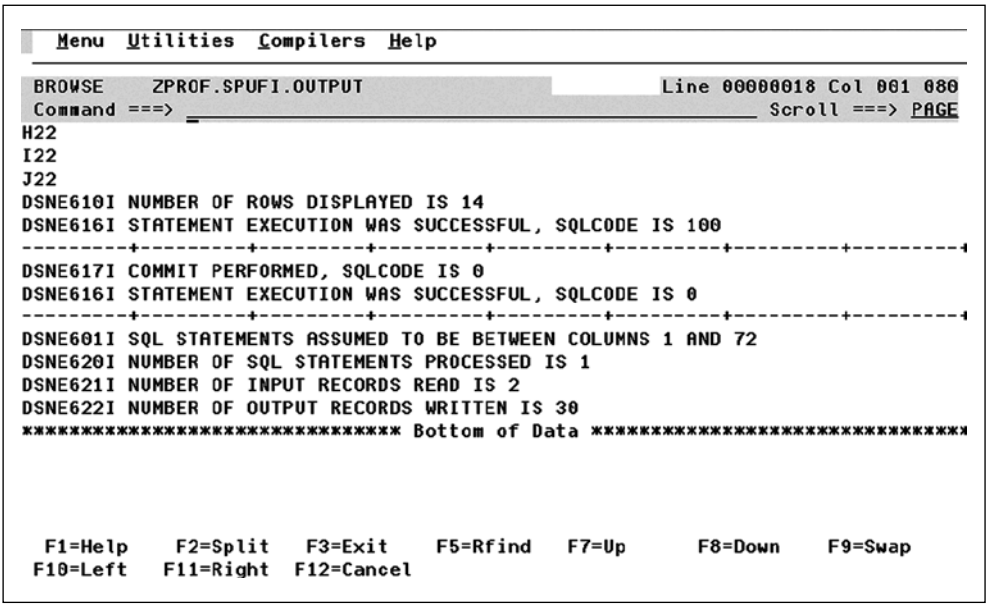


Рис. 12.10. Второй экран результатов

На данном этапе требуется возвратиться в первую панель SPUFI, нажав клавишу F3. Появится экран, представленный на рис. 12.8.

Обратите внимание на звездочку (*) в опции 6, вызванную тем, что редактирование SQL только что было завершено. Если на данном этапе нажать Enter, произойдет выполнение SQL-оператора, после чего автоматически откроется выходной файл, так как в пункте BROWSE OUTPUT установлено значение YES. Первая часть выходных данных представлена на рис. 12.9.

Для того чтобы вывести второй (в данном случае, последний) экран результатов, следует нажать F8, после чего появится экран, представленный на рис. 12.10.

Обратите внимание на то, что таблица результатов содержит только один столбец. Дело в том, что только этот столбец (DEPTNO) был указан в операторе SELECT. Значения в столбце DEPTNO были извлечены из всех (14) строк таблицы. Было выдано также несколько сообщений. Одно из них содержит количество извлеченных строк. Другое указывает, что SQLCODE (код завершения SQL, указывающий успешное или неуспешное завершение) равен 100, что означает конец файла и отсутствие каких-либо еще результатов.

Дополнительные сведения. Дополнительные сведения о языке SQL см. в руководстве *DB2 UDB for z/OS: SQL Reference*. Эту и другие публикации IBM можно найти на веб-сайте z/OS Internet Library:

```
http://www.ibm.com/servers/eserver/zseries/zos/bkserv/
```

12.9 Программирование приложений для DB2

SQL не является полным языком программирования, однако он необходим для доступа и управления данными в базе данных DB2. Он представляет собой непроцедурный язык 4-го поколения (4GL), разработанный в середине 1970-х для использования в DB2. SQL можно либо использовать динамически в интерпретационной программе вроде SPUFI, либо встраивать и компилировать или ассемблировать в составе основного языка.

Так как же написать приложение, осуществляющее доступ к данным в DB2?

Для этого SQL встраивается в исходный код языка программирования, такого как Java, Smalltalk, REXX, C, C++, COBOL, Fortran, PL/I и высокоуровневый ассемблер. Существует две категории SQL-операторов, которые можно использовать в программе: статические и динамические.

- Статические.
  Статическими SQL-операторами называются полные SQL-операторы, записанные внутри исходного кода. В процессе подготовки программы, DB2 определяет пути доступа для операторов, и они записываются в DB2. SQL-операторы не изменяются от одного запуска программы к другому; при этом используются одни и те же заданные пути доступа, так что подсистеме DB2 не требуется создавать их повторно, что может вызвать дополнительную нагрузку (**Примечание**. Все SQL-операторы должны иметь путь доступа).
- Динамические.
  Динамическими SQL-операторами называются SQL-операторы, частично или полностью неизвестные на момент написания программы. Только после запус-

ка программы подсистема DB2 знает, какие операторы будут использоваться, и способна определить требуемые пути доступа. Пути доступа не сохраняются, так как операторы могут изменяться от одного запуска программы к другому. Примером такой программы является SPUFI. SPUFI в действительности представляет собой приложение, принимающее динамические SQL-операторы. Это те операторы, которые вводятся во входном файле. При каждом запуске SPUFI, SQL-операторы могут быть разными, поэтому в приложение встраиваются специальные подготовительные SQL-операторы.

Ниже мы рассмотрим статические SQL-операторы, чтобы получить представление о процессах, происходящих при использовании DB2. Добавим, что все может выглядеть сложным, однако использование каждого действия является достаточно обоснованным.

12.9.1 Подготовка программы в DB2: алгоритм

Традиционный процесс подготовки программы, состоящий из компиляции и редактирования связей, при подготовке SQL-кода должен включать дополнительные этапы, так как компиляторы не распознают SQL. Эти этапы, наряду с компиляцией и редактированием связей, можно выполнять через панель DB2I, хотя весь процесс, за исключением вызова DCLGEN, обычно выполняется в одном потоке заданий JCL. При чтении объяснений см. рис. 12.11.

DCLGEN

DCLGEN представляет способ автоматического генерирования исходных определений для объектов DB2, используемых в программе. Они создаются в разделе библиотеки DCLGEN, которую можно включить в вашу исходную программу. Если ее не включить, необходимо составлять определения вручную. Администратор базы данных DB2 обычно создает их на основе правил компании. На этом этапе необходима работающая система DB2, так как определения берутся из каталога DB2.

Прекомпиляция

Так как компиляторы не могут обрабатывать SQL-код, на этапе прекомпиляции SQL-выражения заключаются в комментарии, подставляя оператор CALL обращения к DB2. Он передает некоторые параметры, например, адреса переменных основного языка (для извлечения данных), номера операторов и (что очень важно) измененную отметку времени, называемую *маркером согласованности* (*consistency token*; часто называется просто отметкой времени). На этом этапе не требуется запуск системы DB2; все выполняется без доступа к DB2.

Прекомпилятор определяет SQL-код по специальным флагам начала и конца, которые должны быть включены для каждого SQL-оператора. Во всех языках программирования используется одинаковый флаг начала – EXEC SQL. Флаги конца различаются. В COBOL используется END-EXEC. (точка), тогда как в C и других языках используется точка с запятой. Ниже приведен пример на языке COBOL:

```
EXEC SQL
SELECT EMPNO, LASTNAME
```

```
INTO :EMPNO, :LASTNAME
FROM EMP
END-EXEC.
```

В этом примере, EMPNO и LASTNAME извлекаются в переменные основного языка, перед которыми указывается двоеточие. Переменными основного языка являются переменные, определенные на основном языке (COBOL, PL/1 и т. д.) – языке включающем SQL-код. На этапе вызова DCLGEN также происходит определение этих переменных. В данном случае имя переменной основного языка соответствует имени столбца, что не является обязательным требованием; можно использовать любое имя с типом данных, совместимым с типом данных столбца.

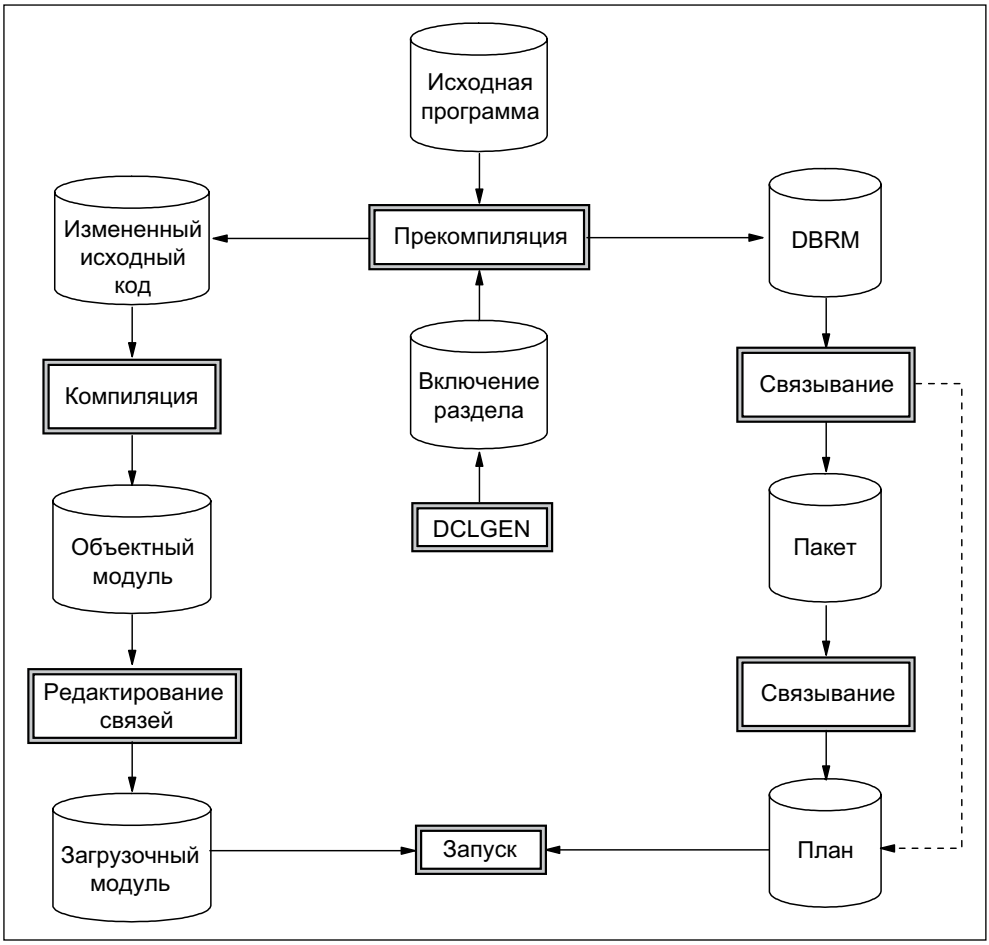


Рис. 12.11. Алгоритм подготовки программы

После прекомпиляции программа разделяется на две части.

- Измененный исходный код; представляет первоначальный исходный код, в котором SQL-код заключен в комментарии и замещен операторами CALL.

- Модуль запросов базы данных (database request module, DBRM), который обычно представляет собой раздел библиотеки PDS и содержит SQL-операторы программы.

Измененный исходный код передается компилятору для компиляции и редактирования связей, вследствие чего создается исполняемый загрузочный модуль, как для любой другой программы, не содержащей SQL.

Кстати, можно вставить в программу любой тип SQL-кода: DML, DDL и DCL, если только не нарушаются правила авторизации.

Связывание

В DB2 связывание можно рассматривать как аналог процесса компиляции для DBRM. Связывание выполняет три операции.

- Выполняет проверку синтаксиса на наличие ошибок.
- Выполняет проверку полномочий.
- Что важнее всего, определяет пути доступа для операторов. DB2 содержит компонент, называемый *оптимизатором* (*optimizer*), который рассматривает различные способы осуществления доступа к данным, такие как сканирование целой таблицы, использование индекса, выбор используемого индекса и т. д. Он определяет показатели стоимости каждого пути доступа и выбирает путь доступа с наименьшей стоимостью. Такой оптимизатор называется оптимизатором на основе стоимости (в отличие от оптимизатора на основе правил).

SQL-код вместе с путем доступа (и маркером согласованности – отметкой времени) хранится как пакет в оглавлении DB2. Другая информация, в частности информация о пакете и действительный SQL-код, хранится в каталоге. При связывании создается исполняемый SQL-код для одного приложения в пакете. Теперь DB2 имеет всю информацию, необходимую для получения запрашиваемых данных для данной программы.

Часто программы вызывают подпрограммы, которые также содержат SQL-вызовы. Каждая из этих подпрограмм также имеет пакет. Необходимо сгруппировать всю информацию DB2 вместе. Для этого необходим еще один этап: еще одно связывание, но на этот раз предназначенное для создания *плана* (*plan*).

Даже если подпрограмма не используется, все равно требуется создать план. План может содержать не только информацию о программе, но и другую информацию. Это представляет обычную практику: план содержит все пакеты одного проекта, и при каждом запуске используется один и тот же план.

Для полноты следует добавить, что изначально DBRM связывались непосредственно в планы (они назывались встроенными DBRM). Однако при внесении небольшого изменения в одну из программ, необходимо выполнить повторное связывание всего плана. Это также необходимо сделать и в случае добавления индекса.

Как вы знаете, в процессе связывания DB2 обновляет свои оглавление и каталог. Обновление означает недопущение их обновления другими людьми (данные заблокированы и недоступны для них), так что выполнение большинства других действий в DB2 является практически невозможным. Для того чтобы обойти это ограничение,

были реализованы пакеты. Теперь необходимо только лишь выполнить повторное связывание одного пакета, так что длительность обновления очень мала, и воздействие на других пользователей почти отсутствует. В настоящее время все еще используются планы с встроенными DBRM, хотя большинство компаний выполняет их преобразование в пакеты.

Планы используются только в среде мэйнфреймов. На других платформах они не используются.

Запуск

При выполнении приложения происходит загрузка загрузочного модуля в основную память. Когда встречается SQL-оператор, обращение к DB2 (оператор CALL), заменивший SQL-оператор, передает его параметры в DB2. Одним из параметров является маркер согласованности. Этот же маркер (или отметка времени) присутствует и в пакете. Затем выполняется просмотр пакетов в заданном плане DB2 в поисках наличия соответствующей отметки времени, после чего загружается и выполняется подходящий пакет. Таким образом, для запуска необходимо в качестве параметра задать имя плана.

Последнее замечание: результатом SQL-выражения обычно является результирующее множество (содержащее больше одной строки). Приложение может обрабатывать только одну запись или строку за раз. В DB2 добавляется специальная конструкция, называемая *курсором* (*cursor*), который, в сущности, аналогичен указателю и позволяет во встроенном SQL-коде выбирать, обновлять или удалять по одной строке из результирующего множества за один раз.

Дополнительные сведения. Дополнительные сведения см. в публикации *DB2 UDB for z/OS: Application Programming and SQL Guide*.

12.10 Функции IMS Database Manager

Система управления базами данных (СУБД) предоставляет средства доступа транзакций или процессов бизнес-приложений к хранимой информации. Роль СУБД состоит в том, чтобы обеспечить следующие функции:

- осуществление доступа множества пользователей к данным с использованием одной копии данных;
- управление одновременным доступом к данным для обеспечения целостности всех обновлений;
- уменьшение зависимостей методов доступа от аппаратных устройств и операционной системы;
- снижение избыточности данных посредством использования только одной копии данных.

12.11 Структура подсистемы IMS Database

IMS Database Manager представляет собой центральный пункт управления и доступа к данным приложений. IMS содержит полный набор утилит для реализации всех функций в составе продукта IMS. В этом разделе описываются различные типы адрес-

ных пространств z/OS и связи между ними. Основой подсистемы IMS является *управляющий регион* (*control region*), запущенный в одном адресном пространстве z/OS. Он содержит множество зависимых адресных пространств, запущенных в других регионах и реализующих дополнительные функции для управляющего региона, или в которых выполняются IMS-приложения.

Помимо управляющего региона некоторые приложения и утилиты, используемые в IMS, выполняются в отдельных пакетных адресных пространствах. Они отделены от подсистемы IMS и ее управляющего региона и никак с ними не связаны.

Исторически сложилось так, что в некоторых документах, описывающих IMS, термин «регион» используется для описания адресного пространства z/OS (например, управляющий регион IMS). В этой книге термин «регион» используется в соответствии с общеупотребительным значением. Можно считать термин «регион» синонимом термина «адресное пространство z/OS».

На рис. 12.12 представлена подсистема IMS DB/DC. Дополнительные сведения см. в книге *An Introduction to IMS* (ISBN 0-13-185671-5).

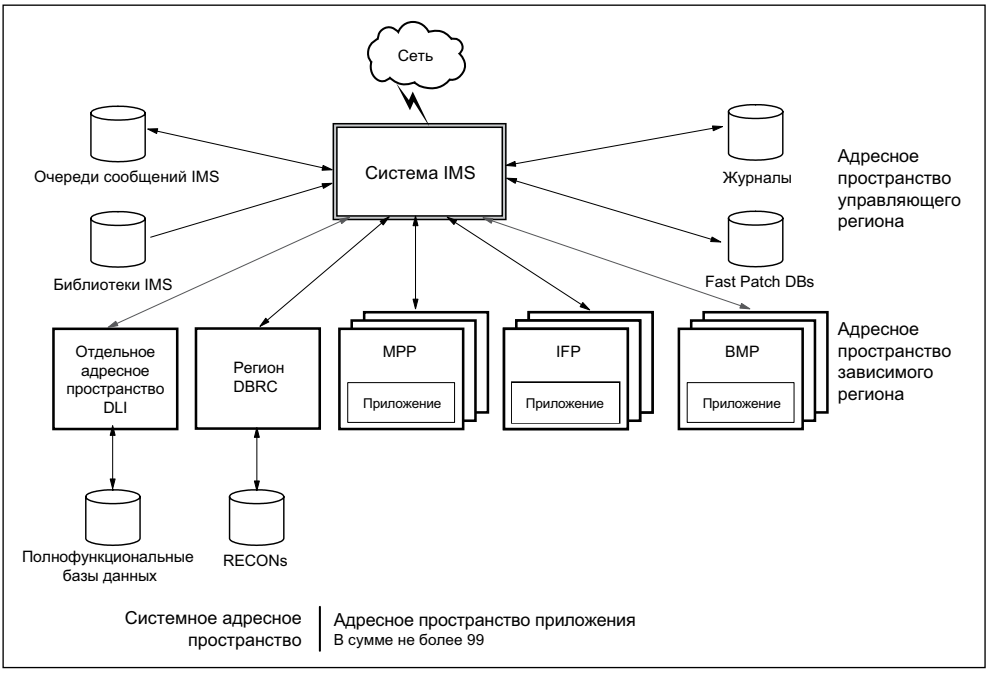


Рис. 12.12. Структура подсистемы IMS DB/DC

12.11.1 Иерархическая модель базы данных IMS

IMS в качестве базового метода хранения данных использует иерархическую модель, которая представляет собой прагматический способ хранения данных и реализации отношений между различными типами сущностей.

В этой модели отдельные типы сущностей реализуются как сегменты в иерархической структуре. Иерархическая структура определяется проектировщиком базы данных на основе взаимоотношений между сущностями и путей доступа, требуемых приложениями.

Обратите внимание на то, что в IMS использование термина «база данных» несколько отличается от других СУБД. В IMS термин «база данных» используется для описания реализации одной иерархии, при этом приложение обычно осуществляет доступ к большому количеству баз данных IMS. С точки зрения реляционной модели в IMS база данных приблизительно эквивалентна таблице.

DL/I позволяет использовать большое разнообразие структур данных. Максимальное число типов сегментов составляет 255 на одну иерархическую структуру данных. В иерархической структуре данных может быть определено не больше 15 уровней сегментов. Не существует ограничений по количеству экземпляров каждого типа сегментов, кроме налагаемых физическими ограничениями метода доступа.

Последовательность доступа к сегментам

Переход по иерархии осуществляется сверху вниз, слева направо, спереди назад (для двойников).

Номера кодов сегментов не учитывают дубликаты, и обработка записей базы данных осуществляется в иерархической последовательности. Включаются все сегменты записи базы данных, так что дубликаты могут существовать в иерархических последовательностях. Сегменты могут содержать поля последовательности, определяющие порядок их хранения и обработки.

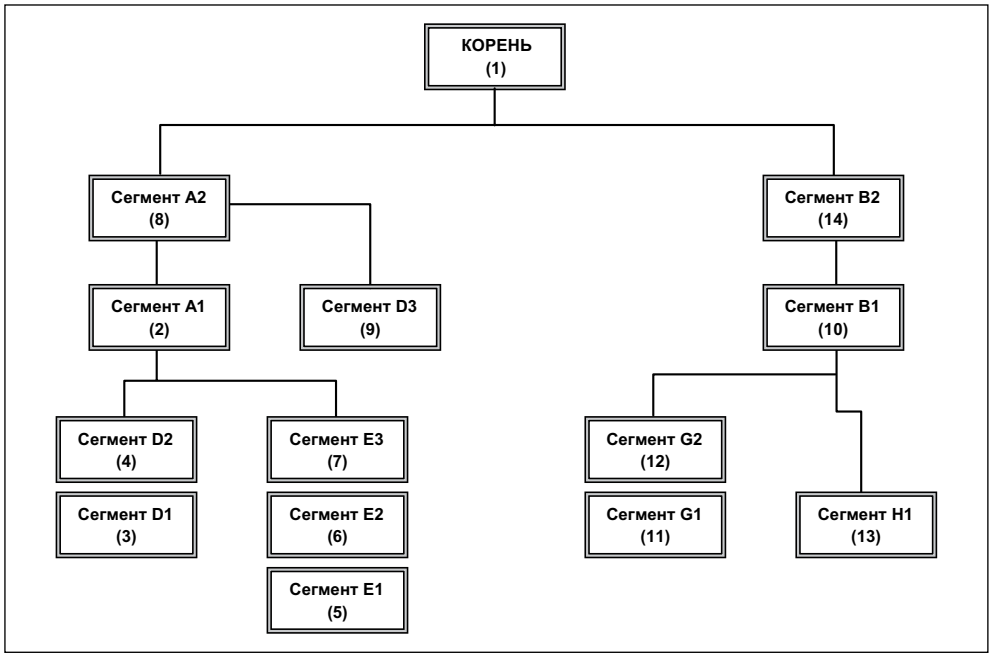


Рис. 12.13. Последовательность

Иерархическая структура данных, представленная на рис. 12.13, описывает данные в одной записи базы данных таким образом, каким они представляются приложению. Она не представляет физического хранения данных. Физическое хранение не имеет отношения к приложению.

Основным строительным элементом иерархической структуры данных является отношение типа «родитель-потомок» между сегментами данных, которое также изображено на рис. 12.13.

12.11.2 Использование служб z/OS в IMS

Система IMS разработана таким образом, чтобы наилучшим образом использовать возможности операционной системы z/OS.

- Она выполняется в нескольких адресных пространствах

  Подсистемы IMS (за исключением пакетных приложений и утилит IMS/DB) обычно состоят из адресного пространства управляющего региона, зависимых адресных пространств, обеспечивающих работу системных служб, и зависимых адресных пространств для приложений. Работа в нескольких адресных пространствах дает следующие преимущества:

  - максимизация использования процессоров при работе на многопроцессорном CPC;
  - возможность параллельной диспетчеризации адресных пространств на разных процессорах;
  - изоляция приложений от системного кода IMS и сокращение простоев при отказах приложений.

- Она выполняет по несколько задач в каждом адресном пространстве

  IMS, особенно в управляющих регионах, создает несколько подзадач z/OS для различных выполняемых функций. Это позволяет осуществлять диспетчеризацию других подзадач IMS в z/OS, пока одна подзадача IMS ожидает системного обслуживания.

- Она использует межпространственные службы z/OS для связи между различными адресными пространствами, составляющими подсистему IMS. Она также использует общую системную область z/OS (Common System Area, CSA) для хранения управляющих блоков IMS, к которым часто осуществляется доступ из адресных пространств, составляющих подсистему IMS. Это сокращает нагрузку, связанную с использованием нескольких адресных пространств.

- Она использует интерфейс подсистем z/OS для обнаружения отказа зависимых адресных пространств и предотвращения отмены зависимых адресных пространств, а также для взаимодействия с другими подсистемами, такими как DB2 и WebSphere MQ.

- Она может использовать сисплекс z/OS (обсуждается далее в этой книге). Несколько подсистем IMS могут выполняться в системах z/OS, составляющих сисплекс, осуществляя доступ к одним и тем же базам данных IMS. Это обеспечивает следующие преимущества:

- повышение доступности – системы z/OS и подсистемы IMS можно включать и отключать, не прерывая обслуживания;
- повышение производительности – несколько подсистем IMS могут обслуживать гораздо более интенсивную нагрузку.

12.11.3 Эволюция IMS

Изначально, все оперативные приложения IMS/DB использовали IMS/TM в качестве интерфейса к базе данных. Однако с ростом популярности DB2 многие клиенты начали разрабатывать оперативные приложения, используя в качестве базы данных DB2, продолжая при этом использовать уже существующие приложения. Этим объясняется существование большого количества смешанных сред.

12.11.4 Пример оперативной обработки

Возвращаясь к примеру туристического агентства, приведенному в главе 11 «Системы управления транзакциями в z/OS», можно привести следующие примеры IMS-транзакций, используемых в авиакомпании:

- некоторые пакетные задания требуют ежедневного обновления, например платежи, осуществляемые туристическими агентствами и другими клиентами;
- другое пакетное задание может осуществлять отправку напоминаний туристическим агентствам и другим клиентам о необходимости внесения платежей;
- проверка выполнения (и оплаты) бронирования может осуществляться в оперативном приложении;
- проверка наличия свободных мест.

12.12 Заключение

Данные можно хранить в обычном файле, однако при этом обычно происходит множественное дублирование, что может вызвать несогласованность данных. Поэтому лучше создавать центральные базы данных, к которым можно осуществлять доступ (для чтения и изменения) из различных мест. Поддержка согласованности, безопасности и т. д. выполняется системой управления базами данных; пользователям и разработчикам не приходится об этом беспокоиться.

Реляционная база данных представляет основной способ организации данных в современном деловом мире. В СУБД DB2 производства IBM реализованы такие понятия реляционных баз данных, как первичные ключи, ссылочная целостность, язык доступа к базе данных (SQL), пустые значения и нормализованная структура. В реляционных базах данных основной структурой является таблица, состоящая из столбцов и строк.

В DB2 имеет место иерархическая зависимость от основных объектов. В структуре таблицы можно создавать индексы и представления. При удалении таблицы эти объекты также удаляются. Таблицы находятся в физическом наборе данных, называемом табличным пространством, связанным с базой данных, представляющей собой логический набор табличных пространств. К новым объектам схемы в DB2 относятся UDT, UDF, LOB, триггеры и хранимые процедуры.

DB2 также содержит системные структуры, позволяющие управлять подсистемой. Каталог и оглавление содержат метаданные обо всех объектах в РСУБД. Буферные пулы используются для хранения страниц данных из дискового хранилища в целях их быстрого извлечения; активные или архивные журналы и BSDS являются способами регистрации всех изменений в данных, выполняемых в целях восстановления.

Единственным способом доступа к данным в базах данных DB2 является использование SQL. SQL не является полным языком программирования, он работает на уровне множеств, используя результирующую таблицу при управлении данными. SQL состоит из трех категорий, реализующих различные функции: DML, DDL и DCL. На мэйнфреймах для ввода SQL-операторов используется инструмент SPUFI.

Для использования SQL в приложениях необходимо выполнить некоторые дополнительные действия, так как традиционные компиляторы языков третьего поколения (3GL) не распознают SQL. Прекомпилятор заключает SQL-операторы в комментарии, копирует их в DBRM, снабжая маркерами согласованности, и заменяет их обращениями к DB2. Затем выполняется компиляция и редактирование связей в измененном исходном коде. DBRM выполняет процесс связывания, во время которого определяется путь доступа и исполняемый SQL-код сохраняется в пакете. После этого пакеты логически связываются с планом. При запуске обращение к DB2 в загрузочном модуле передает свой маркер согласованности в DB2 для сопоставления с SQL-кодом в соответствующем плане и выполнения SQL-кода.

SQL может содержать статические и динамические операторы, и для определения пути доступа, выбранного оптимизатором для SQL-кода, можно использовать оператор EXPLAIN.

Оператор EXPLAIN определяет путь доступа запроса в целях повышения его производительности. Операторы EXPLAIN особенно полезны для составления запросов к многотабличным базам данных с множественным доступом.

Основные термины в этой главе

Полнофункциональная база данных	DL/I	Измененный исходный код
SPUFI	SQL	SYSADM
EXPLAIN	Представление	DBMS
Многозадачность	Многопоточность	Администратор базы данных

12.13 Контрольные вопросы

Чтобы проверить понимание материала этой главы, ответьте на следующие вопросы:

1. Назовите некоторые обязанности администратора базы данных?
2. Какие объекты DB2 определяют физическую область хранения? Это таблицы?
3. Какие сложности могут возникнуть в следующем SQL-операторе?

```
SELECT *
FROM PAYROLL;
```

4. Какую категорию SQL вы бы использовали для определения объектов в DB2?
5. Каким образом прекомпилятор обнаруживает SQL-оператор в программе?

6. Каким образом загрузочный модуль повторно объединяется с SQL-операторами?
7. Каким образом можно определить, какой путь доступа выбрал оптимизатор? Какой процесс создает этот путь доступа?
8. Что такое хранимая процедура?
9. Назовите некоторые обязанности системного администратора?
10. Назовите некоторые обязанности администратора базы данных?
11. Назовите некоторые способы обеспечения безопасности в DB2?
12. Какова структура базы данных в IMS-DB? Опишите ее.

12.14 Упражнение 1 – Использование SPUFI в COBOL-программе

Для выполнения этого упражнения необходимо подключение к DB2.

12.14.1 Этап 1: Создание файлов

Прежде чем начать выполнение упражнения для DB2, необходимо создать два дополнительных PDS:

- ZUSER##.DB2.INCLUDE – для хранения DCLGEN;
- ZUSER##.DB2.DBRM – для хранения DBRM.

В качестве базового набора данных можно использовать ZUSER##.LANG.CNTL.

Кроме того, требуется также файл ZUSER##.SPUFI.OUTPUT, представляющий собой простой файл формата VB с длиной записи 4092 и длиной блока 4096.

12.14.2 Этап 2: DCLGEN

DCLGEN представляет простой способ генерирования COBOL-операторов определений для объектов DB2, используемых в программе. Эти операторы затем можно включить в исходную программу.

Во-первых, в меню DB2I (DB2 Interactive) нужно выбрать D, что соответствует DB2I Defaults (см. рис. 12.14), и нажать Enter.

На панели DB2I Defaults Panel 1 задайте для опции 3 (Application Language) значение IBMCOB (рис. 12.15).

Нажмите Enter, и на панели DB2I Defaults Panel 2 задайте значение DEFAULT для опции 3 (COBOL string delimiter) и значение G для опции 3 (DBCS symbol). Нажмите Enter (см. рис. 12.16).

Это нужно для того, чтобы убедиться в наличии правильного языка.

После нажатия Enter происходит возврат на главную панель DB2I (см. рис. 12.14); теперь следует выбрать опцию 2 (DCLGEN).

Необходимо наличие уже выделенного целевого набора данных для содержания определения DCLGEN (ZUSER##.DB2.INCLUDE); он должен быть создан для вас. Если он отсутствует, перейдите в меню ISPF и создайте файл PDS.

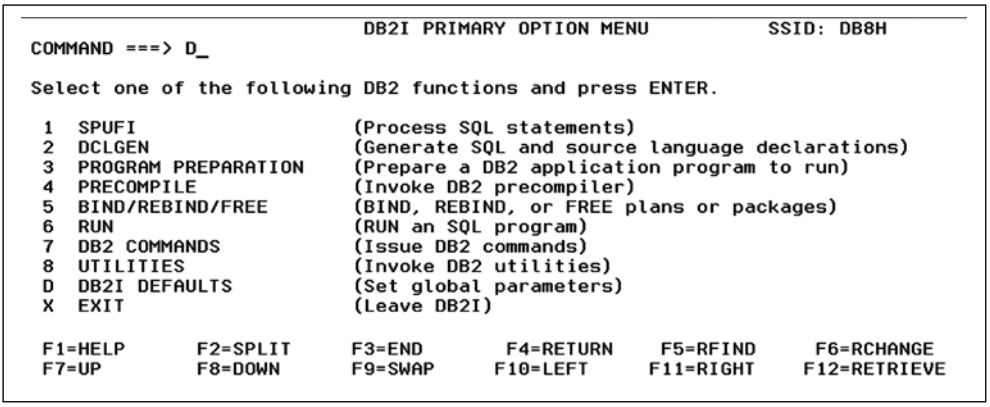


Рис. 12.14. Меню DB2

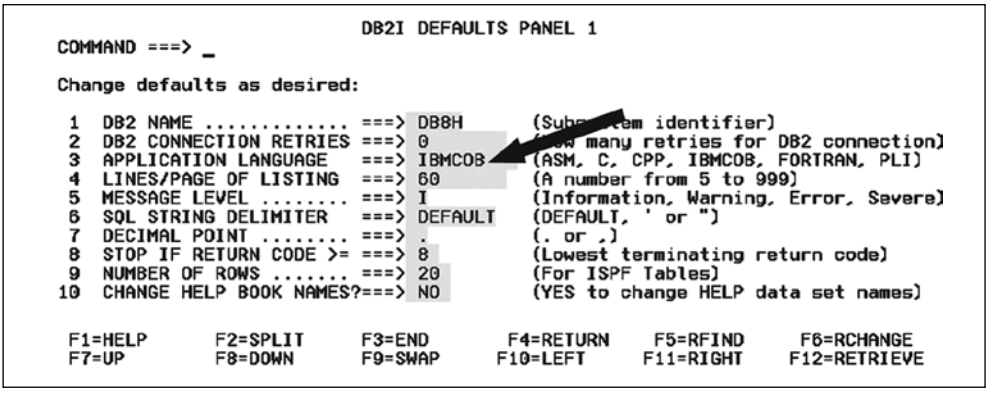


Рис. 12.15. Панель DB2I Defaults Panel 1

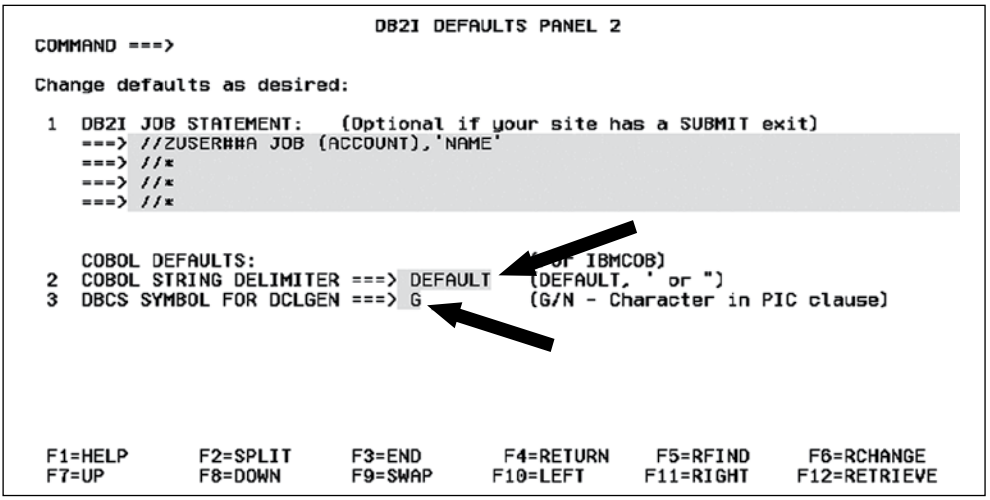


Рис. 12.16. Панель DB2I Default panel 2

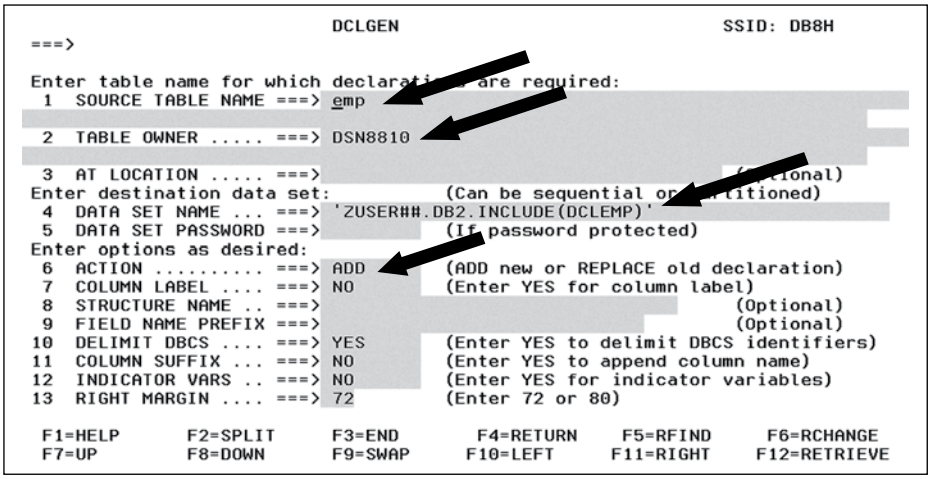


Рис. 12.17. DCLGEN

Как показано на рис. 12.17, необходимо задать таблицу, владельца таблицы, PDS-файл и действие ADD. Результирующее сообщение должно иметь следующий вид:

```
EXECUTION COMPLETE, MEMBER DCLEMP ADDED
***
```

При изменении определения таблицы необходимо также изменить DCLGEN и использовать REPLACE.

12.14.3 Этап 3: Тестирование SQL-кода

Перейдите в SPUFI; используйте PDS-файл SPUFI.CNTL. В этом PDS-файле найдите раздел SELECT. Он представляет SQL-оператор, используемый в программе. Выражение where отсутствует, что позволяет просмотреть все результаты, которые можно получить. Кроме того, он позволяет увидеть отделы, заданные в таблице.

Конечно же в более сложных запросах это является общепринятой практикой. Как разработчику приложений, вам важно убедиться, что вы выполняете правильный SQL-код.

12.14.4 Этап 4: Создание программы

На данном этапе можно создать программу или использовать программу, содержащуюся в LANG.SOURCE(COBDB2). Этот пример программы определяет среднюю зарплату по одному отделу. Нужно указать отдел, и программа выдаст результат. Чтобы завершить выполнение программы, введите `999`.

Для того чтобы изменить эту программу, добавьте следующее:

- ваши переменные (включите раздел, созданный на этапе 1),
- Укажите SQL-разделители для языка COBOL.

Позиции для их добавления отмечены символами «???».

12.14.5 Этап 5: Завершение создания программы

Откройте задание LANG.CNTL(COBDB2) и внесите изменения, указанные в начале задания.

Задание содержит следующие шаги:

Шаг PC: представляет этап прекомпиляции в DB2; на этом этапе происходит разделение исходного кода на две части: DBRM и измененный исходный код.

Шаги COB, PLKED и LKED: представляют этапы компиляции и редактирования связей измененного исходного кода.

Шаг BIND: представляет связывание пакета и плана.

Вопрос: Если бы потребовалось изменить программу, какое связывание можно было бы не выполнять? Вы можете свободно изменять программу. Вместо среднего значения, можно запросить минимальную или максимальную заработную плату в отделе (потребуется только лишь изменить SQL).

Шаг Run: запускает программу в пакете для двух отделов: A00 и D21.

12.14.6 Этап 6: Запуск программы из TSO

Вместо того чтобы запускать программу в пакете, попробуйте запустить ее из приглашения TSO READY. Для этого необходимо распределить оба файла для сеанса (это нужно сделать до запуска задания).

Введите следующие команды, нажимая Enter после каждой строки:

```
TSO alloc da(*) f(sysprint) reuse
tso alloc da(*) f(sysin) reuse
```

Затем вернитесь в экран DB2I.

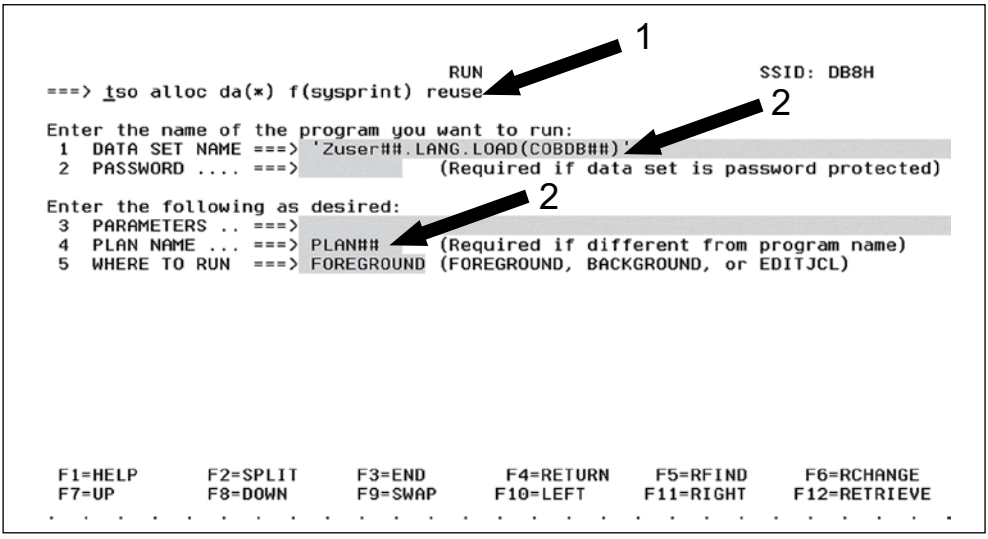


Рис. 12.18. Готово к выполнению

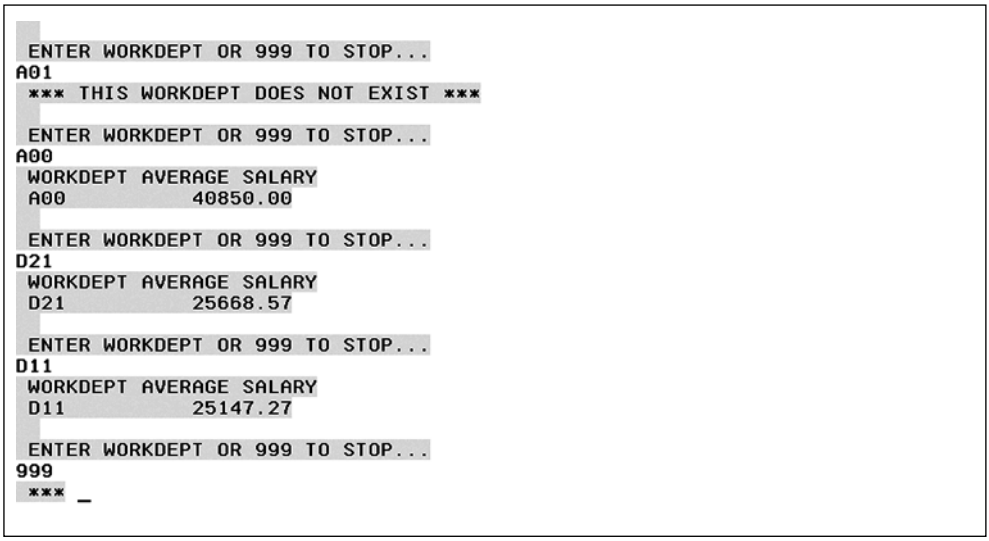


Рис. 12.19. Выполнение программы

Выберите опцию 6 RUN. Здесь следует ввести имя файла и имя плана (см. рис. 12.18).

13

z/OS HTTP Server

Цель. Как специалисту в сфере мэйнфреймов, вам нужно знать, каким образом выполнить развертывание веб-приложения в z/OS, и как настроить z/OS на обслуживание веб-задач.

После завершения работы над этой главой вы сможете:

- назвать три серверных режима;
- описать, что такое статические и динамические веб-страницы;
- назвать, по меньшей мере, по две функции из следующих групп: базовые функции, функции безопасности и функции кеширования.

13.1 Введение в веб-задачи в z/OS

По мере того как предприятия переносят многие свои приложения в Интернет, организации, использующие мэйнфреймы, сталкиваются с трудностями обеспечения и управления новыми веб-задачами в дополнение к более традиционным задачам, таким как пакетная обработка.

В следующих главах рассматривается, каким образом программные продукты промежуточного уровня используются для предоставления основных функций, необходимых для обеспечения обработки веб-задач в z/OS.

- Глава 13 «z/OS HTTP Server».
- Глава 14 «WebSphere Application Server в z/OS».
- Глава 15 «Обмен сообщениями и управление очередями».

В примерах, представленных в этих главах, используются продукты компании IBM, однако на современном рынке существует множество подобных программных продуктов промежуточного уровня.

13.2 Что такое z/OS HTTP Server?

z/OS HTTP Server обрабатывает статические и динамические веб-страницы. HTTP Server имеет те же возможности, что и любой другой веб-сервер, однако он также содержит некоторые возможности, характерные для z/OS. HTTP Server может работать в любом из нижеперечисленных трех режимов, каждый из которых имеет свои преимущества при обслуживании веб-задач:

Автономный сервер	Этот режим обычно используется в реализациях, использующих только HTTP Server (простых веб-сайтах). Его основная роль состоит в том, чтобы обеспечить ограниченную видимость из Интернета.
Масштабируемый сервер	Этот режим обычно используется на интерактивных веб-сайтах, где объем трафика растет или снижается динамически. Он предназначен для более сложной среды, в которой вызываются сервлеты и JSP.
Несколько серверов	Этот режим представляет собой сочетание автономного и масштабируемого серверов, позволяющее повысить масштабируемость и безопасность в системе. Например, автономный сервер можно использовать в качестве шлюза к масштабируемым серверам, и шлюз может выполнять аутентификацию всех запросов пользователей и перенаправлять запросы на другие серверы.

13.2.1 Обработка статических веб-страниц в z/OS

При использовании веб-сервера в z/OS, например HTTP Server, обработка статических веб-страниц осуществляется подобно веб-серверам на других платформах. Пользователь отправляет HTTP-запрос на HTTP Server, чтобы получить определенный файл. HTTP Server извлекает файл из своего файлового хранилища и передает его пользователю вместе с информацией о файле (в частности, о его MIME-типе и размере) в HTTP-заголовке.

Тем не менее, HTTP Server имеет важное отличие от других веб-серверов. Так как системы z/OS используют кодировку EBCDIC, документы в z/OS необходимо сначала

преобразовать в формат ASCII, обычно используемый в Интернете (двоичные документы, например изображения, преобразовывать не требуется).

HTTP Server выполняет такие преобразования, освобождая программиста от выполнения этого этапа. Однако программист должен использовать FTP для загрузки документов на сервер. Другими словами, программист задает ASCII в качестве формата передачи в FTP, чтобы выполнить преобразование файла из EBCDIC. При двоичной передаче преобразование файла не выполняется.

13.2.2 Обработка динамических веб-страниц в z/OS

Динамические веб-страницы являются важным элементом веб-коммерции. Каждый вид взаимодействия и персонализации требует динамического содержимого. Например, когда пользователь заполняет форму на веб-сайте, необходимо выполнить обработку данных в этой форме и отправить результаты пользователю.

В z/OS применяется два подхода к обработке динамических веб-страниц:

- использование CGI для динамических веб-страниц;
- использование интерфейса подключаемых модулей.

Использование CGI для динамических веб-страниц

Один из способов формирования и передачи динамических веб-страниц состоит в использовании CGI (Common Gateway Interface) – интерфейса, входящего в протокол HTTP. CGI является стандартным способом передачи HTTP-запроса веб-пользователя в приложение. GCI генерирует выходные данные и передает их в HTTP Server, который отправляет их пользователю в HTTP-ответе (рис. 13.1).

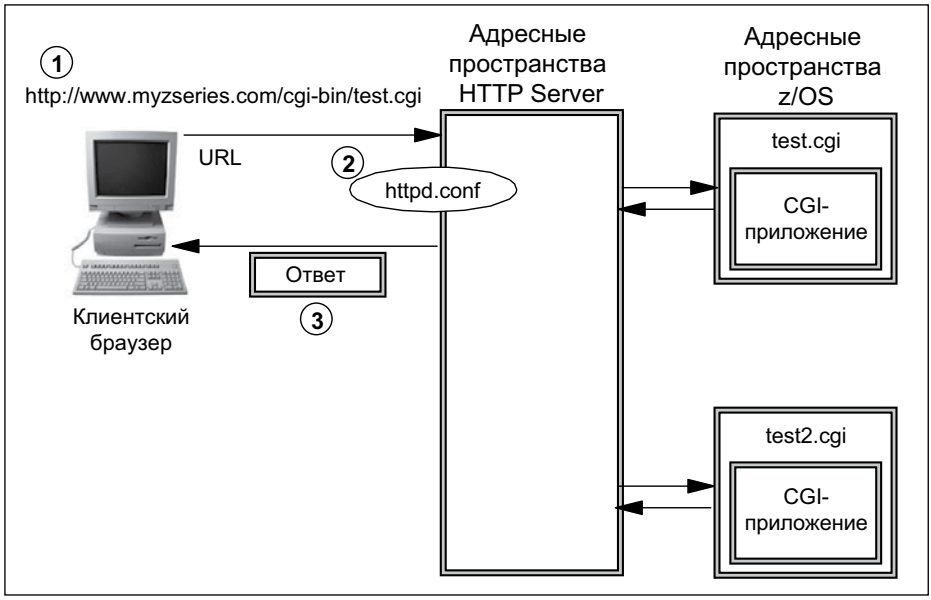


Рис. 13.1. Как работает CGI

Возможности CGI не ограничены только лишь возвратом HTML-страниц; приложение может также создавать простые текстовые документы, XML-документы, изображения, PDF-документы и т. д. MIME-тип должен отражать содержимое HTTP-ответа.

CGI имеет существенный недостаток, заключающийся в том, что каждый HTTP-запрос требует отдельного адресного пространства. Это вызывает снижение эффективности при большом количестве одновременных запросов.

Чтобы устранить эту проблему, был создан FastCGI. Подключаемый модуль FastCGI для HTTP Server представляет собой программу, управляющую несколькими CGI-запросами в едином адресном пространстве, экономя множество программных инструкций для каждого запроса. Дополнительные сведения о подключаемых модулях для HTTP Server см. в разделе «Использование интерфейса подключаемых модулей».

Использование интерфейса подключаемых модулей

Другой способ предоставления динамического содержимого состоит в использовании интерфейса подключаемых модулей (plug-in interface) HTTP Server, который позволяет одному или нескольким продуктам взаимодействовать с HTTP Server. Ниже, например, представлены некоторые способы, используя которые HTTP Server может передавать управление WebSphere.

- Подключаемый модуль WebSphere, одно адресное пространство.
  На рис. 13.2 представлена простая конфигурация, в которой не требуется использовать сервер J2EE™. Этот сервлет может осуществлять подключение к CICS или IMS, либо к DB2 через JDBC™. Однако не рекомендуется составлять код бизнес-логики внутри сервлетов.

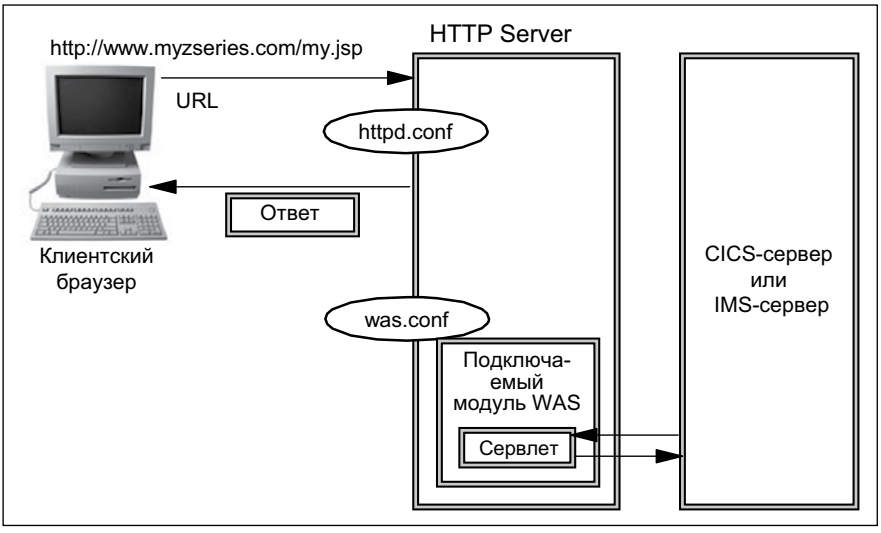


Рис. 13.2. Доступ к сервлетам через подключаемый модуль WebSphere

- Веб-контейнер внутри HTTP Server, отдельный EJB™-контейнер
  На рис. 13.3 показана более распространенная конфигурация, в которой сервлеты выполняются в адресном пространстве, отличном от EJB, так что EJB вы-

зываются удаленными вызовами. Затем EJB получают информацию с других серверов.

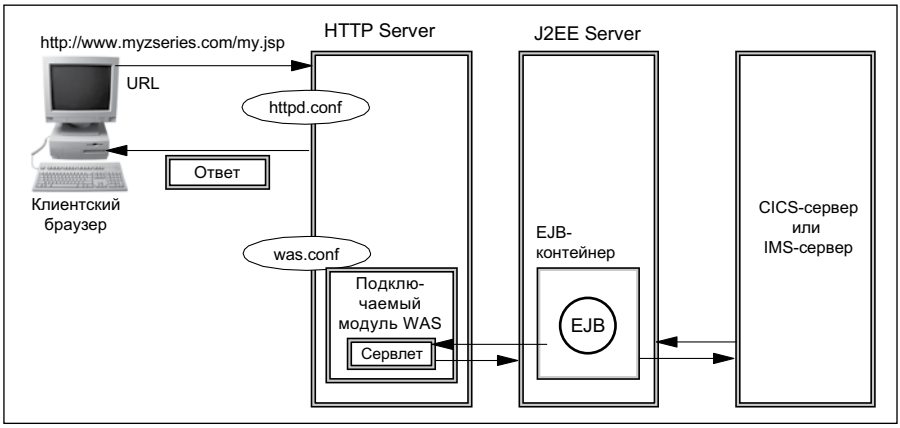


Рис. 13.3. Доступ к EJB из подключаемого модуля WebSphere

- Отдельный J2EE-сервер с веб-контейнером и EJB-контейнером.
  Помимо локального выполнения сервлетов в подключаемом модуле WebSphere, можно также использовать подключаемый модуль WebSphere для удаленного выполнения сервлетов в веб-контейнере, как показано на рис. 13.4. Это позволяет расположить сервлеты и EJB в одном адресном пространстве z/OS, что устраняет необходимость удаленных вызовов EJB.

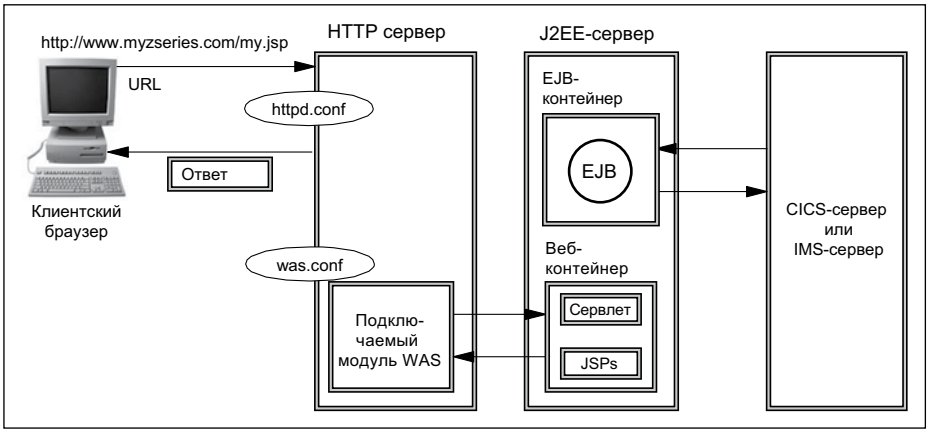


Рис. 13.4. Доступ к сервлетам в веб-контейнере с использованием подключаемого модуля WebSphere

Если уже используется WebSphere Application Server, использование HTTP Server может быть ненужным, хотя существует несколько способов взаимодействия HTTP Server с WebSphere Application Server. Эти способы рассматриваются ниже.

13.3 Возможности HTTP Server

HTTP Server содержит такие же возможности, как и другие веб-серверы, однако дополнительно содержит некоторые функции, характерные для z/OS. Функции, характерные для z/OS, можно разделить на следующие группы:

- базовые функции;
- функции безопасности;
- функции кеширования файлов.

13.3.1 Базовые функции

- Доступ к файлам EBCDIC/ASCII.
  Сервер осуществляет доступ к файлам и, при необходимости, преобразует их из EBCDIC в кодировку ASCII.
- Мониторинг производительности и использования.
  Среди возможностей, характерных для z/OS, HTTP Server может генерировать записи для средства управления системой (System Management Facilities, SMF[1]), которые системный программист может впоследствии просматривать для выполнения анализа производительности и использования.
- Трассировка и ведение журналов.
  HTTP Server поставляется с полным набором средств ведения журналов, трассировки и создания отчетов, позволяющих отследить каждый HTTP-запрос.
- SSI (Server Side Include).
  Функция SSI (Server Side Include) позволяет вставлять информацию в документы (статические или динамические), которые сервер отправляет клиентам. Это может быть переменная (например, дата последнего изменения), выходные данные программы или содержимое другого файла. Включение этой функции без ее использования может оказать серьезное воздействие на производительность.
- SNMP MIB (Simple Network Management Protocol Management Information Base).
  HTTP Server содержит SNMP MIB и субагент SNMP, что позволяет использовать любую систему управления сетью с поддержкой SNMP для мониторинга исправности, пропускной способности и активности сервера. Она может уведомлять о превышении заданных пороговых значений.
- Поддержка cookie-файлов.
  Так как HTTP является протоколом без запоминания состояния, можно реализовать запоминание состояния посредством использования cookie-файлов, хранящих информацию на стороне клиента. Такая поддержка полезна для множества веб-страниц, например, для получения модифицированных документов или для ротации баннеров.
- Многоформатная обработка.
  Эта функция используется для персонализации веб-страниц. Браузер отправляет заголовочную информацию вместе с запросом, включая *заголовок приня-*

[1] SMF относится к дополнительным возможностям z/OS и обеспечивает средства сбора и записи информации, которую можно использовать для оценки использования системы в целях учета, предъявления претензий или настройки производительности.

мия (*accept header*). Эта информация включает язык пользователя. HTTP Server может использовать содержимое заголовка принятия для выбора нужного документа, возвращаемого клиенту.

- Постоянные подключения.
  При использовании этой функции, характерной для HTTP/1.1, не каждый запрос должен устанавливать новое подключение. Постоянные подключения некоторое время остаются активными, что позволяет использовать их для последующих запросов.
- Виртуальные хосты.
  Виртуальные хосты позволяют запустить один веб-сервер, тогда как с точки зрения клиентов все будет выглядеть так, как будто запущено несколько веб-серверов. Это достигается посредством использования разных DNS-имен для одного IP-адреса и/или разных IP-адресов для одного сервера HTTP Server.

13.3.2 Функции безопасности

- Безопасность на уровне потоков.
  Для каждого потока, выполняющегося под управлением HTTP Server, можно установить независимую среду безопасности; это в сущности означает, что каждый клиент, подключающийся к серверу, имеет собственную среду безопасности.
- Поддержка HTTPS/SSL.
  HTTP Server обеспечивает полную поддержку протокола SSL (Secure Socket Layer). HTTPS использует SSL в качестве подуровня обычного уровня HTTP для шифрования и дешифрования HTTP-запросов и HTTP-ответов. HTTPS использует порт 443, тогда как HTTP использует порт 80.
- Поддержка LDAP.
  Протокол LDAP (Lightweight Data Access Protocol) представляет собой упрощенный способ извлечения информации из X.500-совместимого каталога с использованием асинхронного клиент-серверного протокола.
- Аутентификация с использованием сертификатов.
  В составе средств поддержки SSL, HTTP Server может использовать аутентификацию с использованием сертификатов и выступать в качестве центра сертификации.
- Поддержка прокси-сервера.
  HTTP Server может выступать в качестве прокси-сервера. При этом, однако, нельзя использовать Fast Response Cache Accelerator (FRCA).

13.3.3 Кеширование файлов

Производительность можно значительно увеличить посредством использования возможностей кеширования (буферизации) файлов:

- кеширование HFS-файлов в HTTP Server;
- кеширование наборов данных z/OS в HTTP Server;
- кеширование HFS-файлов в z/OS UNIX;
- Fast Response Cache Accelerator (FRCA).

13.3.4 Код подключаемого модуля

Подключаемый модуль WebSphere HTTP Server представляет собой код, выполняющийся на разных веб-серверах: IBM HTTP Server, Apache, IIS, Sun Java™ System. Запросы передаются в подключаемый модуль, где они обрабатываются на основе файла конфигурации.

Подключаемый модуль представляет собой код, поставляемый с WebSphere, выполняющийся на разных HTTP-серверах. В качестве HTTP-серверов может использоваться IBM HTTP Server в z/OS. По мере поступления запросов в HTTP Server используются директивы в файле конфигурации HTTP Server (httpd.conf) для принятия следующего решения: должен ли входящий рабочий запрос обрабатываться HTTP Server или же его необходимо передавать непосредственно в подключаемый модуль.

После поступления запроса в подключаемый модуль логика, обрабатывающая запрос, определяется файлом конфигурации подключаемого модуля, а не файлом конфигурации HTTP Server. Этот файл конфигурации по умолчанию имеет имя plugin-cfg.xml. В этом файле определяется информация о том, на какой из поддерживаемых серверов приложений должен передаваться запрос. Этот файл создается сервером WebSphere Application Server и не обязательно требует изменений, хотя они допускаются.

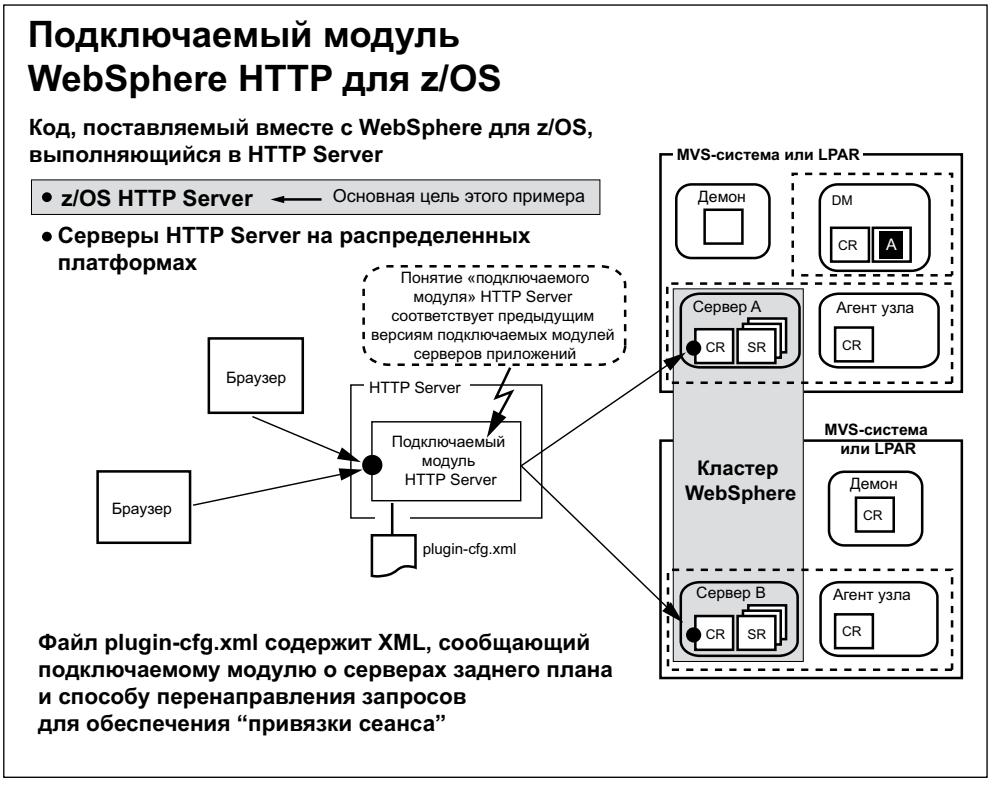


Рис. 13.5. Пример подключаемого модуля

Примечание. В общем, подключаемые модули обеспечивают расширения функциональных возможностей в HTTP Server. На рис. 13.5 представлен пример их использования, хотя существует множество различных подключаемых модулей, которые можно настроить таким образом, чтобы облегчить настройку веб-среды. Другим популярным подключаемым модулем является Lightweight Directory Access Protocol Server (LDAP), используемый для аутентификации в системе безопасности.

13.4 Заключение

z/OS содержит HTTP Server, обрабатывающий как статические, так и динамические веб-страницы. HTTP Server поддерживает подключаемый модуль WebSphere (который обслуживает EJB-контейнеры и J2EE), а также функции безопасности и кеширования файлов. Эти функции упрощают работу с динамическими веб-страницами.

Основные термины в этой главе

CGI	Динамическая веб-страница	FRCA
HTTP	J2EE	LDAP
SSL	Статическая веб-страница	

13.5 Контрольные вопросы

Чтобы проверить понимание материала этой главы, ответьте на следующие вопросы:

1. Назовите три серверных режима.
2. Опишите, что такое статические и динамические веб-страницы.
3. Назовите, по меньшей мере, по две функции из следующих групп: базовые функции, функции безопасности и функции кеширования.

13.6 Упражнения

Для выполнения этого упражнения следует использовать оболочку ISHELL или OMVS. Кроме того, необходимо знать следующее:

- размещение конфигурационного файла httpd.conf;
- IP-адрес или имя сервера HTTP Server.

Выполните следующие действия и ответьте на вопросы:

1. Просмотрите файл httpd.conf продукта HTTP Server, установленного в z/OS. В каком каталоге хранятся веб-документы (F “URL translation rules”)? Скажите также, какой порт следует использовать? (F “Port directive”)?
2. В окне веб-браузера выведите учебный HTTP Server. Каким образом осуществляется подключение WebSphere к HTTP Server? (F “Websphere”)?
3. Используйте OEDIT для создания HTML-документа в папке веб-документов. Назовите его *youridtest*`.html`. Ниже приведен пример:

```
<!doctype html public "//W3//Comment//EN">
```

```
<html>
<head>
<META content="text/html; charset=iso-8859-1">
<title> This is a simple HTML Exercise</title>
</head>
<body bgcolor="#FFFFFF">
<p>Hello World
</body>
</html>
```

4. Откройте свой HTML-документ в веб-браузере, например:

   `www.yourserver.com/`*`youridtest`*`.html`

   Что нужно сделать для того, чтобы установить собственный CGI?

5. Просмотрите файл httpd.conf. Включена ли опция HTCounter CGI «Date and Time»?

   Если да, измените *`youridtext`*`.html` и добавьте следующую строку в раздел body:

   `<img src=»/cgi-bin/datetime?Timebase=Local»>`

   Сохраните файл. Что изменилось?

14

WebSphere Application Server в z/OS

Цель. Как специалисту в сфере мэйнфреймов, вам нужно знать, каким образом выполнить развертывание веб-приложения в z/OS, и как настроить z/OS на обслуживание веб-задач.

После завершения работы над этой главой вы сможете:

- перечислить шесть свойств модели приложений J2EE;
- перечислить три причины выбора системы z/OS в качестве платформы для WebSphere Application Server;
- назвать три коннектора к CICS, DB2 и IMS.

14.1 Что такое WebSphere Application Server для z/OS?

По мере того как предприятия переносят многие свои приложения в Интернет, организации, использующие мэйнфреймы, сталкиваются с трудностями обеспечения и управления новыми веб-задачами в дополнение к более традиционным задачам, таким как пакетная обработка.

WebSphere Application Server представляет собой всестороннюю сложную прикладную систему, основанную на технологии J2EE (Java 2 Enterprise Edition) и вебслужбах. WebSphere Application Server в z/OS является реализацией J2EE, соответствующей текущей спецификации SDK (Software Development Kit), поддерживающей приложения на уровне API. Как уже говорилось, она представляет собой среду развертывания и выполнения Java-приложений, построенную на технологии, основанной на открытых стандартах, поддерживающей все основные функции, в частности, сервлеты, Java Server Pages (JSP) и Enterprise Java Beans (EJB), включая последние технологии интеграции служб и интерфейсов.

Среда выполнения сервера приложений тесно интегрирована со всеми внутренними функциями и службами z/OS. Сервер приложений может взаимодействовать со всеми основными подсистемами операционной системы, включая DB2, CICS и IMS. Она имеет множество атрибутов, связанных с безопасностью, производительностью, масштабируемостью и восстановлением. Кроме того, сервер приложений использует сложные функции администрирования и настройки, обеспечивая «прозрачную» интеграцию в любой центр обработки данных или серверную среду.

WebSphere Application Server представляет собой среду развертывания приложений электронной коммерции. Она построена на технологиях, основанных на открытых стандартах, таких как CORBA, HTML, HTTP, IIOP и J2EE-совместимых Java-стандартах для сервлетов, JSP™ (Java Server Pages) и EJB (Enterprise Java Beans), и поддерживает все Java API, необходимые для совместимости с J2EE.

Управляющее адресное пространство (controller address space) автоматически запускает служебный регион (servant region) при поступлении нагрузки. Как показано на рис. 14.1, экземпляр сервера приложений состоит из управляющего региона (CR) и одного или нескольких служебных регионов (SR).

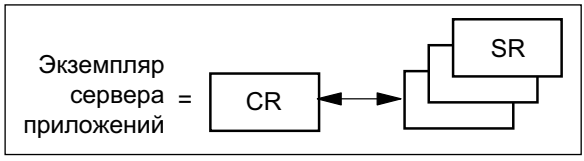


Рис. 14.1. Экземпляр сервера приложений

Сервер приложений в z/OS поддерживает два типа конфигураций: базовую (Base) и сетевое развертывание (Network Deployment). Каждая конфигурация использует примерно одинаковую архитектурную иерархию, состоящую из *серверов* (*servers*), *узлов* (*nodes*) и *ячеек* (*cells*). Однако ячейки и узлы играют важную роль только в конфигурации сетевого развертывания.

14.2 Серверы

Сервер является первичным компонентом времени выполнения; на нем происходит выполнение приложения. Сервер содержит контейнеры и службы, специализирующиеся на обеспечении выполнения определенных компонентов Java-приложений. Каждый сервер приложений выполняется в отдельной виртуальной машине Java (Java Virtual Machine, JVM).

В зависимости от конфигурации серверы могут работать отдельно или в сочетании следующим образом:

- В конфигурации Base (базовой) каждый сервер приложений функционирует как отдельная сущность. Поэтому между серверами приложений не осуществляется распределение нагрузки или общее администрирование.
- К конфигурации Network Deployment (сетевого развертывания) несколько серверов приложений обслуживаются с центрального пункта администрирования.

Кроме того, можно осуществлять кластеризацию серверов приложений для распределения рабочей нагрузки.

> **Примечание.** В этой книге не рассматривается специальный тип сервера приложений, называемый JMS Server.

14.3 Узлы (и агенты узлов)

Узел представляет собой логическую группу серверных процессов под управлением WebSphere, имеющих общую конфигурацию и оперативное управление. Узел обычно связан с одной физической инсталляцией сервера приложений.

При переходе к более сложным конфигурациям серверов приложений появляются такие понятия, как конфигурирование нескольких узлов с одного общего сервера администрирования и распределение нагрузки между узлами. В таких конфигурациях централизованного управления, каждый узел имеет агент узла, взаимодействующий с менеджером развертывания (Deployment Manager) при управлении процессами администрирования.

14.4 Ячейки

Ячейка представляет собой группу узлов, объединенных в один административный домен. В базовой конфигурации ячейка содержит один узел. Этот узел может содержать множество серверов, однако файлы конфигурации для каждого сервера хранятся и обслуживаются отдельно (на основе XML).

При использовании конфигурации Network Deployment ячейка может содержать несколько узлов, администрируемых с единого пункта. Файлы конфигурации и приложений для всех узлов ячейки централизованно хранятся в главном хранилище (repository) конфигурации ячейки. Это централизованное хранилище управляется процессом менеджера развертывания и синхронизируется с локальными копиями, хранящимися на каждом узле.

В адресных пространствах, используемых сервером приложений, используется понятие *контейнеров* (*containers*), обеспечивающих разделение между различными выполняющимися элементами во время выполнения. Отдельный контейнер, называемый EJB-контейнером, используется для выполнения Enterprise Java Beans. Другой контейнер, называемый веб-контейнером, используется для выполнения веб-элементов, таких как HTML, GIF-файлы, сервлеты и Java Server Pages (JSP). Вместе они составляют среду выполнения сервера приложений в JVM.

14.5 Модель приложений J2EE в z/OS

В z/OS модель приложений J2EE ничем не отличается от других платформ; она соответствует спецификации SDK, демонстрируя следующие свойства:

- функциональность – соответствие пользовательским требованиям;
- надежность – работа в изменяющихся условиях;
- практичность – обеспечение простого доступа к функциям приложений;
- эффективность – оптимальное использование системных ресурсов;
- обслуживаемость – возможность легкого внесения изменений;
- переносимость – возможность переноса из одной среды в другую.

WebSphere Application Server в z/OS поддерживает четыре основные модели структуры приложения: модель веб-вычислений, модель интегрированных корпоративных вычислений, модель многопоточных распределенных бизнес-вычислений и модель сервис-ориентированных вычислений. Все эти модели структуры сосредоточены на отделении логики приложения от базовой инфраструктуры; другими словами, физическая топология и явный доступ к информационной системе отделены от модели программирования приложения.

Модель программирования J2EE, поддерживаемая в WebSphere Application Server для z/OS, упрощает создание приложений для новых бизнес-требований, так как она отделяет детали от базовой инфраструктуры. Она осуществляет развертывание компонентной и сервис-ориентированной модели программирования, реализованной в J2EE.

14.6 Выполнение WebSphere Application Server в z/OS

WebSphere Application Server работает как стандартная подсистема в z/OS. Поэтому она использует все преимущества мэйнфреймов и функциональные возможности, соответствующие рассматриваемой платформе, в частности, ее уникальную способность параллельного выполнения сотен разнородных задач и соответствие целям уровня сервиса, определенным пользователем.

14.6.1 Консолидация нагрузки

Как говорилось в предыдущих главах, мэйнфрейм можно использовать для консолидации нагрузки с множества отдельных серверов. Поэтому если имеет место большая административная нагрузка или проблемы физической мощности набора отдельных серверов, мэйнфрейм может взять на себя роль единой серверной среды, управляю-

щей этой нагрузкой. Он может представить единую среду администрирования, управления производительностью и восстановления приложений, использующих службы мэйнфрейма во время выполнения.

Набор серверов приложений можно без труда перенести на один логический раздел мэйнфрейма, обеспечивая простоту управления и мониторинга (логические разделы, или LPAR, рассматриваются в главе 2 «Аппаратные системы мэйнфрейма и высокая доступность»). Консолидация позволяет также осуществлять измерение и сбор показателей, упрощая анализ мощности.

14.6.2 Безопасность WebSphere для z/OS

Сочетание аппаратной и программной безопасности zSeries, вместе с встроенной безопасностью J2EE, обеспечивает существенную защиту от возможных вторжений. Система безопасности продукта представляет собой многоуровневую архитектуру, построенную на основе систем безопасности операционной системы, Java Virtual Machine (JVM) и Java2.

WebSphere Application Server для z/OS осуществляет интеграцию инфраструктуры и механизмов защиты конфиденциальных J2EE-ресурсов и административных ресурсов предприятия с точки зрения сквозной безопасности, основанной на отраслевых стандартах.

Открытая архитектура обеспечивает защищенную связь и взаимодействие со всеми информационными системами предприятия (Enterprise Information Systems, EIS) на мэйнфрейме, куда входят:

- CICS Transaction Server (TS);
- DB2;
- Lotus® Domino®;
- IBM Directory.

WebSphere Application Server интегрируется с RACF и WebSEAL Secure Proxy (Trusted Association Interceptor), обеспечивая унифицированную, основанную на политиках и разрешениях модель защиты всех веб-ресурсов и EJB-компонентов, определенных в спецификации J2EE.

14.6.3 Непрерывная доступность

WebSphere для z/OS использует внутренние возможности обнаружения и исправления ошибок платформы zSeries. WebSphere для z/OS содержит средства управления завершением и восстановлением, осуществляющие обнаружение, изоляцию, исправление и восстановление после программных ошибок. WebSphere для z/OS может проводить дифференцирование и назначение приоритетов задач на основе соглашений об уровне сервиса. Продукт обеспечивает возможности кластеризации, а также возможности внесения изменений в программные компоненты, такие как менеджеры ресурсов, без нарушения работы.

В критически важном приложении, WebSphere для z/OS может использовать средство управления отказами z/OS, называемое менеджером автоматического перезапуска

(automatic restart manager, ARM). Это средство может обнаруживать отказы приложений и перезапускать серверы при возникновении отказов. WebSphere использует ARM для восстановления серверов приложений (служебных регионов). Каждый сервер приложений, запущенный в системе z/OS, регистрируется в группе перезапуска ARM.

WebSphere для z/OS может реализовать функцию, называемую *кластеризацией* (*clustering*). Технологии кластеризации широко используется в системах высокой доступности, включающих WebSphere, как показано на рис. 14.2.

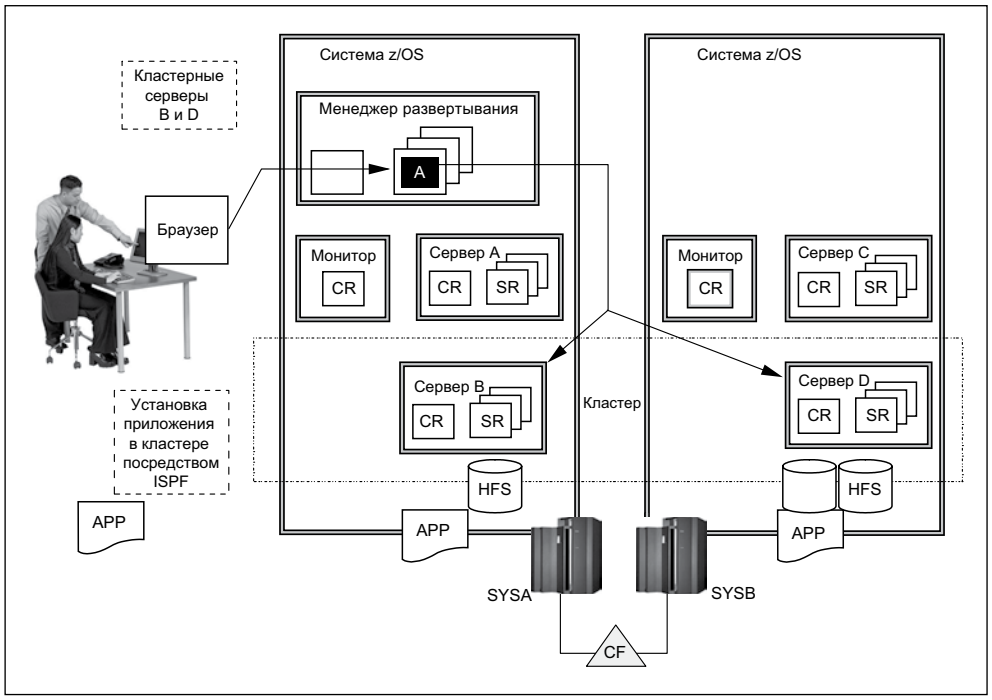


Рис. 14.2. Кластеризация серверов в ячейке

Кластер содержит несколько копий одного и того же компонента; при этом ожидается, что по меньшей мере одна из этих копий будет доступна для обслуживания запроса. В общем, кластер работает как одно целое, обеспечивая некоторое взаимодействие между отдельными копиями, обеспечивая направление запроса копии, способной обслужить запрос.

Проектировщики системы высокой доступности участвуют в определении уровня сервиса, так как они определяют количество и размещение отдельных участников кластеров. WebSphere для z/OS осуществляет управление некоторыми кластерами, необходимое для обеспечения требуемого уровня сервиса. Более высокого уровня доступности можно достичь путем объединения кластеров WebSphere с другими кластерными технологиями.

Кластер WebSphere Application Server состоит из отдельных участников кластера, причем каждый участник содержит одинаковый набор приложений. На входе класте-

ра WebSphere Application Server находится распределитель нагрузки, который направляет нагрузку отдельным участникам кластера.

Кластеры могут быть вертикальными и объединяться в одном LPAR (т. е. содержать два или больше участников, находящихся в системе z/OS) или же иметь горизонтальное размещение по нескольким LPAR, что позволяет достичь наивысшей доступности в случае отказа LPAR, содержащего участника.

Рабочую нагрузку в этом случае можно перенести на остальных участников кластера. Кроме того, в этих двух конфигурациях можно использовать гибрид, при котором кластер состоит из вертикальных и горизонтальных участников (рис. 14.3).

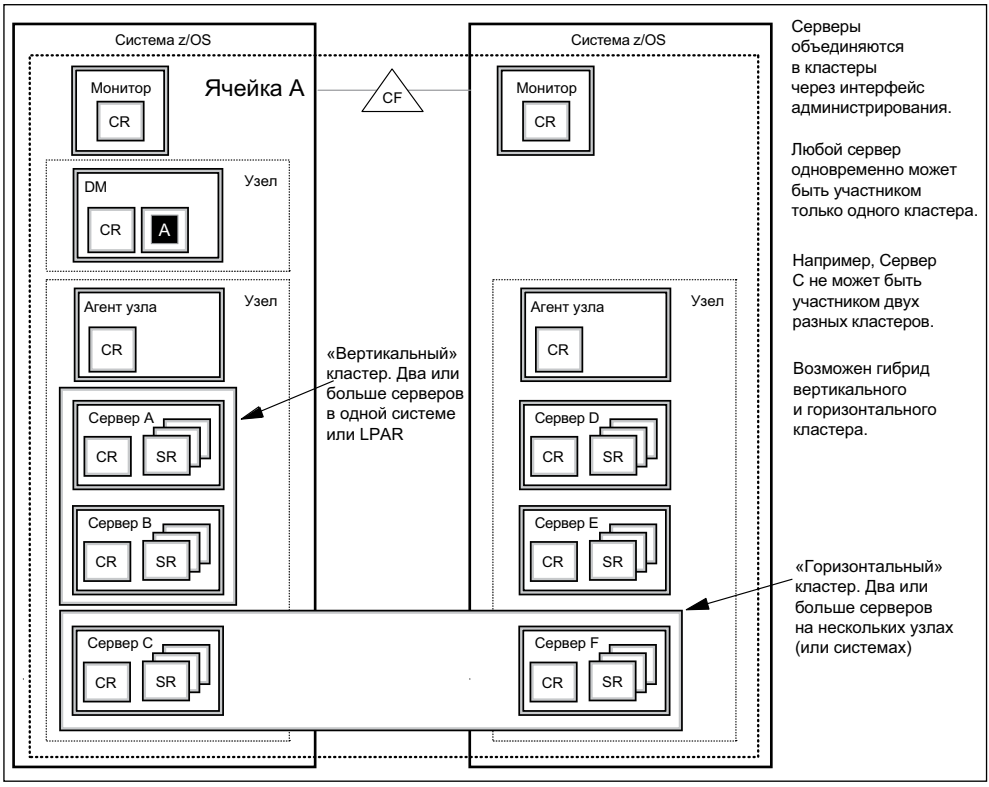


Рис. 14.3. Вертикальные и горизонтальные кластеры

Вы можете спросить, когда следует использовать вертикальные кластеры, а когда горизонтальные. Вертикальные кластеры можно использовать для проверки эффективности диспетчеризации в одной системе. В вертикальном кластере серверы соревнуются друг с другом за доступ к ресурсам.

14.6.4 Производительность

Производительность в значительной степени связана со структурой и кодом приложения вне зависимости от мощности платформы выполнения – неэффективное приложение будет выполняться в z/OS так же плохо, как и на любой другой платформе.

WebSphere Application Server для z/OS использует возможности оборудования мэйнфреймов и программного обеспечения, включающие схемы управления рабочей нагрузкой, динамическое конфигурирование LPAR и функции Parallel Sysplex. В частности, используется три различные функции управления нагрузкой в z/OS.

- Маршрутизация.
  Службы маршрутизации WLM используются для направления клиентов на серверы, размещенные на определенной системе, на основании измерения показателя текущего использования систем, называемого индексом производительности (Performance Index, PI).
- Организация очередей.
  Служба организации очередей WLM используется для диспетчеризации рабочих запросов с управляющего региона на один или несколько серверных регионов. Менеджер работ (Work Manager) может зарегистрироваться в WLM как менеджер очередей (Queuing Manager). Это сообщает WLM о том, что данному серверу требуется использовать очереди, управляемые WLM, для направления работы на другие серверы, что позволяет WLM управлять серверными пространствами для достижения целей производительности, установленных для работы.
- Назначение приоритетов.
  Сервер приложений предусматривает запуск и остановку серверных регионов для назначения приоритета работы. Это позволяет WLM осуществлять управление экземплярами серверов приложений для достижения целей, установленных предприятием.
  WLM рассчитывает индекс производительности (PI) для каждого периода класса обслуживания, что позволяет определить отклонения действительной производительности от целевой. Так как существует несколько типов целей, WLM нужен некоторый способ определения того, насколько хорошо или плохо работа в одном классе обслуживания выполняется по сравнению с другой работой. Класс обслуживания (service class, SC) используется для описания группы работ в рабочей нагрузке с аналогичными характеристиками производительности.

14.7 Конфигурация сервера приложений в z/OS

Конфигурация сервера приложений в z/OS включает следующее:

- узел базового сервера;
- менеджер сетевого развертывания.

14.7.1 Узел базового сервера

Узел базового сервера приложений является простейшей операционной структурой в WebSphere Application Server для z/OS. Он содержит сервер приложений и сервер монитора (один узел и одну ячейку), как показано на рис. 14.4. Все файлы конфигурации и определения хранятся в структуре директорий HFS, созданной для этого базового сервера приложений. Сервер монитора представляет собой специальный сервер

с одним управляющим регионом. Архитектура системы WebSphere для z/OS использует один сервер монитора на одну ячейку на одну систему или LPAR.

Каждый базовый сервер приложений содержит средства администрирования домена своей ячейки и отдельное хранилище для своей конфигурации. Поэтому можно использовать множество базовых серверов приложений, каждый из которых изолируется от других, используя собственную политику администрирования для специальных бизнес-требований.

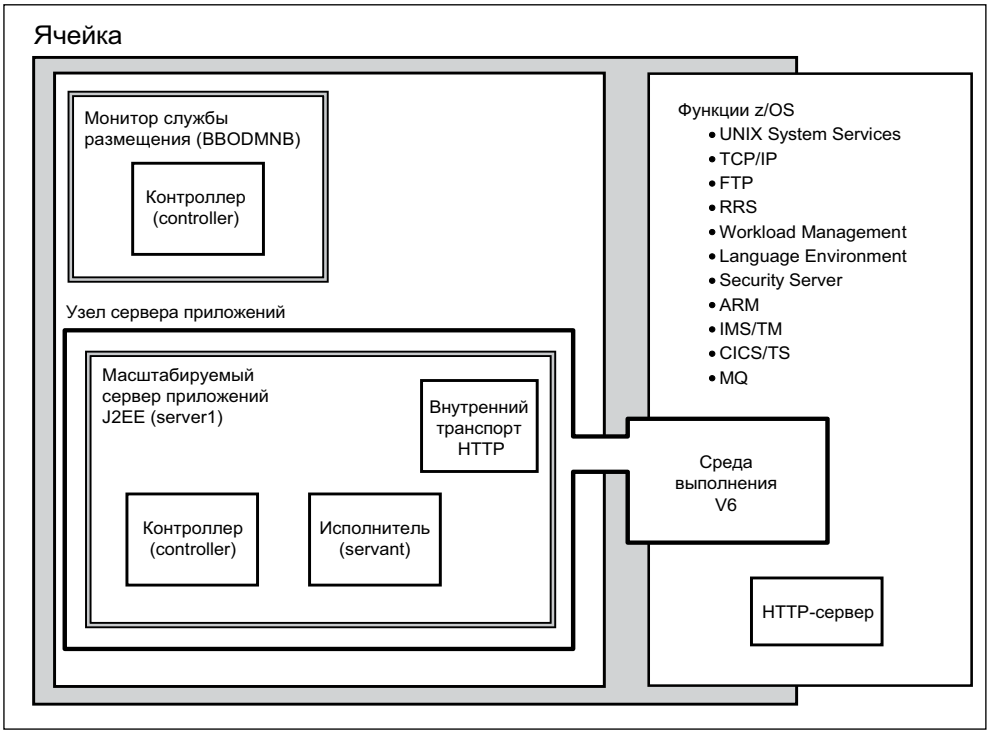


Рис. 14.4. Узел базового сервера

14.7.2 Менеджер сетевого развертывания

Менеджер сетевого развертывания (Network Deployment Manager, см. рис. 14.5) представляет собой расширение базового сервера приложений. Он позволяет системе осуществлять администрирование нескольких серверов приложений с центрального узла. В данном случае серверы приложений привязаны к узлам, и несколько узлов относятся к ячейке. Используя менеджер развертывания, можно без труда администрировать горизонтально и вертикально масштабируемые системы, а также распределенные приложения.

Менеджер сетевого развертывания также осуществляет управление хранилищами на каждом узле, выполняя такие задачи, как создание, обслуживание и удаление хранилищ. Система использует метод извлечения/изменения для обновления конфигурации.

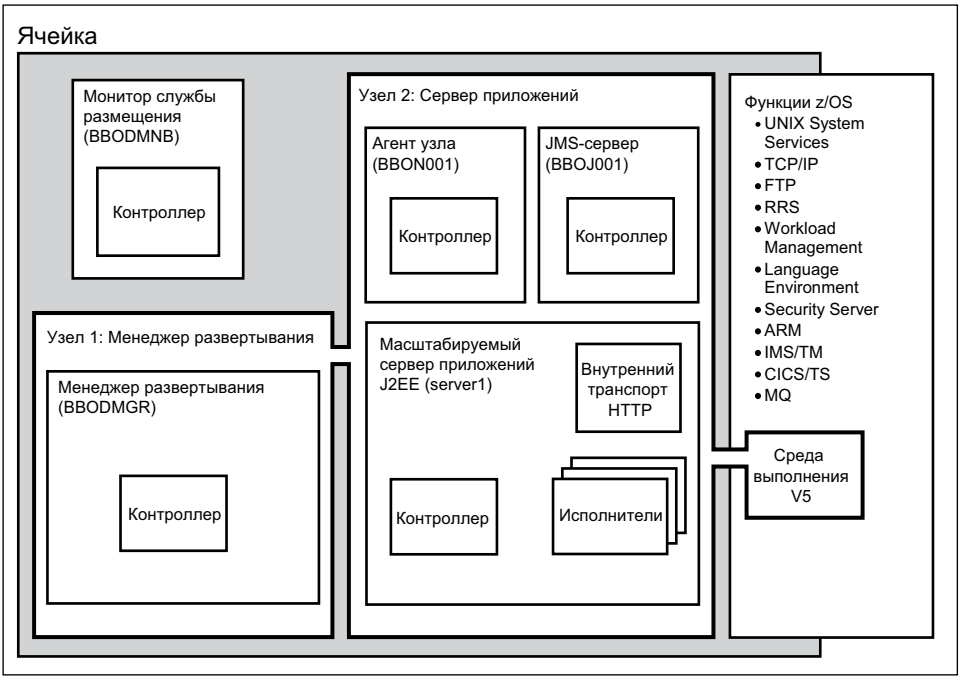


Рис. 14.5. Менеджер сетевого развертывания

14.8 Коннекторы для корпоративных информационных систем

Способность приложений взаимодействовать с ресурсами, находящимися вне процесса сервера приложений, и эффективно использовать эти ресурсы всегда являлась важным требованием. Также важна и способность изготовителей встраивать собственные системы для подключения и использования их ресурсов.

Приложение может требовать доступа к различным типам ресурсов, которые могут расположены на одном компьютере с приложением, а могут быть расположены на другой системе. Таким образом, доступ к ресурсу начинается с подключения, представляющего собой путь от приложения к ресурсу, который может быть другим менеджером транзакций или менеджером баз данных.

Доступ Java-программы к широкому набору ресурсов заднего плана выполняется через адаптер ресурсов. Он представляет собой программный драйвер системного уровня, встраиваемый в сервер приложений и позволяющий Java-приложению подключаться к различным ресурсам заднего плана.

Следующие аспекты являются общими для всех подключений:

- Создание подключения может быть дорогостоящим. Настройка подключения может занять длительное время по сравнению с количеством времени, в течение которого подключение действительно используется.

- Подключения должны быть защищенными. Часто имеет место объединение усилий между приложением и сервером, работающим с ресурсом.
- Подключения должны функционировать должным образом. Их производительность может быть критически важной для успешного выполнения приложения, и она является функцией общей производительности приложения.
- Необходимо выполнять мониторинг и диагностику подключений. Качество диагностики подключения зависит от информации о состоянии сервера и ресурса.
- Методы подключения и работы с ресурсом. Различные архитектуры баз данных требуют использования разных средств для доступа с сервера приложений.
- Качество сервиса, которое становится важным фактором при доступе к ресурсам, расположенным за пределами сервера приложений. Приложение может требовать свойств ACID (Atomicity, Consistency, Isolation, Durability – атомарность, согласованность, изоляция, долговечность), которых можно достичь при использовании данных под управлением транзакций.

Ресурсы предприятия часто представляют собой старые ресурсы, разрабатывавшиеся предприятием в течение длительного времени и являющиеся внешними по отношению к процессу сервера приложений. Каждый тип ресурса имеет собственный протокол подключения и индивидуальный набор интерфейсов доступа к ресурсу. Поэтому для того, чтобы ресурс был доступен из процесса JVM как содержащийся на сервере приложений его необходимо адаптировать.

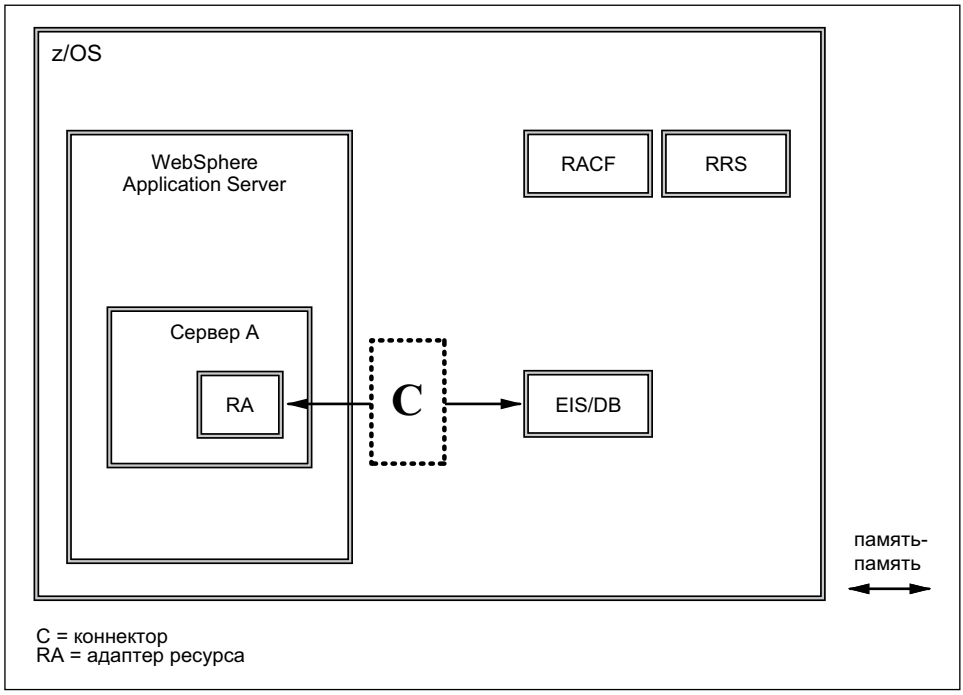


Рис. 14.6. Базовая архитектура взаимодействия между коннектором и EIS

WebSphere Application Server содержит средства связи с другими подсистемами z/OS, такими как CICS, DB2 и IMS. Связь осуществляется через адаптер ресурса и коннектор. Доступ к корпоративным информационным системам (Enterprise Information Systems, EIS) заднего плана расширяет функциональные возможности сервера приложений на существующие функции предприятия, обеспечивая расширенные возможности.

J2EE Connector Architecture (JCA) определяет соглашения между приложением, коннектором и сервером приложений, на котором выполняется приложение. Приложение содержит компонент, называемый *адаптером ресурса* (*resource adapter*). Он находится в коде приложения, обеспечивающем интерфейс с коннектором и создаваемым разработчиком приложения.

С точки зрения программирования это означает, что программисты могут использовать единый унифицированный интерфейс для получения данных из EIS. Адаптер ресурса отсортировывает различные элементы и обеспечивает модель программирования, не зависящую от фактического выполнения EIS и требований к связи.

14.8.1 Коннекторы в z/OS

WebSphere для z/OS содержит следующие *коннекторы*, позволяющие веб-приложениям в z/OS взаимодействовать с программными продуктами промежуточного уровня для мэйнфреймов, такими как CICS, IMS и DB2:

- CICS Transaction Gateway;
- IMS Connect;
- DB2 JDBC.

CICS Transaction Gateway

Система Customer Information Control System (CICS) содержит шлюз CICS Transaction Gateway (CTG) для подключения сервера приложений к CICS. CTG обеспечивает интерфейс между Java и транзакциям CICS-приложений. Он представляет собой набор клиентских и серверных программных компонентов, включающих службы и средства доступа к CICS с сервера приложений. CTG использует специальные API и протоколы в сервлетах или EJB для запрашивания служб и функций менеджера транзакций CICS.

IMS Connect

IMS Connect представляет собой TCP/IP-сервер коннектора, позволяющий клиенту сервера приложений обмениваться сообщениями с IMS Open Transaction Manager Access (OTMA). Этот сервер осуществляет связь между TCP/IP-клиентами и базами данных IMS. Он поддерживает множество TCP/IP-клиентов, осуществляющих доступ ко множеству баз данных. Для защиты информации, передаваемой через TCP/IP, IMS Connect обеспечивает поддержку протокола SSL (Secure Sockets Layer).

IMS Connect также может выполнять функции маршрутизатора между клиентами сервера приложений и локальным клиентами с базами данных и ресурсами IMSplex. Сообщения запросов, получаемые от TCP/IP-клиентов с использованием TCP/IP-подключений или локальных клиентов, использующих z/OS Program Call (PC), передают-

ся в базу данных посредством сеансов межсистемного средства сопряжения (cross-system coupling facility, XCF). IMS Connect получает сообщения ответов из базы данных, после чего передает их обратно запросившим TCP/IP-клиентам или локальным клиентам.

IMS Connect поддерживает TCP/IP-клиентов, связывающихся посредством сокетных вызовов, однако также реализована поддержка любого TCP/IP-клиента, связывающегося, используя другой формат потока входных данных. Программы пользовательских «выходов» сообщений (message exits) могут выполняться в адресном пространстве IMS Connect для преобразования формата сообщений инсталляции z/OS в формат сообщений OTMA, прежде чем IMS Connect отправит сообщение IMS. Пользовательские «выходы» сообщений также осуществляют преобразование формата сообщений OTMA в формат сообщений инсталляции, прежде чем отправить сообщение обратно в IMS Connect. Затем IMS Connect отправляет выходные данные клиенту.

DB2 JDBC

Java Database Connectivity (JDBC) представляет собой интерфейс программирования приложений (API), используемый языком программирования Java для доступа к различным формам табличных данных, равно как и к некоторым иерархическим системам, таким как IMS. Спецификации JDBC были разработаны компанией Sun Microsystems совместно с поставщиками реляционных баз данных, такими как Oracle и IBM, чтобы обеспечить переносимость Java-приложений между платформами баз данных.

Этот интерфейс не обязательно попадает в категорию «коннекторов», так как для его реализации не требуется отдельного адресного пространства. Интерфейс представляет собой Java-конструкцию, подобную Java-классу, но не обеспечивающую реализацию своих методов. В JDBC действительная реализация JDBC-интерфейса обеспечивается изготовителем базы данных и называется «драйвером». Это обеспечивает переносимость, так как все операции доступа к JDBC выполняются посредством стандартных вызовов со стандартными параметрами. Таким образом, можно написать приложение, практически не зависящее от используемой базы данных, так как весь код, зависящий от платформы, хранится в JDBC-драйверах.

В итоге JDBC должен быть гибким в отношении выполняемых и невыполняемых им функций, основываясь только на том факте, что различные системы управления базами данных имеют различный уровень функциональности. JDBC-драйверы обеспечивают физический код, реализующий объекты, методы и типы данных, определенные в спецификации. Стандарты JDBC определяют четыре типа драйверов с номерами от 1 до 4. Различие между ними основано на физической реализации драйвера и способе его связи с базой данных.

z/OS поддерживает только драйверы Type 2 и Type 4.

- Type 2.

  JDBC API вызывает код, зависящий от платформы и базы данных, для доступа к базе данных. Этот тип драйвера является наиболее используемым и обеспечивает наивысшую производительность. Однако так как код драйвера зависит от платформы, для каждой платформы необходимо создавать отдельную версию (этим должен заниматься изготовитель базы данных).

- Type 4.

  Драйвер Type 4 полностью написан на Java и осуществляет доступ к целевой базе данных напрямую, используя протокол самой базы данных (в случае DB2 используется протокол DRDA). Так как драйвер полностью написан на Java, его можно перенести на любую платформу, поддерживающую данный протокол DBMS без изменений, что позволяет приложениям использовать его на различных платформах без изменений.

Java-приложение, выполняющееся под управлением WebSphere Application Server, взаимодействует с драйвером (Universal) Type 4 JDBC, поддерживающим двухфазовую фиксацию, и драйвер взаимодействует напрямую с удаленным сервером базы данных через DRDA. Драйвер Universal Type 4 реализует функциональные возможности DRDA Application Requester.

Для доступа к DB2 в z/OS, IBM предоставляет драйвер Type 2 и драйвер, сочетающий реализации JDBC Type 2 и Type 4. В общем, связь JDBC Type 2 используется в Java-программах, выполняющихся в той же системе z/OS с целевой подсистемой DB2. Связь JDBC Type 4 используется в Java-программах, выполняющихся в системе z/OS, отличной от той, где находится целевая подсистема DB2.

Основные термины в этой главе

Ячейка	CS	CGI
EIS	JMX™	J2EE
SR	Кластер	Узел

14.9 Контрольные вопросы

Чтобы проверить понимание материала этой главы, ответьте на следующие вопросы:

1. Перечислите шесть свойств модели приложений J2EE.
2. Перечислите три причины выполнения WebSphere Application Server в z/OS.
3. Назовите три коннектора.
4. Какое основное различие между HTTP Server и WebSphere Application Server для z/OS?
5. Когда следует использовать коннекторы?

15

Обмен сообщениями и управление очередями

Цель. Как специалисту в сфере мэйнфреймов, вам нужно понимать такие понятия, как обмен сообщениями и управление очередями. Эти функции нужны для связи между разнородными приложениями и платформами.

После завершения работы над этой главой вы сможете:

- объяснить, почему используется обмен сообщениями и управление очередями;
- описать асинхронный поток сообщений;
- описать функции менеджера очередей;
- перечислить три адаптера, связанных с z/OS.

15.1 Что такое WebSphere MQ

Большинство крупных организаций в настоящее время имеют наследие из информационных систем разных производителей, что часто затрудняет совместный доступ к средствам связи и данным между системами. Многим из этих организаций требуется также осуществлять электронную передачу и совместное использование данных с поставщиками и клиентами, которые могут использовать другие разнородные системы. Было бы удобно иметь инструмент обработки сообщений, который получал бы сообщения от системы одного типа и передавал бы их в систему другого типа.

IBM WebSphere MQ упрощает интеграцию приложений, передавая сообщения между приложениями и веб-службами. Этот продукт используется более чем на 35 аппаратных платформах, а также для обмена сообщениями в системе «точка-точка» из приложений на языках Java, C, C++ и COBOL. Три четверти предприятий, приобретающих системы обмена сообщениями между приложениями, выбирают WebSphere MQ. В самой крупной инсталляции передается более 250 миллионов сообщений ежедневно.

Существует всего несколько протоколов, способных согласовывать обновления, удаления и синхронизацию данных, хранящихся в различных базах данных на разных системах. В смешанных средах трудно обеспечить упорядоченность; для их интеграции часто требуются сложные программы.

Очереди сообщений и управляющее ими программное обеспечение, такое как IBM WebSphere MQ для z/OS, позволяют осуществлять межпрограммную связь. В контексте оперативных приложений *обмен сообщениями* (*messaging*) и *управление очередями* (*queuing*) можно понять следующим образом:

- Обмен сообщений подразумевает, что программы общаются посредством передачи друг другу сообщений (данных), а не посредством прямого обращения друг к другу.
- Управление очередями подразумевает, что сообщения помещаются в очереди, расположенные в памяти, так что программы могут выполняться независимо друг от друга, с различными показателями скорости и времени, имея разное расположение и без логической связи между ними.

15.2 Синхронная связь

На рис. 16.1 показан базовый механизм межпрограммной связи с использованием модели синхронной связи.

Программа A готовит сообщение и помещает его в Очередь 1. Программа B получает сообщение из Очереди 1 и обрабатывает его. И Программа A, и Программа B используют интерфейс программирования приложений (API) для помещения сообщений в очередь и получения сообщений из очереди. WebSphere MQ API называется интерфейсом очередей сообщений (Message Queue Interface, MQI).

Когда Программа A помещает сообщение в Очередь 1, Программа B может быть еще не запущена. Очередь надежно хранит сообщение до тех пор, пока Программа B не запустится и не будет готова получить сообщение. Подобным образом в то время,

когда Программа B получит сообщение из Очереди 1, Программа A может уже не работать. При использовании этой модели не требуется, чтобы две программы, общающихся друг с другом, выполнялись одновременно.

Что касается времени ожидания перед продолжением обработки Программой A, это является вопросом проектирования. Такая структура может быть желательной в некоторых ситуациях, однако слишком долгое ожидание в любом случае является нежелательным. Для разрешения такой ситуации разработана асинхронная связь.

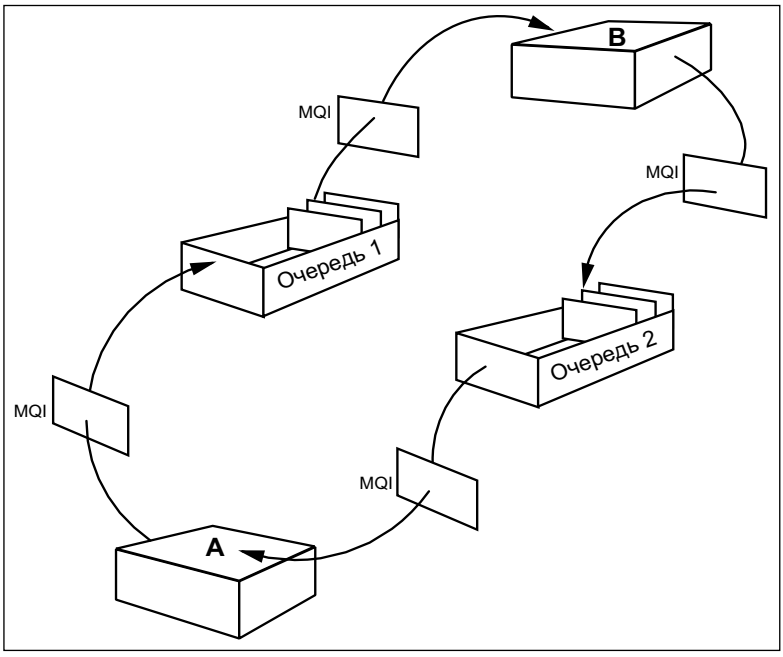


Рис. 15.1. Модель синхронного приложения

15.3 Асинхронная связь

При использовании асинхронной модели Программа A помещает сообщения в Очередь 1 для обработки Программой B, тогда как получением и обработкой ответов из Очереди 2 занимается Программа C, работающая асинхронно относительно Программы A. Обычно Программа A и Программа C являются частью одного приложения. Алгоритм работы представлен на рис. 15.2.

Асинхронная модель является естественной для WebSphere MQ. Программа A может продолжать помещать сообщения в Очередь 1 и не блокируется необходимостью ожидать ответа на каждое сообщение. Она может продолжать помещать сообщения в Очередь 1 даже при отказе Программы B. В этом случае Очередь 1 надежно сохранит сообщения до перезапуска Программы B.

В одной из разновидностей асинхронной модели Программа A может поместить последовательность сообщений в Очередь 1, затем может продолжить выполнение

других задач, после чего возвратиться для получения и обработки ответов. Свойство WebSphere MQ, состоящее в том, что взаимодействующие приложения не должны быть активными одновременно, называется *независимостью от времени* (*time independence*).

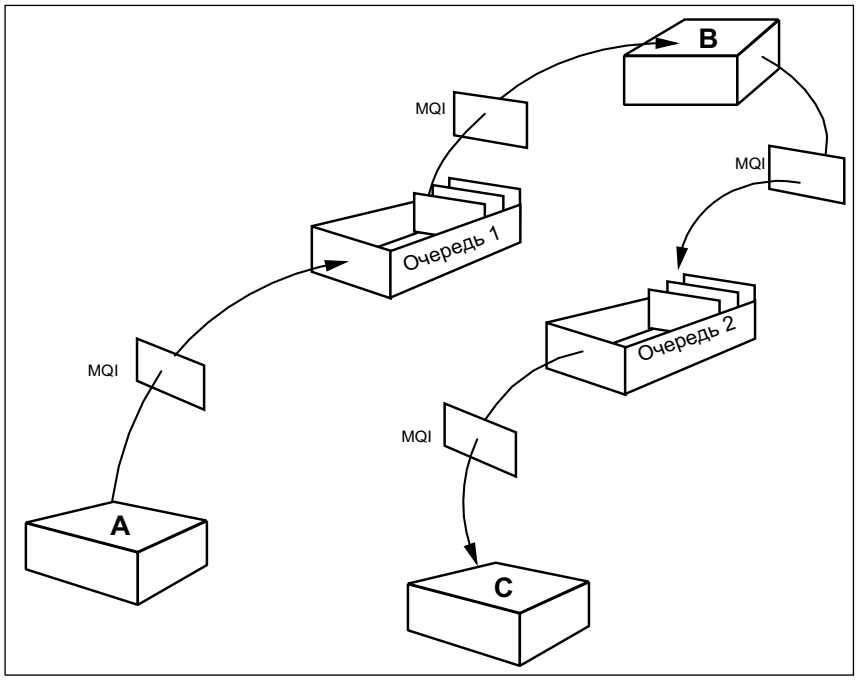


Рис. 15.2. Модель асинхронного приложения

15.4 Типы сообщений

WebSphere MQ использует четыре типа сообщений:

Датаграмма	Сообщение, для которого не ожидается ответ.
Запрос	Сообщение, для которого запрашивается ответ.
Ответ	Ответ на сообщение-запрос.
Отчет	Сообщение, описывающее событие, такое как возникновение ошибки или подтверждение получения или доставки.

15.5 Очереди сообщений и менеджер очередей

Очереди сообщений используются для хранения сообщений, отправленных программами. Они определяются как объекты, принадлежащие менеджеру очередей.

Когда приложение помещает сообщение в очередь, менеджер очередей обеспечивает для этого сообщения:

- безопасное хранение;
- восстанавливаемость;
- однократную доставку целевому приложению.

Это истинно, даже если сообщение необходимо доставить в очередь, принадлежащую другому менеджеру очередей; это свойство WebSphere MQ называется гарантированной доставкой (assured delivery).

15.5.1 Менеджер очередей

Компонент программного обеспечения, владеющий и управляющий очередями, называется менеджером очередей (queue manager, QM). В WebSphere MQ менеджер очередей сообщений называется MQM; он обеспечивает обмен сообщениями приложений, размещение сообщений в соответствующих очередях, перенаправление сообщений другим менеджерам очередей и обработку сообщений через общий интерфейс программирования Message Queue Interface (MQI).

Менеджер очередей может сохранить сообщения для последующей обработки в случае отказов приложения или системы. Сообщения сохраняются в очереди до тех пор, пока через MQI не будет получен ответ об успешном завершении.

Между менеджерами очередей и менеджерами баз данных есть некоторые сходства. Менеджеры очередей владеют и управляют очередями подобно тому, как менеджеры баз данных владеют и управляют своими объектами хранения данных. Они обеспечивают программный интерфейс для доступа к данным, а также обеспечивают безопасность, авторизацию, восстановление и администрирование.

Однако есть и важные различия. Базы данных разработаны для долгосрочного хранения данных с использованием эффективных механизмов поиска, тогда как очереди не имеют такого предназначения. Сообщение в очереди обычно указывает на незавершенность бизнес-процесса; оно может представлять неисполненный запрос, необработанный ответ или непрочитанный отчет. На рис. 15.4 представлен алгоритм работы в менеджерах очередей и менеджерах баз данных.

15.5.2 Типы очередей сообщений

Существует несколько типов очередей сообщений. В этой книге чаще всего используются следующие:

- Локальная очередь.
  Очередь является локальной, если ею владеет менеджер очередей, с которым связано приложение. Она используется для хранения сообщений программ, использующих один и тот же менеджер очередей. Приложение необязательно должно выполняться на одном компьютере с менеджером очередей.
- Удаленная очередь.
  Очередь является удаленной, если ее владельцем является другой менеджер очередей. Удаленная очередь не является реальной очередью; это всего лишь *определение* удаленной очереди для локального менеджера очередей. Программы не могут считывать сообщения из удаленных очередей. Удаленные очереди связаны с транспортной очередью.

- Транспортная очередь.
  Эта локальная очередь имеет специальное назначение: она используется в качестве промежуточного этапа при отправлении сообщений в очереди, принадлежащие другому менеджеру очередей. Транспортные очереди являются прозрачными для приложения; другими словами, они используются очередью инициации менеджера очередей.
  Эта очередь представляет собой локальную очередь, в которую менеджер очередей записывает (невидимым для программиста способом) триггерное сообщение при возникновении определенных условий в другой локальной очереди, например, при помещении сообщения в пустую очередь сообщений или в транспортную очередь. Два приложения WebSphere MQ осуществляют мониторинг очередей инициации и чтение триггерных сообщений: триггерный монитор и инициатор каналов. *Триггерный монитор* (*trigger monitor*) может запускать приложения в зависимости от сообщения. *Инициатор* каналов (*channel initiator*) начинает передачу между менеджерами очередей.
- Очередь недоставленных сообщений.
  Менеджер очередей (QM) должен быть способен обрабатывать ситуации, когда он не может доставить сообщение, например:
  - если целевая очередь заполнена;
  - если целевая очередь не существует;
  - если запись сообщения в целевую очередь была запрещена;
  - если отправитель не имеет полномочий использования целевой очереди;
  - если сообщение имеет слишком большой размер;
  - если сообщение содержит повторяющийся последовательный номер сообщения.

  При возникновении какого-либо из этих условий сообщение записывается в очередь недоставленных сообщений (dead-letter queue). Эта очередь определяется при создании менеджера очередей, и каждый QM должен иметь такую очередь. Она используется в качестве хранилища для всех сообщений, которые невозможно доставить.

15.6 Что такое канал?

Канал (*channel*) представляет собой логическую линию связи. Диалоговый стиль связи между программами требует наличия коммуникационного соединения между каждой парой сообщающихся приложений. Каналы отделяют приложения от базовых коммуникационных протоколов.

WebSphere MQ использует два вида каналов:

- Канал сообщений.
  Канал сообщений соединяет два менеджера очередей посредством специальных канальных агентов (message channel agents, MCA). Канал сообщений является однонаправленным, содержит два канальных агента (отправитель и получатель) и коммуникационный протокол. MCA передает сообщения из транс-

портной очереди в линию связи и из линии связи в целевую очередь. Для двусторонней связи необходимо определить *пару* каналов, состоящих из отправителя и получателя.

- MQI-канал.
  MQI-канал соединяет клиента WebSphere MQ с менеджером очередей. Клиенты не имеют собственного менеджера очередей. MQI-канал является двунаправленным.

15.7 Как обеспечивается транзакционная целостность

Предприятию может требоваться синхронно обслуживать две или больше распределенных баз данных. WebSphere MQ предлагает систему, включающую несколько единиц работы (unit of work), действующих асинхронно, как показано на рис. 15.3.

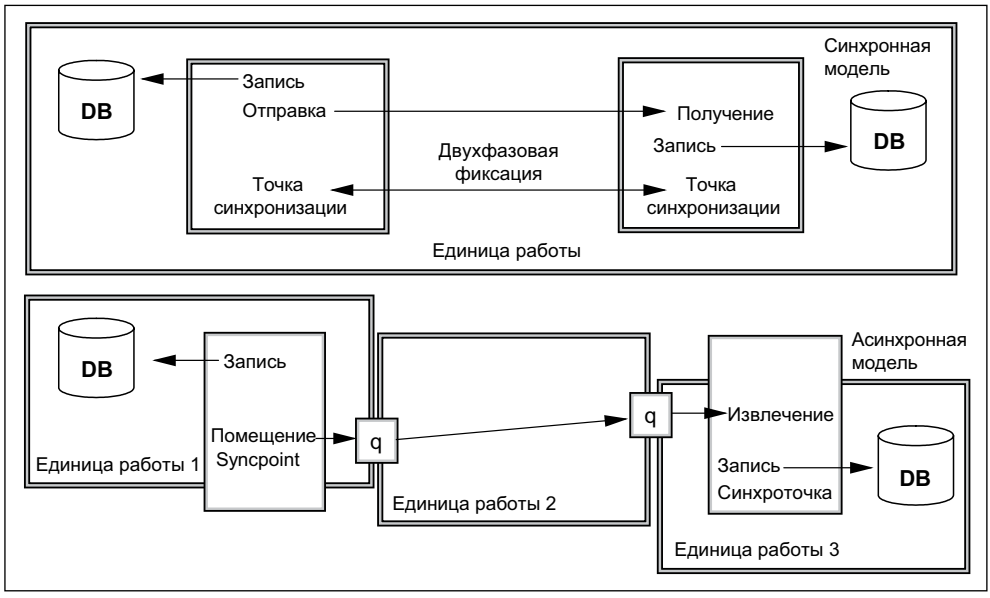


Рис. 15.3. Целостность данных

Верхняя половина рис. 15.3 изображает структуру двухфазовой фиксации транзакций (two-phase commit), тогда как система WebSphere MQ, представленная в нижней половине, работает следующим образом:

- Первое приложение осуществляет запись в базу данных, помещает сообщение в очередь и создает точку синхронизации для фиксации изменений в двух ресурсах. Сообщение содержит данные, используемые для обновления второй базы данных в отдельной системе. Так как очередь является удаленной, сообщение не идет дальше транспортной очереди в единице работы. При фиксации единицы работы сообщение становится доступным для извлечения отправляющим MCA.
- Во второй единице работы отправляющий MCA получает сообщение из транспортной очереди и отправляет его в получающий MCA в системе со второй базой данных, и получающий MCA помещает сообщение в целевую очередь.

Надежность выполнения этой последовательности действий обеспечивается свойством гарантированной доставки WebSphere MQ. После фиксации этой единицы работы сообщение становится доступным для извлечения вторым приложением.

- В третьей единице работы второе приложение получает сообщение из целевой очереди и обновляет базу данных, используя данные, содержащиеся в сообщении.

Целостность бизнес-транзакции обеспечивается транзакционной целостностью единиц работы 1 и 3 и свойством однократной гарантированной доставки WebSphere MQ, реализованным в единице работы 2.

Если бизнес-транзакция является более сложной, может быть вовлечено множество единиц работы.

15.8 Пример обмена сообщениями и управления очередями

Теперь вернемся к примеру туристического агентства, чтобы увидеть, каким образом средства обмена сообщениями участвуют в размещении заказа на тур. Предположим, что туристический агент должен зарезервировать авиабилет, комнату в гостинице и прокатный автомобиль. Все эти операции резервирования должны быть выполнены прежде, чем основную бизнес-транзакцию можно будет считать завершенной (рис. 15.4).

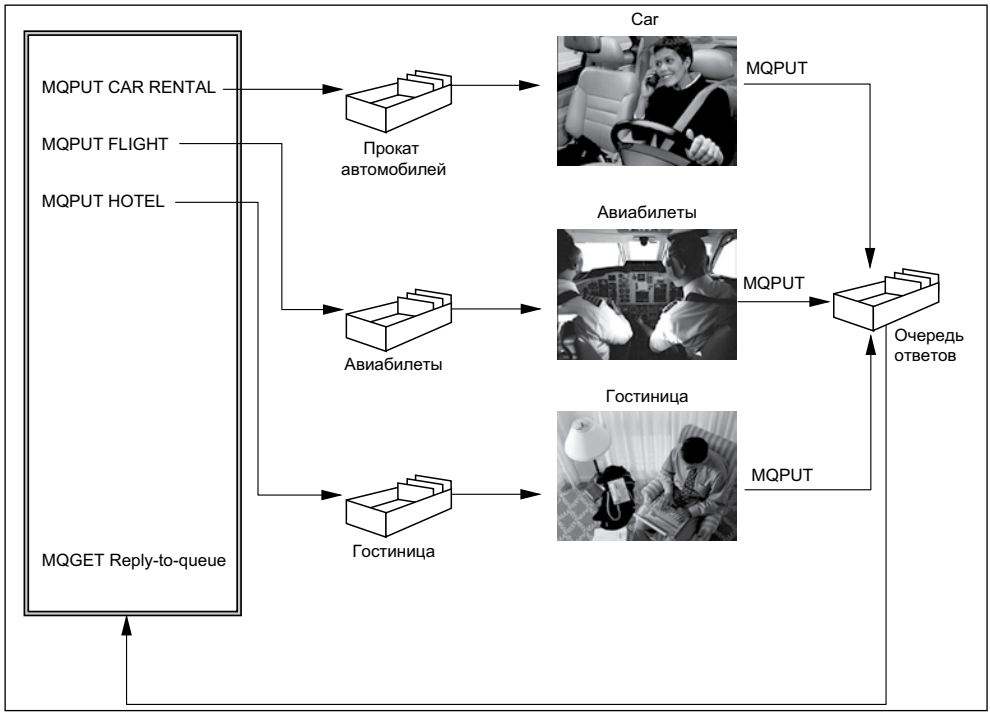


Рис. 15.4. Параллельная обработка

Используя менеджер очередей сообщений, такой как WebSphere MQ, приложение может отправить несколько запросов одновременно; ему не требуется ожидать ответа на один запрос перед отправлением следующего. Сообщение помещается в каждую из трех очередей, обслуживающих приложение бронирования авиабилетов, приложение бронирования гостиниц и приложение проката автомобилей. Каждое приложение может выполнить соответствующую задачу параллельно другим двум и поместить ответное сообщение в очередь ответов. Приложение агентства ожидает эти ответы и генерирует консолидированный ответ для туристического агента.

Проектирование системы таким образом может улучшить общее время реагирования. Хотя приложение может обрабатывать ответы только после получения всех ответов, логика программы может также определить, что делать, если за заданный период времени получен только частичный набор ответов.

15.9 Взаимодействие с CICS, IMS, пакетными заданиями или TSO/E

Существуют версии WebSphere MQ для различных платформ. WebSphere MQ для z/OS включает несколько адаптеров, обеспечивающих поддержку обмена сообщениями и управления очередями для:

- CICS – мост WebSphere MQ-CICS;
- IMS – мост WebSphere MQ-IMS;
- пакетные задания или TSO/E.

15.10 Заключение

В среде оперативных приложений, средства обмена сообщениями и управления очередями обеспечивают связь между приложениями на различных платформах. IBM WebSphere MQ для z/OS представляет собой пример программного продукта, управляющего обменом сообщениями и управлением очередями на мэйнфреймах и в других средах. Используя средства обмена сообщениями, программы взаимодействуют посредством сообщений, а не посредством прямого обращения друг к другу. Используя средства управления очередями, сообщения сохраняются в очередях, что позволяет программам выполняться независимо друг от друга (асинхронно).

Ниже перечислены некоторые функциональные преимущества WebSphere MQ.

1. Общий интерфейс программирования приложений (MQI), согласованный на всех поддерживаемых платформах.
2. Передача данных с гарантированной доставкой. Сообщения не исчезают даже при отказе системы. Не выполняется и многократная доставка сообщений.
3. Асинхронная связь. Другими словами, сообщающимся приложениям не обязательно быть активными одновременно.
4. Обработка с управлением сообщениями как стиль структуры приложения. Приложение разделяется на отдельные функциональные модули, которые могут выполняться на различных системах, в разное время или параллельно.

5. Программирование приложений выполняется быстрее, когда программисту не приходится иметь дело с трудностями сетевой связи.

Основные термины в этой главе

Локальная очередь	Канал	С управлением сообщениями
MQI	Асинхронное приложение	Очередь недоставленных сообщений
QM	Удаленная очередь	Точка синхронизации

15.11 Контрольные вопросы

Чтобы проверить понимание материала этой главы, ответьте на следующие вопросы:

1. Почему обмен сообщениями и управление очередями необходимо для связи между разнородными приложениями и платформами?
2. Опишите асинхронный поток сообщений.
3. Опишите функции менеджера очередей.
4. Перечислите три адаптера, связанных с z/OS.
5. Каково назначение MQI?
6. Для чего используется очередь недоставленных сообщений?

Часть 4

Системное программирование в z/OS

В этой части мы рассмотрим внутреннее устройство z/OS и обсудим системные библиотеки, безопасность и процедуры от загрузки системы (IPL) до остановки системы z/OS. Эта часть также включает главы, посвященные сведениям об аппаратном обеспечении и виртуализации, а также о кластеризации нескольких систем z/OS в *сисплексе*.

16

Обзор системного программирования

Цель. Введение в основные принципы функционирования z/OS.
После прочтения данной главы вы сможете:

- описать обязанности системного программиста z/OS;
- описать системные библиотеки, их использование и методы управления их содержимым;
- перечислить различные типы консолей операторов;
- описать процесс начальной загрузки системы.

16.1 Роль системного программиста

Системный программист отвечает за управление аппаратной конфигурацией мэйнфрейма и за установку, настройку и обслуживание операционной системы мэйнфрейма. В инсталляции важно обеспечить доступность системы и ее служб, а также соответствие соглашениям об уровне сервиса. Инсталляции, работающие 24 часа в день, 7 дней в неделю должны обеспечивать минимальные нарушения работы.

В этой главе рассматриваются некоторые области, представляющие интерес для потенциальных системных программистов z/OS. Хотя эта книга не охватывает все аспекты системного программирования, важно понимать, что работа системного программиста z/OS очень сложна и требует навыков во многих аспектах системы, таких как:

- конфигурации устройств ввода-вывода;
- конфигурации процессоров;
- определения консолей;
- системные библиотеки, содержащие программное обеспечение;
- системные наборы данных и их размещение;
- параметры настройки, используемые для определения конфигурации z/OS;
- администрирование безопасности.

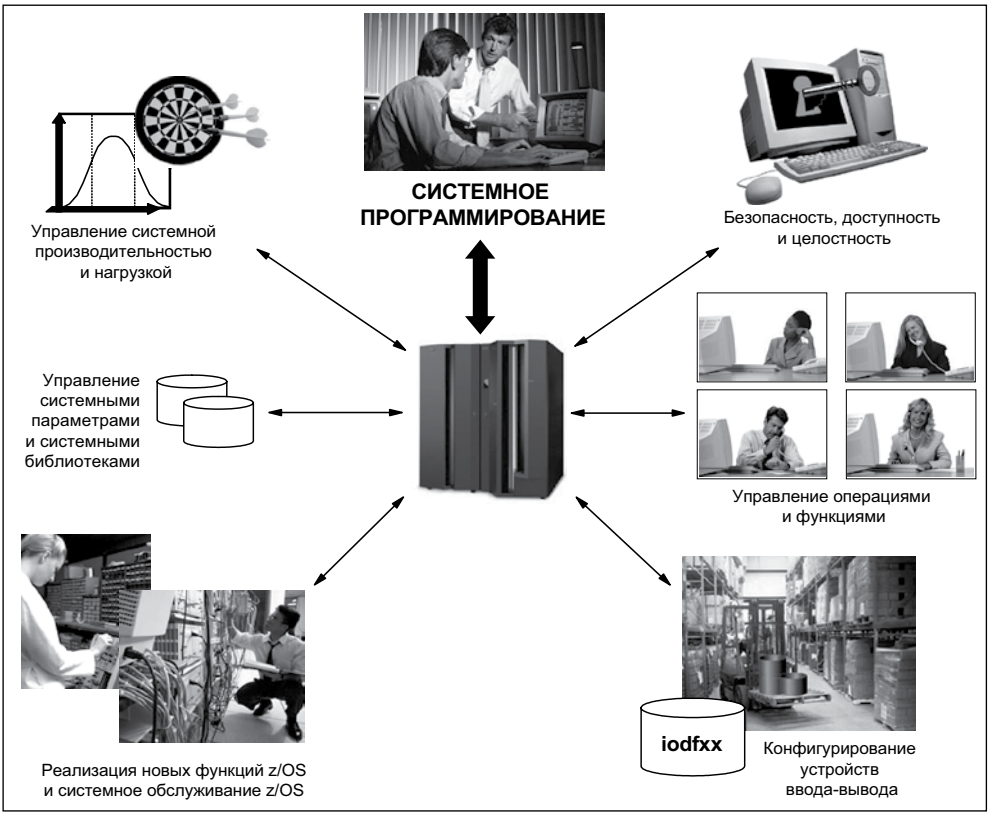


Рис. 16.1. Некоторые сферы, в которые вовлечен системный программист

Как показано на рис. 16.1, роль системного программиста обычно предполагает некоторое вовлечение в следующие аспекты обслуживания системы:

- Настройка системы.
- Управление системной производительностью.
- Конфигурирование устройств ввода-вывода.
- Следование процессу управления изменениями.
- Конфигурирование консолей.
- Инициализация системы.

16.2 Что подразумевается под разделением обязанностей

В крупной инсталляции z/OS обычно имеет место «разделение обязанностей» как между сотрудниками отдела системного программирования, так и между отделом системного программирования и другими отделами ИТ-организации.

Типичная инсталляция z/OS предполагает наличие следующих ролей:

- системный программист z/OS;
- системный программист CICS;
- системный программист баз данных;
- администратор базы данных;
- системный программист сетевых технологий;
- специалист по автоматизации;
- специалист по управлению безопасностью;
- специалист по управлению оборудованием;
- аналитик производственного контроля;
- системный оператор;
- оператор сети;
- администратор безопасности;
- специалист по управлению обслуживанием.

Разделение отчасти является требованием аудита; оно не допускает наличие у одного человека слишком высоких полномочий в системе.

Например, при добавлении нового приложения в систему необходимо выполнить множество задач, прежде чем приложение смогут использовать конечные пользователи. Аналитик производственного контроля нужен для добавления пакетных приложений в группу запланированных пакетных задаиий, добавления новых процедур в библиотеку процедур и настройки операционных процедур. Системный программист нужен для выполнения задач, связанных с самой системой, таких как настройка привилегий безопасности и добавление программ в системные библиотеки. Программист также участвует в настройке автоматизации новых приложений.

Однако в тестовой системе одному человеку может быть необходимо выполнять все роли, включая роль оператора, и это часто является наилучшим способом понять, как все работает.

16.3 Настройка системы

В этом разделе рассматриваются следующие темы:

- системные библиотеки, содержащие программное обеспечение
- системные наборы данных и их размещение
- конфигурация устройств ввода-вывода
- определения консолей
- параметры настройки, используемые для определения конфигурации z/OS
- реализация и обслуживание z/OS

16.3.1 Системные библиотеки z/OS

Как показано на рис. 16.2, в системе существуют различные типы данных.

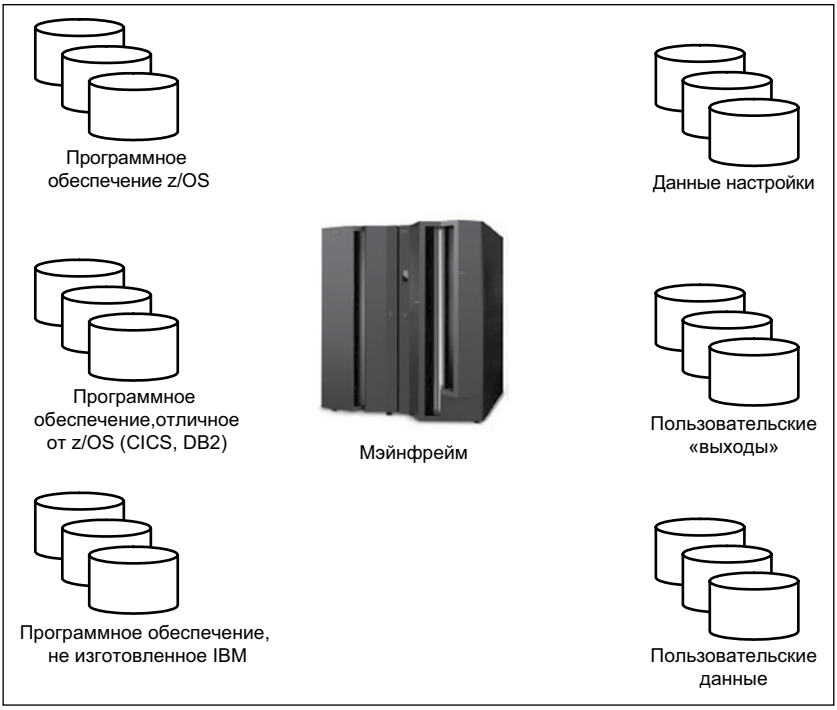


Рис. 16.2. Типы данных

Во-первых, системы содержат программное обеспечение z/OS, поставляемое IBM. Оно обычно устанавливается на группу дисковых томов, называемых резидентными системными томами (system residence volumes, SYSRES).

Гибкость z/OS в основном обеспечивается благодаря использованию этих SYSRES-групп. Они позволяют осуществлять обслуживание новой группы, клонированной из рабочей группы, тогда как текущая группа обрабатывает рабочую нагрузку.

После этого допускается короткое отключение для начальной загрузки с новой группы, что означает завершение обслуживания! Кроме того, изменения можно отменить путем начальной загрузки со старой группы.

Управление исправлениями для z/OS осуществляется средством SMP/E (System Modification Program/Extended). Используется косвенная каталогизация с использованием системных символов, так что библиотека может быть каталогизирована как расположенная, например, на SYSRES-томе 2, тогда как имя этого тома разрешается системой во время начальной загрузки из системных символов. Обсуждение символов приведено в разделе 16.3.11 «Что такое системные символы».

Другая группа томов включает тома, содержащие продукты, отличные от z/OS, и тома, содержащие программное обеспечение, не изготовленное IBM. Эти тома можно объединить в одну группу. Большая часть программ, отличных от z/OS, обычно не располагается на SYSRES-томах, так как управление SYSRES-группами в SMP/E обычно осуществляется как для единого целого. Для остального программного обеспечения управление осуществляется отдельно. Эти тома не входят в SYSRES-группы, и поэтому существует только одна копия каждой библиотеки. В эту группу можно добавить требуемое количество томов, каждый из которых имеет отдельное имя диска.

Данными настройки (*customization data*) называются такие элементы как SYS1.PARMLIB, SYS1.PROCLIB, главный каталог, IODF, страничные наборы данных, JES-спулы, директория /etc и прочие элементы, важные для работы системы. Кроме того, в них хранятся данные SMP/E, используемые при управлении программным обеспечением.

Эти наборы данных не всегда расположены на DASD-томах, отличных от томов, содержащих программное обеспечение z/OS, поставляемое IBM; в некоторых инсталляциях PARMLIB и PROCLIB размещается на первом SYSRES-томе, тогда как в других инсталляциях они размещаются на томе главного каталога или на другом томе. Это вопрос выбора, зависящий от того, как осуществляется управление SYSRES-томами. Каждая инсталляция использует предпочтительный метод.

Во многих системах некоторые настройки, заданные IBM по умолчанию, неприемлемы, поэтому их требуется изменить. Пользовательские «выходы» и пользовательские модификации (usermod) добавляются в код IBM, вследствие чего система работает требуемым образом. Управление модификациями обычно осуществляется через SMP/E.

И, наконец, имеются *пользовательские данные* (*user data*), которые обычно представляют собой самый крупный пул дисковых томов. Он не является частью системных библиотек и представлен здесь для полноты. Он содержит рабочие, тестовые и пользовательские данные. Часто он разделяется на пулы и управляется системой SMS (System Managed Storage), которая может направлять данные на тома, управляемые соответствующим образом. Например, рабочие данные могут записываться на тома, для которых ежедневно выполняется резервное копирование, тогда как пользовательские данные могут копироваться раз в неделю и переноситься на магнитную ленту после короткого периода бездействия, чтобы освобождить дисковые тома для следующих данных.

z/OS содержит множество стандартных системных библиотек, таких как SYS1.PARMLIB, SYS1.LINKLIB, SYS1.LPALIB, SYS1.PROCLIB и SYS1.NUCLEUS. Некоторые из них связаны с обработкой начальной загрузки, тогда как другие связаны с порядком поиска вызываемых программ или системной безопасностью, как описано ниже:

- SYS1.PARMLIB содержит управляющие параметры всей системы.
- SYS1.LINKLIB содержит множество системных исполняемых модулей.
- SYS1.LPALIB содержит системные исполняемые модули, загружаемые в область загрузки модулей (link pack area) при инициализации системы.
- SYS1.PROCLIB содержит JCL-процедуры, распространяемые с z/OS.
- SYS1.NUCLEUS содержит базовые модули супервизора в системе.
- SYS1.SVCLIB содержит программы вызова супервизора.

16.3.2 SYS1.PARMLIB

SYS1.PARMLIB представляет собой обязательный секционированный набор данных, содержащий разделы, поставляемые компанией IBM и созданные при инсталляции. Он должен находиться на томе с прямым доступом, который может представлять собой резидентный системный том. PARMLIB является важным набором данных в операционной системе z/OS, подобным по функциям директории /etc в системе UNIX.

Назначение PARMLIB состоит в том, чтобы содержать множество параметров инициализации в предварительно заданной форме в едином наборе данных, сокращая, таким образом, необходимость ввода параметров оператором.

Все параметры и разделы набора данных SYS1.PARMLIB описываются в руководстве *z/OS MVS Initialization and Tuning Reference*, SA22-7592. Некоторые наиболее важные разделы PARMLIB рассматриваются в этой главе.

16.3.3 Область загрузки модулей (LPA)

Область загрузки модулей (link pack area, LPA) представляет собой раздел общей области адресного пространства. Она находится ниже области системных очередей (system queue area, SQA) и содержит перемещаемую область загрузки модулей (pageable link pack area, PLPA), фиксированную область загрузки модулей (fixed link pack area, FLPA), если такая существует, и изменяемую область загрузки модулей (modified link pack area, MLPA).

Модули в области загрузки модулей (LPA) загружаются в общую память и разделяются между всеми адресными пространствами в системе. Так как эти модули являются реентерабельными и не являются самоизменяющимися, каждый из них может использоваться множеством задач в любом числе адресных пространств одновременно. Модули, расположенные в LPA, не требуется переносить в виртуальную память, так как они уже находятся в виртуальной памяти.

Модули, находящиеся в любом участке LPA, всегда находятся в виртуальной памяти, а модули, находящиеся в FLPA, также всегда находятся в основной памяти. К модулям, находящимся в LPA, необходимо очень часто обращаться, чтобы не допустить изъятия их страниц. Когда страница находится в LPA (но не в FLPA), и к ней не обращаются постоянно разные адресные пространства, повышается вероятность ее изъятия.

16.3.4 Перемещаемая область загрузки модулей (PLPA)

PLPA представляет собой область общей памяти, загружаемую во время начальной загрузки (при выполнении «холодного» запуска с включенной опцией CLPA). Эта область содержит системные программы с доступом только для чтения, а также вы-

бранные инсталляцией реентерабельные пользовательские программы с доступом только для чтения, которые могут совместно использоваться пользователями системы. PLPA и расширенная PLPA содержит все разделы SYS1.LPALIB и других библиотек, заданных в активном LPALSTxx посредством параметра LPA в IEASYSxx или с консоли оператора при инициализации системы (это замещает спецификацию PARMLIB).

Можно использовать один или несколько разделов LPALSTxx в SYS1.PARMLIB для сцепления наборов данных программных библиотек вашей инсталляции с SYS1.LPALIB. Можно также использовать раздел LPALSTxx для добавления доступных только для чтения реентерабельных пользовательских программ вашей инсталляции в перемещаемую область загрузки модулей (PLPA). Система использует такое сцепление, называемое *LPALST-сцеплением*, для построения PLPA в процессе инициализации ядра. Библиотека SYS1.LPALIB должна размещаться на томе с прямым доступом, который может представлять собой резидентный системный том.

На рис. 16.3 представлен пример раздела LPALSTxx.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
EDIT     SYS1.PARMLIB(LPALST7B) - 01.03            Columns 00001 00072
Command ===>                                       Scroll ===> CSR
****** ***************************** Top of Data ******************************
000200 SYS1.LPALIB,
000220 SYS1.SERBLPA,
000300 ISF.SISFLPA,
000500 ING.SINGMOD3,
000600 NETVIEW.SCNMLPA1,
000700 SDF2.V1R4M0.SDGILPA,
000800 REXX.SEAGLPA,
001000 SYS1.SIATLPA,
001100 EOY.SEOYLPA,
001200 SYS1.SBDTLPA,
001300 CEE.SCEELPA,
001400 ISP.SISPLPA,
001600 SYS1.SORTLPA,
001700 SYS1.SICELPA,
001800 EUV.SEUVLPA,
001900 TCPIP.SEZALPA,
002000 EQAW.SEQALPA,
002001 IDI.SIDIALPA,
002002 IDI.SIDILPA1,
002003 DWW.SDWWLPA(SBOX20),
002010 SYS1.SDWWDLPA,
002020 DVG.NFTP230.SDVGLPA,
002200 CICSTS22.CICS.SDFHLPA(SBOXD3)
****** ***************************** Bottom of Data ***************************
```

Рис. 16.3. Пример раздела LPALST PARMLIB

16.3.5 Фиксированная область загрузки модулей (FLPA)

FLPA загружается во время начальной загрузки вместе с модулями, перечисленными в активном разделе IEAFIXxx библиотеки SYS1.PARMLIB. Эту область следует использовать только для модулей, производительность которых значительно возрастает при их размещении в фиксированной, а не в перемещаемой области. Наилучшими кандидатами для размещения в FLPA являются модули, использующиеся редко, но от которых требуется быстрое реагирование.

В FLPA можно включить модули из сцепления LPALST, сцепления linklist, SYS1.MIGLIB и SYS1.SVCLIB. FLPA выбирается посредством использования значения парамет-

ра FIX в IEASYSxx, добавляемого к IEAFIX для формирования имени раздела IEAFIXxx библиотеки PARMLIB, или с консоли оператора при инициализации системы.

На рис. 16.4 представлен раздел IEAFIX PARMLIB; часть модулей, расположенных в FLPA, принадлежат библиотеке SYS1.LPALIB.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
EDIT      SYS1.PARMLIB(IEAFIX00) - 01.00              Columns 00001
00072
Command ===>                                          Scroll ===> CSR
****** ***************************** Top of Data ******************************
000001 INCLUDE LIBRARY(SYS1.LPALIB) MODULES(
000002          IEAVAR00
000003          IEAVAR06
000004          IGC0001G
000005          )
000006 INCLUDE LIBRARY(FFST.V120ESA.SEPWMOD2) MODULES(
000007          EPWSTUB
000008          )
****** **************************** Bottom of Data ****************************
```

Рис. 16.4. Раздел IEAFIX PARMLIB

16.3.6 Изменяемая область загрузки модулей (MLPA)

MLPA может использоваться для содержания реентерабельных программ из авторизованных библиотек APF (см. раздел 18.7.1 «Авторизованные программы»), предназначенных для использования в качестве перемещаемого расширения области загрузки модулей для текущей начальной загрузки. Обратите внимание на то, что MLPA существует только на протяжении текущей IPL. Таким образом, если требуется использование MLPA, модули в MLPA должны быть заданы для каждого варианта начальной загрузки (включая быстрый запуск и «теплый» запуск). Когда система осуществляет поиск программы, MLPA просматривается перед PLPA. MLPA можно использовать во время начального запуска для временного изменения или обновления PLPA новыми модулями или модулями замены.

16.3.7 SYS1.PROCLIB

SYS1.PROCLIB представляет собой обязательный секционированный набор данных, содержащий JCL-процедуры, используемые для выполнения некоторых системных функций. JCL может использоваться для системных задач или для обработки программных задач, вызываемых оператором или программистом.

16.3.8 Подсистема главного планировщика

Подсистема главного планировщика (*master scheduler subsystem*) используется для установления связи между операционной системой и первичной подсистемой ввода заданий, в качестве которой может использоваться JES2 или JES3. При запуске z/OS

выполняется инициализация системных служб, таких как системный журнал и коммуникационная задача, и запускают адресное пространство главного планировщика, которое становится адресным пространством с первым номером (ASID=1).

Затем главный планировщик может запустить подсистему ввода заданий (JES2 или JES3). JES является первичной подсистемой ввода заданий. На многих рабочих системах JES не запускается сразу же; вместо этого пакет автоматизации запускает все задачи в контролируемой последовательности. Затем происходит запуск других подсистем. Все подсистемы определяются в разделе IEFSSNxx библиотеки PARMLIB. Эти подсистемы являются *вторичными подсистемами* (*secondary subsystems*).

Начальный загрузочный модуль MSTJCL00 находится в библиотеке SYS1.LINKLIB. Если требуются изменения, рекомендуется создать раздел MSTJCLxx в наборе PARMLIB. Суффикс определяется параметром MSTRJCL в разделе IEASYSxx PARMLIB. Раздел MSTJCLxx обычно называется *главным JCL* (*master JCL*). Он содержит операторы определения данных (DD) для всех системных входных и выходных наборов данных, необходимых для осуществления связи между операционной системой и JES.

В Примере 16.1 представлен образец раздела MSTJCLxx.

Пример 16.1. Образец главного JCL

```
File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------
EDIT  SYS1.PARMLIB(MSTJCL00) - 01.07  Columns 00001 00072
Command ===> Scroll ===> CSR
****** **************************** Top of Data ***********************
000100 //MSTRJCL    JOB MSGLEVEL=(1,1),TIME=1440
000200 //           EXEC PGM=IEEMB860,DPRTY=(15,15)
000300 //STCINRDR   DD SYSOUT=(A,INTRDR)
000400 //TSOINRDR   DD SYSOUT=(A,INTRDR)
000500 //IEFPDSI    DD DSN=SYS1.PROCLIB,DISP=SHR
000600 //           DD DSN=CPAC.PROCLIB,DISP=SHR
000700 //           DD DSN=SYS1.IBM.PROCLIB,DISP=SHR
000800 //IEFJOBS    DD DSN=SYS1.STCJOBS,DISP=SHR
000900 //SYSUADS    DD DSN=SYS1.UADS,DISP=SHR
****** **************************** Bottom of Data ***************
```

Когда главному планировщику необходимо выполнить запуск запускаемой задачи, система определяет, относится ли команда START к процедуре или к заданию. Если раздел MSTJCLxx содержит оператор IEFJOBS DD, система осуществляет поиск раздела, запрашиваемого командой START, в сцепленном IEFJOBS DD.

Если сцепленный IEFJOBS не содержит раздела с таким именем, или если сцепленный IEFJOBS не существует, система осуществляет поиск раздела, запрашиваемого командой START, в IEFPDSI DD. Если раздел найден, система изучает первую запись на наличие допустимого оператора JOB и, если он существует, использует раздел как

исходный JCL-код для запускаемой задачи. Если раздел не содержит допустимый оператор JOB в первой записи, система предполагает, что исходный JCL-код является процедурой и создает JCL для вызова процедуры.

После создания (или обнаружения) исходного JCL-кода, система обрабатывает JCL. При поставке MSTJCL00 содержит оператор IEFPDSI DD, определяющий набор данных, содержащий исходный JCL-код процедуры для запускаемых задач. Обычно в качестве набора данных используется SYS1.PROCLIB; он может представлять собой сцепление. Для того чтобы полезная работа могла быть выполнена, SYS1.PROCLIB должна, по меньшей мере, содержать процедуру первичного JES, как показано в следующем разделе.

16.3.9 Библиотека процедур заданий

SYS1.PROCLIB содержит каталогизированную процедуру JES2. Эта процедура определяет библиотеки процедур, связанные с заданиями, как показано в Примере 16.2.

Пример 16.2. Как определить библиотеки процедур в процедуре JES2

```
//PROC00      DD DSN=SYS1.PROCLIB,DISP=SHR
//            DD DSN=SYS3.PROD.PROCLIB,DISP=SHR
//PROC01      DD DSN=SYS1.PROC2,DISP=SHR
...
//PROC99      DD DSN=SYS1.LASTPROC,DISP=SHR
...
```

Многие инсталляции имеют очень длинные списки библиотек процедур в JES-процедуре. Это вызвано тем, что JCLLIB является сравнительно недавней инновацией.

Следует обращать внимание на то, сколько пользователей может удалять эти библиотеки, так как JES не запустится, если хотя бы одна библиотека отсутствует. Обычно используемую библиотеку удалить нельзя, однако JES не блокирует эти библиотеки, хотя использует их постоянно.

Для замещения спецификации по умолчанию используется следующий оператор:

```
/*JOBPARM PROCLIB=
```

После имени библиотеки процедур задается имя оператора DD в процедуре JES2, указывающее на используемую библиотеку.Для примера, на рис. 16.2 предположим, что запущено задание в классе A и что класс имеет заданную по умолчанию спецификацию PROCLIB в PROC00. Для того чтобы использовать процедуру, расположенную в SYS1.LASTPROC, потребуется включить в JCL-код следующий оператор:

```
/*JOBPARM PROCLIB=PROC99
```

Другой способ определения библиотеки процедур состоит в использовании JCL-оператора JCLLIB. Этот оператор позволяет составлять и использовать процедуры, не используя библиотеки системных процедур. Система осуществляет поиск в библиотеках в том порядке, в котором они перечисляются в операторе JCLLIB, прежде чем осуществлять поиск в незаданных явно используемых по умолчанию библиотеках системных процедур.

Использование оператора JCLLIB представлено в Примере 16.3.

Пример 16.3. Оператор JCLLIB

```
//MYJOB       JOB
//MYLIBS      JCLLIB  ORDER=(MY.PROCLIB.JCL,SECOND.PROCLIB.JCL)
//S1          EXEC  PROC=MYPROC1
...
```

Если предположить, что системная библиотека процедур по умолчанию содержит только SYS1.PROCLIB, система выполняет в библиотеках поиск процедуры MYPROC1 в следующем порядке:

1. MY.PROCLIB.JCL.
2. SECOND.PROCLIB.JCL.
3. SYS1.PROCLIB.

16.3.10 Порядок поиска для программ

При запросе программы через системную службу (такую как LINK, LOAD, XCTL или ATTACH) с использованием опций по умолчанию, система ищет ее в следующей последовательности:

1. Job pack area (JPA).

   Программа, находящаяся в JPA, уже была загружена в требуемое адресное пространство. Если можно использовать копию в JPA, она будет использоваться. В противном случае система либо ищет новую копию, либо откладывает запрос до тех пор, пока не станет доступна копия в JPA (например, система откладывает запрос до тех пор, пока не завершится обработка для предыдущего инициатора запроса, прежде чем можно будет использовать модуль многократного последовательного использования, уже находящийся в JPA).

2. TASKLIB.

   Программа может распределить один или несколько наборов данных в сцеплении TASKLIB. Модули, загружаемые неавторизованными задачами, находящиеся в TASKLIB, должны быть перенесены в приватную область виртуальной памяти, прежде чем их можно будет запустить. Модули, которые ранее были загружены в общую область виртуальной памяти (модули в LPA или модули, загруженные авторизованной программой в CSA), должны быть загружены в общую область виртуальной памяти, прежде чем их можно будет запустить.

3. STEPLIB или JOBLIB.

   Представляют собой специальные DD-имена, которые можно использовать для распределения наборов данных, поиск в которых необходимо выполнять до заданной по умолчанию последовательности поиска программ в системе. Наборы данных могут размещаться в сцепленных STEPLIB и JOBLIB в JCL-коде или программой, использующей динамическое распределение. Однако поиск модулей будет выполнен только в одном из них. Если для определенного шага задания распределены обе библиотеки STEPLIB и JOBLIB, система выполняет поиск в STEPLIB, игнорируя JOBLIB.

В любых наборах данных, сцепленных в STEPLIB или JOBLIB, поиск будет осуществляться после TASKLIB, но перед LPA. Модули, находящиеся в STEPLIB или JOBLIB, должны быть перенесены в приватную область виртуальной памяти, прежде чем их можно будет запустить. Модули, которые ранее были загружены в общую область виртуальной памяти (модули в LPA или модули, загруженные авторизованной программой в CSA), должны быть загружены в общую область виртуальной памяти, прежде чем их можно будет запустить.

4. LPA, поиск в которой осуществляется в следующем порядке:

   а) модули в динамической LPA, заданные в разделах PROGxx;

   б) модули в фиксированной LPA (FLPA), заданные в разделах IEAFIXxx;

   в) модули в изменяемой LPA (MLPA), заданные в разделах IEALPAxx;

   г) модули в перемещаемой LPA (PLPA), загружаемые из библиотек, заданных в LPALSTxx или PROGxx.

   Модули в LPA загружаются в общую память, разделяемую всеми адресными пространствами в системе. Так как эти модули являются реентерабельными и не являются самоизменяющимися, каждый из них может использоваться множеством задач в любом числе адресных пространств одновременно. Модули, расположенные в LPA, не требуется переносить в виртуальную память, так как они уже находятся в виртуальной памяти.

5. Библиотеки в linklist, заданные в PROGxx и LNKLSTxx.
   По умолчанию linklist начинается с SYS1.LINKLIB, SYS1.MIGLIB и SYS1.CSSLIB. Однако можно изменить этот порядок, используя SYSLIB в PROGxx, и добавить другие библиотеки в сцепление linklist. Система должна перенести модули, находящиеся в linklist, в приватную область виртуальной памяти, прежде чем можно будет запускать программы.
   Заданный по умолчанию порядок поиска можно изменить, указав определенные опции в макросе, используемом для вызова программ. На порядок поиска, используемый в системе, влияют параметры EP, EPLOC, DE, DCB и TASKLIB. Дополнительные сведения об этих параметрах см. в разделе, посвященном поиску загрузочного модуля, в руководстве *z/OS MVS Programming: Assembler Services Guide*. Некоторые подсистемы IBM (особенно CICS и IMS) и приложения (в частности, ISPF) используют эти средства для установления другого порядка поиска программ.

16.3.11 Что такое системные символы

Системные символы (*system symbols*) представляют собой элементы, позволяющие различным системам z/OS совместно использовать определения PARMLIB, сохраняя уникальные значения в этих определениях. Системные символы действуют подобно переменным в программе; они могут принимать различные значения на основании входных данных программы. При указании системного символа в разделяемом определении PARMLIB системный символ выступает в качестве «заполнителя». Каждая система, использующая определение, во время инициализации замещает системный символ уникальным значением.

Каждый системный символ имеет имя (которое начинается с амперсанда (&) и может заканчиваться точкой (.)) и подстановочный текст, представляющий собой строку символов, которую система подставляет вместо символа в каждом его вхождении.

Существует два типа системных символов:

Динамические Подстановочный текст может измениться в любой момент начальной загрузки.

Статические Подстановочный текст определен при инициализации системы и остается неизменным на протяжении начальной загрузки.

Некоторые символы зарезервированы для использования в системе. Для вывода символов, используемых в системе, применяется команда D SYMBOLS. В Примере 16.4 представлены результаты ввода этой команды.

Пример 16.4. Фрагмент выходных данных команды D SYMBOLS (некоторые строки удалены)

```
HQX7708 ----------------- SDSF PRIMARY OPTION MENU --
COMMAND INPUT ===> -D SYMBOLS
    IEA007I STATIC SYSTEM SYMBOL VALUES
       &SYSALVL. = «2»
       &SYSCLONE. = «70»
       &SYSNAME. = «SC70»
       &SYSPLEX. = «SANDBOX»
       &SYSR1. = «Z17RC1»
       &ALLCLST1. = «CANCEL»
       &CMDLIST1. = «70,00»
       &COMMDSN1. = «COMMON»
       &DB2. = «V8»
       &DCEPROC1. = «.»
       &DFHSMCMD. = «00»
       &DFHSMHST. = «6»
       &DFHSMPRI. = «NO»
       &DFSPROC1. = «.»
       &DLIB1. = «Z17DL1»
       &DLIB2. = «Z17DL2»
       &DLIB3. = «Z17DL3»
       &DLIB4. = «Z17DL4»
       &IEFSSNXX. = "R7"
       &IFAPRDXX. = "4A"
```

Раздел IEASYMxx библиотеки PARMLIB является единым местом указания системных параметров для каждой системы в многосистемной среде. IEASYMxx содержит операторы, определяющие статические системные символы и задающие разделы IEASYSxx библиотеки PARMLIB, содержащие системные параметры (оператор SYSPARM).

В Примере 16.5 представлен раздел IEASYMxx библиотеки PARMLIB.

Пример 16.5. Фрагмент раздела IEASYMxx библиотеки PARMLIB (некоторые строки удалены)

```
SYSDEF        SYSCLONE(&SYSNAME(3:2))
SYMDEF          (&SYSR2='&SYSR1(1:5).2')
SYMDEF          (&SYSR3='&SYSR1(1:5).3')
SYMDEF          (&DLIB1='&SYSR1(1:3).DL1')
SYMDEF          (&DLIB2='&SYSR1(1:3).DL2')
SYMDEF          (&DLIB3='&SYSR1(1:3).DL3')
SYMDEF          (&DLIB4='&SYSR1(1:3).DL4')
SYMDEF          (&ALLCLST1='CANCEL')
SYMDEF          (&CMDLIST1='&SYSCLONE.,00')
SYMDEF          (&COMMDSN1='COMMON')
SYMDEF          (&DFHSMCMD='00')
SYMDEF          (&IFAPRDXX='00')
SYMDEF          (&DCEPROC1='.')
SYMDEF          (&DFSPROC1='.')
SYSDEF         HWNAME(SCZP901)
LPARNAME        (A13)
SYSNAME         (SC70)
SYSPARM         (R3,70)
SYMDEF          (&IFAPRDXX='4A')
SYMDEF          (&DFHSMHST='6')
SYMDEF          (&DFHSMPRI='NO')
SYMDEF          (&DB2='V8')
```

В этом примере переменная &SYSNAME имеет значение, заданное ключевым словом SYSNAME; в данном случае, используется значение SC70. Так как каждая система в сисплексе имеет уникальное имя, в указании ресурсов, уникальных для системы, можно использовать &SYSNAME, где это разрешено. Например, можно задать для набора данных SMF имя SYS1.&SYSNAME.MAN1, которое при запуске в SC70 после подстановки будет иметь вид SYS1.SC70.MAN1.

Можно использовать переменные для построения значений других переменных. На рис. 16.5 показано, что &SYSCLONE принимает значение &SYSNAME, начиная с позиции 3 и длиной 2. В данном случае &SYSCLONE имеет значение 70. Подобным образом, мы видим, что &SYSR2 создается из первых 5 позиций &SYSR1 с суффиксом 2. Где определяется &SYSR1? &SYSR1 определяется системой на основании серийного номера (VOLSER) загрузочного тома. Если возвратиться к рис. 16.4, мы увидим значения &SYSR1 и &SYSR2.

Мы также видим определение глобальной переменной, определенной во всех системах (&IFAPRDXX со значением 00), и ее переопределение в SC70 на значение 4A.

Системные символы используются в случаях, когда несколько систем z/OS используют одну библиотеку PARMLIB. В этом случае использование символов позволя-

ет использовать отдельные разделы с символьной подстановкой, вместо того чтобы каждая система требовала уникальный раздел. Раздел LOADxx определяет раздел IEASYMxx, который система может использовать.

16.4 Управление системной производительностью

Задача настройки системы является итерационным и непрерывным процессом, оказывающим непосредственное влияние на всех пользователей системных ресурсов на предприятии. Компонент z/OS Workload Management (WLM), обсуждавшийся в разделе «Что такое управление рабочей нагрузкой?», является важной частью этого процесса и включает первоначальную настройку выбора требуемых параметров для различных компонентов системы и подсистем.

После достижения системой рабочего состояния и установления критериев выбора выполняемых заданий с использованием классов заданий и приоритетов, WLM управляет распределением доступных ресурсов в соответствии с параметрами, заданными инсталляцией.

Однако WLM может работать только с доступными ресурсами. Если их недостаточно для требований инсталяции, даже оптимальное распределение может не дать требуемый результат; следует изучить другие области системы, чтобы определить возможность увеличения доступных ресурсов. По мере роста требований к системе, когда возникает необходимость сместить приоритеты или получить дополнительные ресурсы (более мощный процессор, больше памяти или больше терминалов), системному программисту требуется изменить параметры WLM, чтобы отразить измененные условия.

16.5 Конфигурирование устройств ввода-вывода

Необходимо определить конфигурации ввода-вывода в операционной системе (программном обеспечении) и канальной подсистеме (аппаратном обеспечении). Компонент Hardware Configuration Definition (HCD) операционной системы z/OS консолидирует процессы конфигурации аппаратного и программного ввода-вывода под управлением единого интерактивного интерфейса конечного пользователя. Выходными данными HCD является файл определения ввода-вывода (IODF), содержащий данные конфигурации ввода-вывода. IODF используется для определения нескольких аппаратных и программных конфигураций в операционной системе z/OS.

При активации нового файла IODF, HCD определяет конфигурацию ввода-вывода в канальной подсистеме и/или операционной системе. Используя функцию активации HCD или команду оператора z/OS ACTIVATE, можно вносить изменения в текущую конфигурацию без перезагрузки системы (initial program load, IPL) или сброса по питанию (power-on reset, POR) аппаратного обеспечения. Внесение изменений при работающей системе называется *динамическим конфигурированием* (*dynamic configuration*) или *динамическим реконфигурированием* (*dynamic reconfiguration*).

16.6 Следование процессу управления изменениями

Руководство вычислительного центра обычно отвечает за соглашения об уровне сервиса (SLA) часто посредством команды специалистов или менеджеров по сервису. Изменение механизмов и практик управления в вычислительном центре выполняется для обеспечения соответствия SLA.

Реализация любого изменения должна происходить под управлением операторского персонала. При внедрении в рабочую среду изменений, вызвавших проблемы или нестабильность, операторский персонал отвечает за наблюдение, отчетность, а затем за управление действиями по исправлению проблемы или отмене изменений.

Хотя чаще всего системные программисты разрабатывают и реализуют собственные изменения, иногда изменения производятся по запросу через систему управления изменениями. Все инструкции для операторского персонала или других групп заносятся в журнал изменений, и утверждение каждой группой является обязательным.

Реализация изменений в бизнес-приложениях обычно выполняется аналитиком производственного контроля. Изменения в приложениях обычно размещаются в тестовых библиотеках, и официальный запрос (с контрольной записью) вызывает перенос программ из тестовых библиотек в рабочую среду.

Процедуры, использующиеся при внесении изменений, должны быть переданы всем заинтересованным сторонам. Когда все стороны посчитают описание изменения полным, тогда рассматривается вариант его реализации, вследствие чего оно либо планируется, либо откладывается, либо отклоняется.

При планировании изменений следует учитывать следующие факторы:

- преимущества, которые даст изменение;
- что произойдет, если изменение не будет выполнено;
- ресурсы, требуемые для реализации изменения;
- относительная важность запроса на изменение в сравнении с другими;
- взаимные зависимости между запросами на изменения.

Любое изменение предполагает риск. Одно из преимущество мэйнфрейма состоит в том, что он обеспечивает очень высокую доступность. Поэтому все изменения должны подвергаться тщательному контролю и управлению. Значительная часть времени любого системного программиста занята планированием и оценкой рисков реализации изменения. Одним из наиболее важных аспектов изменения является наличие способа его отмены и перехода к предыдущему состоянию.

16.6.1 Оценка рисков

Как правило, руководство вычислительного центра проводит еженедельное собрание для обсуждения, утверждения или отклонения изменений. Эти изменения могут относиться к приложениям, системе, сети, оборудованию или электропитанию.

Важной частью любого изменения является *оценка рисков* (*risk assessment*), при которой изменение рассматривается и оценивается с точки зрения риска для системы. Внесение изменений с невысокими рисками может разрешаться в течение дня, тогда как внесение изменений с более высокими рисками планируется на время простоя.

Также в вычислительных центрах часто используются периоды невысокого и высокого риска, что оказывает воздействие на решения. Например, если система осуществляет авторизацию операций по кредитным картам, тогда периоды перед крупными национальными праздниками обычно характеризуются чрезвычайно высокой занятостью, и внесение изменений в эти периоды может быть запрещено. Кроме того, на предприятиях розничной торговли периодами чрезвычайной занятости могут быть ежегодные распродажи, что может вызвать отклонение изменений.

ИТ-организации достигают своих целей посредством применения организованных процессов и политик управления изменениями. К таким целям относятся:

- высокая доступность сервиса;
- повышенная безопасность;
- готовность аудита;
- снижение расходов.

16.6.2 Система управления регистрацией изменений

Система управления регистрацией изменений (*change control record system*), как правило, используется для обеспечения запрашивания, отслеживания и утверждения изменений. Она обычно работает совместно с *системой управления проблемами* (*problem management system*). Например, если в понедельник утром в рабочей системе возникла серьезная проблема, первым делом нужно рассмотреть изменения, которые были внесены в выходные дни, чтобы определить, вызвали ли они какие-либо проблемы.

Регистрация также показывает, что система находится под контролем, что часто бывает необходимо доказать аудиторам, особенно в секторе финансовых услуг с сильным регулированием. Закон Сарбейнса-Оксли 2002 года в США, регулирующий корпоративное управление, оговаривает необходимость эффективной системы внутреннего контроля. Эффективное управление изменениями и проблемами в сфере информационных услуг необходимо для соответствия этому закону. Кроме того, восьмая директива Закона о компаниях в Европейском Союзе, вокруг которой на момент написания этой книги велось обсуждение, касается подобных вопросов, что и Закон Сарбейнса-Оксли.

По этим причинам прежде чем реализовать какое-либо изменение, как минимум, должен быть определен набор контролируемых документов, называемых *формами запроса изменений* (*change request forms*). Они должны оговаривать следующее:

- Кто – отдел, группа или человек, требующий изменение, ответственный за внесение изменения, ответственный за тестирование и ответственный за отмену изменения при необходимости, а также ответственные за утверждение изменений.
- Что – затрагиваемые системы или службы (например электронная почта, файловая служба, домен и т. д.). Требуется включить максимальное количество подробностей. В идеале, следует включить полные инструкции, чтобы при необходимости внесение изменений мог выполнить кто-нибудь другой.
- Когда – дата/время начала и приблизительная продолжительность изменения.

Часто указывается три даты: запрашиваемая, запланированная и действительная.

- Где – область изменений, т. е. подразделения, здания, отделы или группы, затрагиваемые изменением или требующие помощи при изменении.
- Как – план реализации изменений и план отмены изменений при необходимости.
- Приоритет – высокий, средний, низкий, обычный, аварийный, датированный (например, перевод часов).
- Риск – высокий, средний, низкий.
- Воздействие – что произойдет, если реализовать изменение; что произойдет, если не реализовать изменение; какие другие системы могут быть затронуты; что произойдет в случае возникновения неожиданной ситуации.

16.6.3 Производственный контроль

Производственным контролем обычно занимается специализированный персонал, управляющий планированием пакетных заданий, используя такие инструменты как Tivoli® Workload Scheduler для построения и управления сложным планированием пакетных задач. Это может включать ежедневное и еженедельное создание резервных копий, выполняемое в определенных точках сложной последовательности пакетов приложений. При этом также могут отключаться и затем повторно включаться базы данных и оперативные службы. Хотя при внесении таких изменений, аналитикам производственного контроля часто требуется учитывать национальные праздники и другие особые события, например, (в случае предприятий розничной торговли) зимнюю распродажу.

Аналитики производственного контроля отвечают также за перенос последней версии программы в рабочую среду. Это обычно включает перенос исходного кода в защищенную рабочую библиотеку, перекомпиляцию кода для генерирования рабочего загрузочного модуля и помещение модуля в рабочую загрузочную библиотеку. JCL копируется и обновляется в соответствии с производственными стандартами и помещается в соответствующие библиотеки процедур, и пакеты приложений добавляются в планировщик заданий.

Кроме того, если требуется выполнить добавление новой библиотеки в linklist или ее авторизацию, может привлекаться системный программист.

16.7 Конфигурирование консолей

Управление z/OS включает управление оборудованием, в частности – процессорами и периферийными устройствами (включая консоли, на которых работают операторы), а также программным обеспечением, в частности – операционной системой z/OS, подсистемой ввода заданий, подсистемами, управляющими автоматизированными операциями (например NetView®), и всеми приложениями, выполняющимися в z/OS.

Управление системой z/OS включает следующее:

- обработку сообщений и команд, составляющую основу взаимодействия оператора с z/OS и основу автоматизации z/OS;
- консольные операции, определяющие способ взаимодействия операторов с z/OS для мониторинга или управления аппаратным и программным обеспечением.

При планировании операций в z/OS требуется учитывать, как операторы используют консоли для выполнения своей работы, и как управлять сообщениями и командами. Системному программисту нужно убедиться в том, что операторы получают на своих консолях требуемые сообщения для выполнения своих задач, и выбрать соответствующие сообщения для их подавления, автоматизации и прочих видов обработки сообщений.

С точки зрения управления z/OS при планировании важно учитывать, каким образом инсталляция осуществляет консольное восстановление, а также должен ли оператор перезагружать систему для изменения опций обработки.

Так как сообщения также являются основой для автоматизированных операций, системный программист должен понимать принципы обработки сообщений для планирования автоматизации в z/OS.

Чем больше инсталляций используют многосистемные среды, тем более важной становится необходимость согласования операционной деятельности этих систем. Даже в единичных системах z/OS инсталляции нужно контролировать связь между функциональными областями.

Как в односистемных, так и в многосистемных средах команды, вводимые операторами с консолей, могут представлять угрозу безопасности и требуют тщательного согласования. Как планировщику, системному программисту нужно быть уверенным в квалификации операторов z/OS.

Консольная конфигурация (*console configuration*) содержит различные консоли, используемые операторами для связи с z/OS. Инсталляция сначала определяет устройства ввода-вывода, которые она может использовать как консоли через Hardware Configuration Definition (HCD) – интерактивный интерфейс хоста, позволяющий системному программисту определять аппаратную конфигурацию для канальной подсистемы и операционной системы.

Hardware Configuration Manager (HCM) представляет собой графический пользовательский интерфейс для HCD. HCM взаимодействует с HCD в клиент-серверном режиме (т. е. HCM работает на рабочей станции, и HCD выполняется на хосте). Базовые системы требуют наличия внутренней модели подключений к устройствам, однако для системного программиста более удобным и эффективным методом будет поддержка (и дополнение) этой модели в визуальном формате. HCM обеспечивает синхронизацию конфигурационных данных в файле на рабочей станции с IODF на хосте. Несмотря на то, что можно использовать HCD непосредственно для задач конфигурирования оборудования, многие клиенты предпочитают использовать исключительно HCM из-за графического интерфейса.

Помимо HCD, после определения устройств операционной системе z/OS сообщается, какие устройства следует использовать как консоли, путем указания соответствующих номеров устройств в разделе CONSOLxx библиотеки PARMLIB.

Как правило, операторы в системе z/OS получают сообщения и вводят команды в консолях MCS и SMCS. Они могут использовать другие консоли (например, консоли NetView) для взаимодействия с z/OS, однако в данной книге мы рассматриваем консоли MCS, SMCS и EMCS, так как они чаще всего используются в вычислительных центрах, работающих с z/OS.

Консоли MCS (Multiple Console Support, MCS) представляют собой устройства с локальным подключением к системе z/OS, обеспечивающие базовые средства связи между операторами и z/OS. Консоли MCS подключены к управляющим устройствам, *не* поддерживающим сетевую архитектуру систем или протоколы SNA.

Консоли SMCS (SNA Multiple Console Support) представляют собой устройства, которые не должны быть локально подключены к системе z/OS, обеспечивающие базовые средства связи между операторами и z/OS. Консоли SMCS используют z/OS Communications Server для обеспечения связи между операторами и z/OS вместо прямого ввода-вывода в консольное устройство.

Консоли EMCS (Extended Multiple Console Support) представляют собой устройства (отличные от консолей MCS и SMCS), в которых операторы или программы могут вводить команды и получать сообщения. Определение консолей EMCS в консольной конфигурации позволяет системным программистам расширять количество консолей за максимально допустимый предел для консолей MCS, равный 99 для каждой системы z/OS в сисплексе.

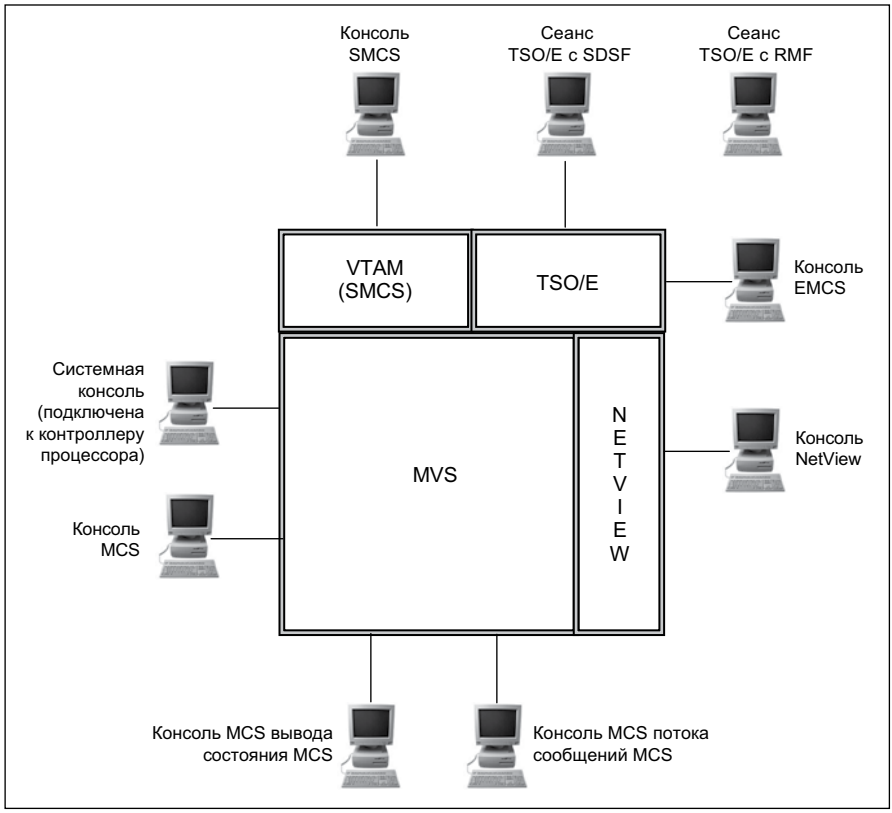


Рис. 16.5. Пример консольной конфигурации для системы z/OS

Системный программист определяет эти консоли в конфигурации в соответствии с их функциями. Важные сообщения, требующие реагирования, могут направляться оператору, который может вводить команды с консоли. Другая консоль может высту-

пать в качестве монитора для вывода сообщений оператору, работающему в некоторой функциональной области, например, с библиотекой пулов магнитных лент, или для вывода сообщений о принтерах, установленных в инсталляции.

На рис. 16.5 показана консольная конфигурация для системы z/OS, включающая системную консоль, консоль SMCS, NetView и TSO/E.

Функции *системной консоли* входят в консоль управления аппаратными средствами (Hardware Management Console, HMC). Оператор может использовать системную консоль для запуска z/OS и других системных программ, а также в ситуациях восстановления, когда другие консоли недоступны.

Помимо консолей MCS и SMCS, система z/OS, представленная на рис. 16.5, содержит консоль NetView, определенную в ней. NetView работает с системными сообщениями и командными списками при автоматизации задач оператора z/OS. Многие системные операции можно контролировать с консоли NetView.

Пользователи могут осуществлять мониторинг многих системных функций z/OS с терминалов TSO/E. Используя средство SDSF (System Display and Search Facility) и средство RMF™ (Resource Measurement Facility), пользователи TSO/E могут осуществлять мониторинг z/OS и реагировать на проблемы балансировки нагрузки и производительности. Авторизованный пользователь TSO/E может также инициировать сеанс расширенной консоли MCS (EMCS) для взаимодействия с z/OS.

На рис. 16.5 показаны следующие консоли MCS:

- Консоль MCS, с которой оператор может просматривать сообщения и вводить команды z/OS. Эта консоль работает в полнофункциональном режиме, так как она может получать сообщения и принимать команды. Оператор может контролировать операции системы z/OS с консоли MCS или SMCS.
- Консоль MCS вывода состояния.

  Оператор может просматривать информацию о состоянии системы, используя команды DEVSERV, DISPLAY, TRACK или CONFIG. Однако так как эта консоль является консолью вывода состояния, оператор не может вводить команды с консоли. Оператор в полнофункциональном режиме консоли может вводить эти команды и направлять выходные данные для просмотра на консоль вывода состояния.
- Консоль MCS потока сообщений.

  Консоль потока сообщений может выводить системные сообщения. Оператор может просматривать сообщения, направленные на эту консоль. Однако эта консоль является консолью потока сообщений, поэтому оператор не может вводить команды через нее. Коды перенаправления (routing codes) и уровень информации сообщений для консоли определяется таким образом, чтобы система могла направлять требуемые сообщения для вывода на экран консоли. Таким образом, оператор, отвечающий за функциональную область, например, за библиотеку пулов магнитных лент, может просматривать сообщения MOUNT.

Во многих инсталляциях вместо того, чтобы использовать большое количество экранов, используются рабочие станции операторов, выводящие множество экранов на одном многооконном дисплее. Как правило, консоль аппаратных средств исполь-

зуется отдельно, но большинство других терминалов объединяется. Управление системами осуществляется посредством извещений об исключительных ситуациях из системы автоматизации.

IBM Open Systems Adapter-Express Integrated Console Controller (OSA-ICC) представляет собой современный способ объединения консолей. OSA-ICC использует TCP/IP-соединения через Ethernet LAN для подключения к персональным компьютерам как к консолям через подключение TN3270 (telnet).

16.8 Инициализация системы

Начальная загрузка (initial program load, IPL) представляет собой акт загрузки копии операционной системы с диска в основную память процессора и ее выполнение.

Системы z/OS предназначены для непрерывной работы в течение многих месяцев между перезагрузками, что позволяет обеспечить непрерывную доступность рабочих задач. Стандартной причиной для перезагрузки является внесение изменений, и уровень изменений в системе определяет график перезагрузки. Например:

- Тестовую систему можно перезагружать ежедневно или даже чаще.
- Банковскую систему высокой доступности можно перезагружать раз в год или даже реже, чтобы обновить версии программного обеспечения.
- Часто причиной перезагрузок могут быть внешние факторы, например, необходимость тестирования и обслуживания систем питания в машинном зале.
- Иногда плохо работающее программное обеспечение нерационально использует системные ресурсы, которые можно восстановить только перезагрузкой, однако такое поведение обычно подлежит изучению и коррекции.

Многие изменения, которые в прошлом требовали перезагрузки, теперь можно реализовывать динамически. Примеры таких изменений:

- добавление библиотеки в linklist для подсистемы, такой как CICS;
- добавление модулей в LPA.

z/OS перезагружается с использованием консоли управления аппаратными средствами (Hardware Management Console, HMC). Для перезагрузки z/OS необходимо предоставить следующую информацию:

- адрес устройства загрузочного тома;
- раздел LOADxx, содержащий указатели на системные параметры;
- набор данных IODF, содержащий конфигурационную информацию;
- адрес устройства тома IODF.

16.8.1 Процесс инициализации

Процесс инициализации системы (рис. 16.6) готовит программу управления системой и ее среду к выполнению работы данной инсталляции. Этот процесс состоит из следующих этапов:

- инициализация системы и памяти, включая создание адресных пространств системных компонентов;
- инициализация главного планировщика и инициализация подсистемы.

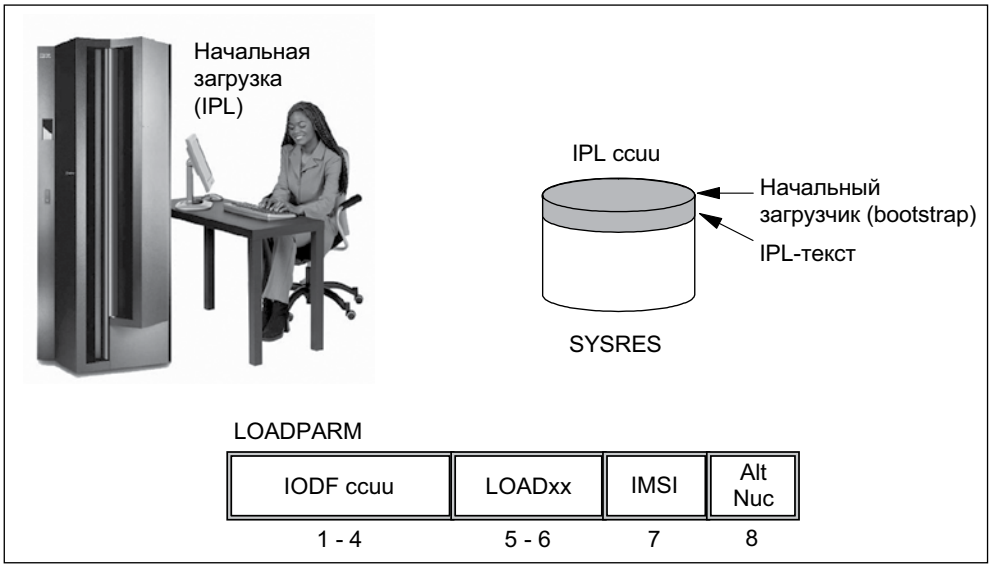


Рис. 16.6. Загрузка машины

После инициализации системы, когда подсистема ввода заданий активна, инсталляция может передавать задания для обработки с использованием команды START, LOGON или MOUNT.

Процесс инициализации начинается, когда системный программист выбирает функцию LOAD на консоли HMC. z/OS находит всю включенную и доступную основную память и начинает создавать различные системные области.

Не все диски, подключенные к CPU, содержат загружаемый код. Диск, содержащий загружаемый код, называется загрузочным (IPLable) диском, а если точнее, SYSRES-томом.

Загрузочные диски содержат начальный загрузчик (bootstrap module) на нулевой дорожке нулевого цилиндра. При начальной загрузке этот начальный загрузчик загружается в память по нулевому реальному адресу, и ему передается управление. Затем начальный загрузчик считывает программу управления начальной загрузкой IEAIPL00 (также называемую IPL-текстом) и передает ей управление. Это, в свою очередь, запускает более сложную задачу загрузки и запуска операционной системы.

После загрузки начального загрузчика и передачи управления IEAIPL00, IEAIPL00 готовит среду, подходящую для запуска программ и модулей, составляющих операционную систему, следующим образом:

1. IEAIPL00 очищает основную память, записывая в нее нули, прежде чем определить области памяти для главного планировщика.
2. IEAIPL00 находит набор данных SYS1.NUCLEUS на SYSRES-томе и загружает с него набор программ, называемых модулями инициализации ресурсов начальной загрузки (IPL Resource Initialization Modules, IRIM).
3. Эти IRIM начинают создавать нормальную среду операционной системы из блоков управления и подсистем.

Некоторые из наиболее важных задач, выполняющихся модулями IRIM, перечислены ниже:

- Чтение информации LOADPARM, вводимой с аппаратной консоли при выполнении команды IPL.
- Поиск тома, заданного в LOADPARM, на котором находится набор данных IODF. IRIM, сначала пытается найти LOADxx в SYS0.IPLPARM. Если это не удается, выполняется поиск SYS1.IPLPARM и т. д. до SYS9.IPLPARM. Если к этому моменту раздел LOADxx все еще не найден, поиск продолжается в SYS1.PARMLIB (если система и там не найдет LOADxx, система переходит в состояние ожидания).
- Если раздел LOADxx найден, он открывается, и из него считывается информация, включающая суффикс ядра (если он не замещен в LOADPARM), имя главного каталога и суффикс используемого раздела IEASYSxx.
- Загрузка ядра операционной системы.
- Инициализация виртуальной памяти в адресном пространстве главного планировщика для областей SQA (System Queue Area), ESQA (Extended SQA), LSQA (Local SQA) и PSA (Prefixed Save Area). В конце последовательности начальной загрузки PSA заменяет IEAIPL00 в нулевой ячейке реальной памяти, где он будет храниться.
- Инициализация управления реальной памятью, включая таблицу сегментов для главного планировщика, записей таблицы сегментов для областей общей памяти и таблицы фреймов страниц.

Затем последний из IRIM загружает первую часть программы инициализации ядра (Nucleus Initialization Program, NIP), которая вызывает модули инициализации ресурсов (Resource Initialization Modules, RIM); один из ранних таких модулей запускает коммуникации с NIP-консолью, определенной в IODF.

Система продолжает процесс инициализации, интерпретируя и действуя в соответствии с заданными системными параметрами. NIP выполняет следующие основные функции инициализации:

- Расширяет области SQA и ESQA до размерах, заданных системным параметром SQA.
- Создает перемещаемую область загрузки модулей (pageable link pack area, PLPA) и расширенную PLPA для «холодного» запуска; сбрасывает таблицы таким образом, чтобы они соответствовали существующей области PLPA и расширенной PLPA для быстрого запуска или «теплого» запуска. Дополнительные сведения о быстром запуска и «теплом» запуске см. в руководстве *z/OS MVS Initialization and Tuning Reference*.
- Загружает модули в фиксированную область загрузки модулей (fixed link pack area, FLPA) или в расширенную FLPA. Обратите внимание на то, что NIP выполняет эту функцию, только если задан системный параметр FIX.
- Загружает модули в изменяемую область загрузки модулей (modified link pack area, MLPA) и в расширенную MLPA. Обратите внимание на то, что NIP выполняет эту функцию, только если задан системный параметр MLPA.
- Выделяет виртуальную память для общей системной области (CSA) и расширенной CSA. Объем выделяемой памяти зависит от значений, заданных в системном параметре CSA при начальной загрузке.

- Защищает от изменения страницы NUCMAP, PLPA и расширенную PLPA, MLPA и расширенную MLPA, FLPA и расширенную FLPA, а также фрагменты ядра.

> **Примечание.** Инсталляция может отменить защиту от изменения MLPA и FLPA путем использования значения NOPROT в системных параметрах MLPA и FIX.

IEASYSnn, раздел PARMLIB, содержит параметры и указатели, контролирующие направление начальной загрузки. См. Пример 16.6.

Пример 16.6. Фрагмент листинга раздела IEASYS00

```
File   Edit   Edit_Settings  Menu  Utilities  Compilers   Test  Help
-------------------------------------------------------------------
EDIT SYS1.PARMLIB(IEASYS00) - 01.68 Columns 00001 00072
Command ===> Scroll ===> CSR
****** **************************** Top of Data **********************
000001 ALLOC=00,
000002 APG=07,
000003 CLOCK=00,
000004 CLPA,
000005 CMB=(UNITR,COMM,GRAPH,CHRDR),
000006 CMD=(&CMDLIST1.),
000007 CON=00,
000008 COUPLE=00, WAS  FK
000009 CSA=(2M,128M),
000010 DEVSUP=00,
000011 DIAG=00,
000012 DUMP=DASD,
000013 FIX=00,
000014 GRS=STAR,
000015 GRSCNF=ML,
000016 GRSRNL=02,
000017 IOS=00,
000018 LNKAUTH=LNKLST,
000019 LOGCLS=L,
000020 LOGLMT=999999,
000021 LOGREC=SYS1.&SYSNAME..LOGREC,
000022 LPA=(00,L),
000023 MAXUSER=1000,
000024 MSTRJCL=00,
000025 NSYSLX=250,
000026 OMVS=&OMVSPARM.,
```

Для вывода информации о том, как была загружена система, можно использовать команду D IPLINFO, как показано в Примере 16.7.

Пример 16.7. Выходные данные команды D IPLINFO

```
D IPLINFO
     IEE254I 11.11.35 IPLINFO DISPLAY 906
     SYSTEM IPLED AT 10.53.04 ON 08/15/2005
     RELEASE z/OS 01.07.00 LICENSE = z/OS
     USED LOADS8 IN SYS0.IPLPARM ON C730
     ARCHLVL = 2 MTLSHARE = N
     IEASYM LIST = XX
     IEASYS LIST = (R3,65) (OP)
     IODF DEVICE C730
     IPL DEVICE 8603 VOLUME Z17RC1
```

Создание системных адресных пространств

Помимо инициализации системных областей z/OS устанавливает адресные пространства системных компонентов. Создается адресное пространство для главного планировщика и другие системные адресные пространства для различных подсистем и системных компонентов. К адресным пространствам компонентов относятся: *MASTER*, ALLOCAS, APPC, CATALOG, и т. д.

Инициализация главного планировщика

Программы инициализации главного планировщика инициализируют системные службы, в частности, системный журнал и коммуникационную задачу, и запускают сам главный планировщик. Они также вызывают создание системного адресного пространства для подсистемы ввода заданий (JES2 или JES3), после чего запускают подсистему ввода заданий.

Инициализация подсистем

Инициализация подсистем представляет собой процесс подготовки подсистемы к использованию в системе. Разделы IEFSSNxx библиотеки SYS1.PARMLIB содержат определения первичных подсистем, таких как JES2 или JES3, и вторичных подсистем, таких как NetView и DB2. Дополнительные сведения о данных, содержащихся в разделах IEFSSNxx для вторичных подсистем, см. в руководстве по установке для требуемой системы.

Во время инициализации системы заданные подсистемы инициализируются. Сначала нужно определить первичную подсистему (JES), так как другим подсистемам, таким как DB2, требуются функции первичной подсистемы в программах инициализации. Проблемы могут возникнуть, если подсистемы, использующие сервис привязки подсистем (subsystem affinity service) в своих программах инициализации, иници-

ализируются до первичной подсистемы. После инициализации первичной JES инициализация подсистем осуществляется в том порядке, в котором разделы IEFSSNxx PARMLIB заданы параметром SSN. Например, при SSN=(aa,bb) раздел IEFSSNaa библиотеки PARMLIB будет обрабатываться перед разделом IEFSSNbb.

Обработка START/LOGON/MOUNT

После инициализации системы и активации подсистемы ввода заданий можно передавать задания на обработку. Если задание активируется командой START (для пакетных заданий), LOGON (для заданий с разделением времени) или MOUNT, выделяется новое адресное пространство. Обратите внимание на то, что прежде чем использовать команду LOGON, оператор должен запустить VTAM и TSO, имеющие собственные адресные пространства.

На рис. 16.7 показаны некоторые важные системные адресные пространства, а также VTAM, CICS, TSO, пользователь TSO и инициатор пакетной обработки. Каждое адресное пространство по умолчанию имеет 2 Гб виртуальной памяти, независимо от того, работает ли система в 31-разрядном или в 64-разрядном режиме.

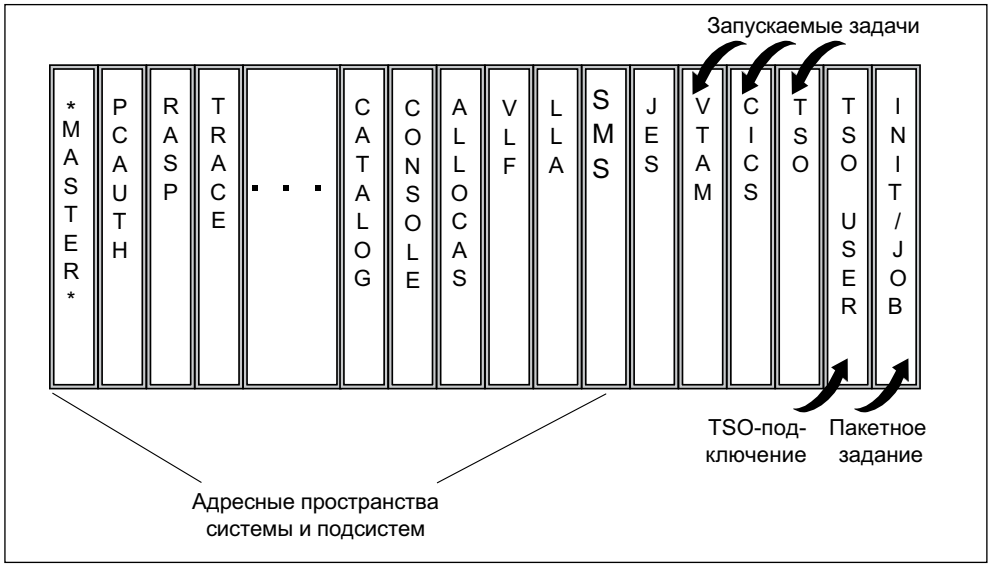


Рис. 16.7. Схема виртуальной памяти для нескольких адресных пространств

Каждое адресное пространство отображается, как показано на рис. 3.9 «Области памяти в адресном пространстве». Приватные области доступны только для этого адресного пространства, а общие области доступны для всех адресных пространств.

Во время инициализации системы z/OS, оператор использует системную консоль или консоль управления аппаратными средствами, подключенную к элементу поддержки. С системной консоли оператор инициализирует программу управления системой на этапе выполнения программы инициализации ядра (Nucleus Initialization Program, NIP).

На этапе NIP система может запросить у оператора предоставление системных параметров, контролирующих работу z/OS. Система также выдает информационные сообщения, информирующие оператора об этапах процесса инициализации.

16.8.2 Типы начальной загрузки

Существует несколько типов начальной загрузки (IPL); ниже они описаны подробно:

- «Холодный» запуск.
  Начальная загрузка, при которой происходит загрузка (или перезагрузка) PLPA и очистка страниц наборов данных VIO. Первая начальная загрузка после инициализации системы всегда представляет собой «холодный» запуск, так как изначально загружается PLPA. Последующие начальные загрузки являются «холодными» запусками, когда происходит перезагрузка PLPA, либо для изменения ее содержимого, либо для восстановления ее содержимого, если оно было утрачено. Обычно выполняется после внесения изменений в LPA (например при загрузке нового SYSRES-тома, содержащего данные обслуживания).
- Быстрый запуск.
  Начальная загрузка, при которой не происходит перезагрузка PLPA, но происходит очистка страниц наборов данных VIO (система сбрасывает таблицы сегментов и страниц таким образом, чтобы они соответствовали последней созданной области PLPA). Обычно выполняется, если не было внесено изменений в LPA, однако необходимо обновить VIO. Предотвращает «теплый» запуск заданий, использующих наборы данных VIO.
- «Теплый» запуск.
  Начальная загрузка, при которой не происходит перезагрузка PLPA и сохраняются страницы журналируемых наборов данных VIO . Это позволяет заданиям, выполнявшимся во время начальной загрузки, перезапуститься вместе с журналируемыми наборами данных VIO.

> **Примечание.** VIO представляет собой метод использования памяти таким образом, чтобы сохранить небольшие временные наборы данных для мгновенного доступа. Однако в отличие от RAM-диска на ПК, эти наборы данных действительно сохраняются на диске и могут использоваться при перезапуске. Очевидно, что таким способом не следует сохранять слишком много данных, поэтому их размер ограничен.

Часто предпочтительным методом является выполнение «холодного» запуска (с указанием CLPA). Можно использовать другие опции, однако следует быть очень внимательным, чтобы не допустить неожиданных изменений или отмены изменений. «Теплый» запуск можно использовать, когда имеются длительно выполняющиеся задания, которые требуется перезапустить после начальной загрузки, но альтернативный подход состоит в том, чтобы разделить задания на фрагменты меньшего размера, передающие реальные наборы данных, а не использовать VIO. Современные дисковые контроллеры с большим кешем в некоторой степени устранили необходимость длительного хранения наборов данных VIO.

Кроме того, не следует путать «холодный» запуск IPL (обычно используется термин CLPA, а не «“холодный” запуск») с «холодным» запуском JES. «Холодный» запуск JES выполняется в рабочей системе чрезвычайно редко, если вообще выполняется, и полностью уничтожает существующие данные в JES.

16.8.3 Завершение работы системы

Для того чтобы завершить работу системы, нужно поочередно в правильном порядке закрыть все задачи. В современных системах это является задачей пакета Automation (автоматизации). Для завершения работы системы обычно требуется одна команда. Это удалит большинство задач за исключением задачи Automation. Задача Automation закрывается вручную, после чего выполняются команды удаления системы из сисплекса или кольца синхронизации.

16.9 Заключение

Роль системного программиста z/OS состоит в установке, настройке и обслуживании операционной системы.

Системный программист должен (среди прочего) иметь представление о следующих вопросах:

- настройка системы;
- управление рабочей нагрузкой;
- производительность системы;
- конфигурирование устройств ввода-вывода;
- операции.

Для того чтобы максимально увеличить производительность задания извлечения модулей, операционная система z/OS была разработана таким образом, чтобы удерживать в памяти те модули, которые нужны для быстрого реагирования в операционной системе, а также для работы критически важных приложений. Область загрузки модулей (link pack area, LPA), linklist и авторизованные библиотеки являются основами процесса выборки (fetch).

Также в этой главе обсуждалась роль системного программиста в конфигурировании консолей и настройке автоматизации на основе сообщений.

Рассматривался запуск или начальная загрузка системы, а также следующие вопросы:

- начальная загрузка и процесс инициализации;
- типы начальной загрузки: «холодный» запуск, быстрый запуск и «теплый» запуск;
- причины для перезагрузки.

Основные термины в этой главе

HCD	IODF	SYSRES
SMP/E	linklist	IPL
WTOR	PARMLIB	PROCLIB
Системные символы	PSA	LPA
Ядро (nucleus)	LOADPARM	SQA

16.10 Контрольные вопросы

Чтобы проверить понимание материала этой главы, ответьте на следующие вопросы:

1. В Примере 16.2 предположим, что класс, назначенный определенному заданию, имеет заданную по умолчанию сцепленную PROCLIB для PROC00. Этому заданию нужна процедура, которая находится в SYS1.OTHERPRO. Что можно сделать для выполнения этого задания? В каких библиотеках процедур можно было бы выполнять поиск, если ничего не было сделано?
2. Почему консольные операции часто автоматизируют?
3. Почему структура сообщений и команд хорошо подходит для автоматизации?
4. Почему необходимо выполнять перезагрузки системы?
5. Какие три типа перезагрузки существуют, и в чем различие между ними?

16.11 Темы для дальнейшего обсуждения

1. Одна из причин, почему мэйнфрейм считается безопасной средой, связана с тем, что он не позволяет выполнять простое подключение устройств, но использовать можно только те устройства, которые программно определил системный программист. Считаете ли вы это правильным?
2. Сравните описание в разделе «Порядок поиска для программ» с путями поиска, используемыми в других операционных системах.
3. Обсудите следующее утверждение в отношении z/OS и других операционных систем, которые вам знакомы: основная цель системного программиста состоит в том, чтобы избежать перезагрузки системы.

16.12 Упражнения

1. Определите, какие разделы IEASYSxx использовались при текущей начальной загрузке. Задал ли оператор суффикс альтернативного IEASYSxx?
2. Задал ли оператор какой-либо параметр в ответ на сообщение `SPECIFY SYSTEM PARAMETERS`? Если задан ответ `Y`, найдите соответствующие разделы PARMLIB для этого параметра и определите значение параметра, которое было бы активным, если бы не был задан этот ответ оператора.
3. Выполните следующие действия:
   а) В своей системе найдите адрес загрузочного устройства (IPL device address) и загрузочный том (IPL Volume). Перейдите в SDSF, введите ULOG, после чего введите /D IPLINFO.
   б) Что такое адрес устройства IODF?
   в) Какой раздел LOADxx использовался при начальной загрузке? Какой набор данных содержит этот раздел LOADxx?
   г) Просмотрите этот раздел; какое имя имеет системный каталог, используемый системой?
   д) Какое имя имеет используемый набор данных IODF? Введите /D IOS,CONFIG.
   е) Системные параметры могут поступать с нескольких наборов данных PARMLIB. Введите /D PARMLIB. Какие наборы данных PARMLIB используются в вашей системе?

17

Использование SMP/E

Цель. Как системный программист z/OS, вы будете отвечать за обеспечение правильной установки всех программных продуктов и их модификаций в системе. Вам также необходимо будет обеспечить установку требуемых версий всех продуктов, чтобы различные элементы системы могли работать совместно. На первый взгляд это может показаться не очень сложной задачей, однако с ростом сложности конфигурации программного обеспечения увеличивается и сложность мониторинга всех элементов системы.

Основным средством установки и обновления программного обеспечения в системе z/OS является SMP/E. SMP/E консолидирует данные установки, обеспечивает высокую гибкость выбора устанавливаемых изменений, обеспечивает диалоговый интерфейс и поддерживает динамическое распределение наборов данных.

После завершения работы над этой главой вы сможете объяснить:

- что такое SMP/E;
- что такое модификации системы;
- какие наборы данных используются в SMP/E.
- каким образом SMP/E может помочь в установке и обслуживании продуктов и в мониторинге изменений в продуктах.

17.1 Что такое SMP/E?

SMP/E представляет собой инструмент операционной системы z/OS, предназначенный для управления установкой программных продуктов в системе z/OS и для отслеживания модификаций этих продуктов. SMP/E контролирует эти изменения на уровне компонентов, используя следующие методы:

- Выбор правильных версий кода для установки из большого количества потенциальных изменений.
- Вызов системных утилит для установки изменений.
- Ведение учета установленных изменений с использованием средства, позволяющего запрашивать состояние программного обеспечения и, при необходимости, отменять изменение.

Сведения обо всем коде и его модификации расположены в базе данных SMP/E, называемой консолидированной базой данных состава программного обеспечения (consolidated software inventory – CSI) и содержащей один или несколько наборов данных VSAM.

SMP/E можно запустить либо используя пакетные задания, либо используя диалоги в ISPF/PDF. Используя диалоги SMP/E, можно выполнять интерактивные запросы к базе данных SMP/E, а также создавать и передавать задания на обработки команд SMP/E. Основные команды работы с SMP/E рассматриваются в разделе 17.11, «Работа с SMP/E».

Дополнительные сведения. Дополнительные сведения о SMP/E см. в публикации *SMP/E User's Guide*. Эту и другие публикации IBM можно найти на веб-сайте z/OS Internet Library:
`http://www.ibm.com/servers/eserver/zseries/zos/bkserv/`

17.2 Место SMP/E в системе

Может показаться, что система z/OS представляет собой один большой блок кода, управляющий центральным процессором. В действительности, z/OS представляет собой сложную систему, содержащую множество небольших блоков кода. Каждый из этих небольших блоков кода выполняет в системе определенную функцию (рис. 17.1).

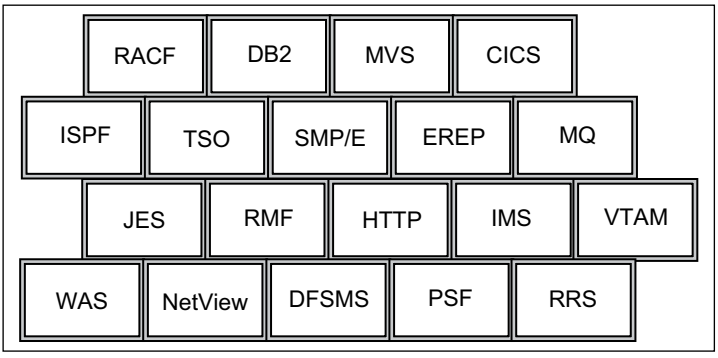


Рис. 17.1. Система с точки зрения SMP/E

Например, система z/OS может содержать следующие функции:

- базовая управляющая программа (Base Control Program, BCP);
- CICS;
- DFSMS;
- HTTP Server;
- ISPF;
- JES2 или JES3;
- OSA/SF (Open Systems Adapter/Support Facility);
- RMF (Resource Measurement Facility);
- SDSF (System Display and Search Facility);
- SMP/E;
- TSO/E (Time Sharing Option/Extensions);
- WebSphere MQ;
- z/OS UNIX System Services (z/OS UNIX).

Каждая системная функция состоит из одного или нескольких загрузочных модулей. В среде z/OS загрузочный модуль представляет базовый модуль машинного исполняемого кода. Загрузочные модули создаются посредством сочетания одного или нескольких объектных модулей и их обработки утилитой компоновки. Компоновка модулей представляет собой процесс разрешения внешних ссылок и адресов. Поэтому функции системы представлены одним или несколькими объектными модулями, которые были скомбинированы и скомпонованы.

Для того чтобы увидеть, откуда берется объектный модуль, рассмотрим пример на рис. 17.2.

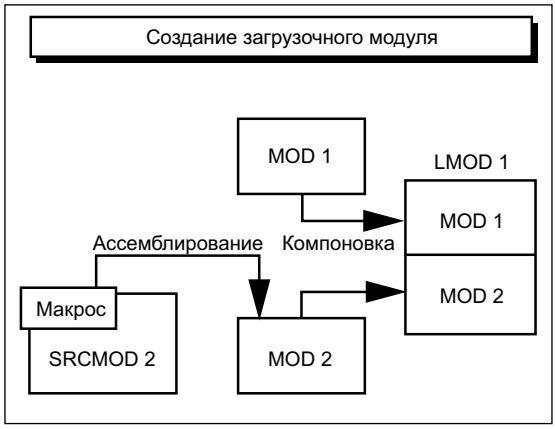


Рис. 17.2. Создание загрузочного модуля

Чаще всего объектные модули идут в составе продукта. В данном примере объектный модуль MOD1 поставлялся в составе продукта. В других случаях может потребоваться выполнить ассемблирование исходного кода, переданного поставщиками

продукта, для создания объектного модуля. Можно изменить исходный код и затем выполнить его ассемблирование, чтобы создать объектный модуль. В данном примере SRCMOD2 является исходным кодом, который нужно ассемблировать для создания объектного модуля MOD2. После ассемблирования необходимо выполнить компоновку объектного модуля MOD2 и объектного модуля MOD1, в результате чего получается загрузочный модуль LMOD1.

Помимо объектных модулей и исходного кода, большинство продуктов содержат множество других частей, например, макросов, справочных панелей, командных списков и прочих разделов библиотек z/OS. Эти модули, макросы и другие типы данных и кода являются основными строительными блоками системы. Все эти строительные блоки называются *элементами*.

Элементы связаны и зависят от других продуктов и служб, которые могут быть установлены в той же системе z/OS. Они описывают связь программного обеспечения с другими продуктами и службами, которые могут быть установлены в той же системе z/OS.

17.3 Изменение элементов системы

Рано или поздно наступает момент, когда нужно внести изменения в программное обеспечение системы z/OS. Эти изменения могут быть необходимы для улучшения практичности или надежности продукта. По разным причинам может потребоваться добавить в систему некоторые новые функции, обновить некоторые элементы системы или изменить некоторые элементы. Программное обеспечение, как программные продукты, так и службы, состоят из таких элементов, как макросы, модули, исходный код и прочие типы данных (например, CLIST или образцы процедур).

17.3.1 Что такое SYSMOD?

SMP/E может устанавливать различные системные обновления, если они упакованы как системные модификации (system modification, *SYSMOD*). SYSMOD представляет собой пакет элементов и управляющей информации, который SMP/E должен установить и отслеживать системные изменения.

SYSMOD – входные данные для SMP/E, определяющие внедрение, замену или обновление элементов в z/OS

SYSMOD представляют собой сочетание элементов и управляющей информации. Они состоят из двух частей:

- Управляющие операторы модификации (modification control statements, MCS), выделенные первыми двумя символами ++, сообщающие SMP/E:.
  - какие элементы обновляются или заменяются;
  - какая связь между SYSMOD и программным продуктом и другими SYSMOD;
  - прочую информацию об установке.
- Текст модификации, описывающий объектные модули, макросы и прочие элементы, содержащиеся в SYSMOD.

17.3.2 Типы SYSMOD

Существует четыре различных категории SYSMOD, каждая из которых поддерживает определенную задачу, которую может потребоваться выполнить.

FUNCTION Этот тип SYSMOD внедряет в систему новый продукт, новую версию продукта или обновленные функции существующего продукта.

PTF Временное программное исправление (program temporary fix, PTF) представляет собой исправление заявленной проблемы, поставляемое компанией IBM. Они предназначены для установки во всех средах. PTF можно использовать как профилактические средство для недопущения возникновения некоторых известных проблем, которые еще не появились в вашей системе, или как корректирующее средство для исправления уже возникших проблем. Установке PTF всегда должна предшествовать установка функционального (FUNCTION) SYSMOD, а часто и других PTF.

APAR Авторизованный отчет об анализе программы (authorized program analysis report, APAR) представляет собой временное исправление, предназначенное для корректирования или обхода проблемы для первого сообщившего о проблеме. APAR может быть неприменим в определенной среде. Установке APAR всегда должна предшествовать установка функционального SYSMOD, а часто и определенного PTF. Другими словами, APAR предназначен для установки на элементе с определенным сервисным уровнем.

USERMOD Этот тип SYSMOD создается пользователем, либо для изменения кода, поставляемого IBM, либо для добавления независимых функций в систему. Установке USERMOD всегда должна предшествовать установка функционального SYSMOD, а иногда и некоторых PTF, APAR или других USERMOD.

SMP/E отслеживает функциональные и сервисные уровни каждого элемента и использует эту иерархию SYSMOD, например, для определения того, какие функциональные и сервисные уровни элемента необходимо установить, а также для определения правильного порядка установки обновлений элементов.

17.4 Внедрение элемента в систему

Один из способов модификации системы состоит во внедрении новых элементов в систему. Чтобы выполнить внедрение элементов через SMP/E, можно установить функциональный SYSMOD. Функциональный SYSMOD внедряет новый продукт, новую версию или релиз продукта, обновленные функции для существующего продукта в системе. Все остальные типы SYSMOD зависят от функционального SYSMOD, так как все они выполняют модификацию элементов изначально внедренных функциональным SYSMOD.

Под установкой функционального SYSMOD подразумевается размещение всех элементов продукта в системные наборы данных или библиотеки. Примерами библиотек являются SYS1.LPALIB, SYS1.MIGLIB и SYS1.SVCLIB.

На рис. 17.3 представлен процесс создания исполняемого кода в библиотеках рабочей системы.

На рис. 17.3 показано, как при установке функционального SYSMOD выполняется компоновка объектных модулей MOD1, MOD2, MOD3 и MOD4 для создания загрузочного модуля LMOD2. Исполняемый код, созданный в загрузочном модуле LMOD2, устанавливается в системных библиотеках путем инсталляции функционального SYSMOD.

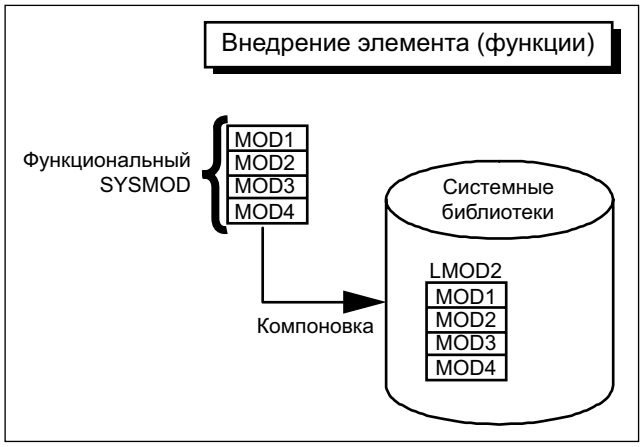


Рис. 17.3. Внедрение элемента

Существует два типа функциональных SYSMOD.

- *Базовый (base)* функциональный SYSMOD добавляет или заменяет системную функцию целиком. Примерами базовых функций являются SMP/E и JES2.
- *Зависимый (dependent)* функциональный SYSMOD осуществляет добавление к существующей системной функции. Он называется зависимым, так как его установка зависит от уже установленной базовой функции. Примерами зависимых функций являются языковые возможности SMP/E.

Как базовые функциональные SYSMOD, так и зависимые функциональные SYSMOD используются для внедрения новых элементов в систему. На рис. 17.4 представлен пример простого функционального SYSMOD, внедряющего четыре элемента.

```
++FUNCTION(FUN0001)          /* SYSMOD type and identifier.   */.
++VER(Z038)                  /* For an OS/390 system          */.
++MOD(MOD1)  RELFILE(1)      /* Introduce this module         */
             DISTLIB(AOSFB)  /* in this distribution library. */.
++MOD(MOD2)  RELFILE(1)      /* Introduce this module         */
             DISTLIB(AOSFB)  /* in this distribution library. */.
++MOD(MOD3)  RELFILE(1)      /* Introduce this module         */
             DISTLIB(AOSFB)  /* in this distribution library. */.
++MOD(MOD4)  RELFILE(1)      /* Introduce this module         */
             DISTLIB(AOSFB)  /* in this distribution library. */.
```

Рис. 17.4. Пример простого функционального SYSMOD

17.5 Недопущение или исправление проблем с элементом

При обнаружении проблемы с программным элементом IBM предоставляет своим клиентам протестированное исправление для этой проблемы. Это исправление поставляется в формате временного программного исправления (PTF). Хотя вы могли

и не сталкиваться с проблемой, которую исправляет PTF, лучше установить PTF в своей системе. PTF SYSMOD используется для установки PTF и предотвращения возникновения проблемы в системе.

Обычно PTF предназначены для полной замены или обновления одного или нескольких элементов системной функции (рис. 17.5).

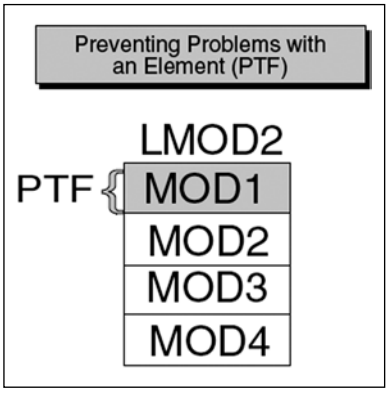


Рис. 17.5. Предотвращение возникновения проблем с элементом

На рис. 17.5 представлен ранее установленный загрузочный модуль LMOD2. Если требуется заменить элемент MOD1, нужно установить PTF SYSMOD, содержащий модуль MOD1. Этот PTF SYSMOD заменяет элемент, содержащий ошибку, на исправленный элемент. При установке PTF SYSMOD, SMP/E выполняет перекомпоновку LMOD2, чтобы включить в него новую исправленную версию MOD1.

На рис. 17.6 представлен пример простого PTF SYSMOD.

```
++PTF(PTF0001)              /* SYSMOD type and identifier.   */.
++VER(Z038) FMID(FUN0001)   /* Apply to this product.        */.
++MOD(MOD1)                 /* Replace this module           */
           DISTLIB(AOSFB)   /* in this distribution library. */.
...
... object code for module
...
```

Рис. 17.6. Пример простого PTF SYSMOD

PTF SYSMOD всегда зависят от установки функционального SYSMOD. Иногда некоторые PTF SYSMOD могут также быть зависимыми от установки других PTF SYSMOD. Такие зависимости называются *предварительными условиями* (*prerequisites*). Мы рассмотрим предварительные условия для PTF при обсуждении сложности отслеживания элементов системы.

17.6 Исправление проблем с элементом

APAR – временное исправление ошибки в системной управляющей программе IBM или в лицензированной программе, выполняющееся для определенного пользователя

Иногда возникает необходимость исправить серьезную проблему, возникшую в системе, до того как будет доступен PTF. В такой ситуации IBM предоставляет авторизованный отчет об анализе программы (authorized program analysis report APAR). APAR представляет собой исправление, предназначенное для быстрой коррекции определенной области элемента или замены ошибочного элемента. APAR SYSMOD устанавливается для внедрения исправления, обновляя, таким образом, ошибочный элемент.

На рис. 17.7 серым цветом выделена область MOD2, содержащая ошибку.

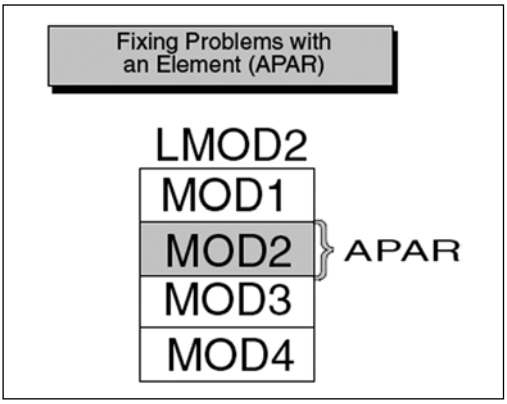


Рис. 17.7. Исправление проблем с элементом

Обработка APAR SYSMOD обеспечивает модификацию объектного модуля MOD2. Во время установки APAR SYSMOD происходит обновление (и исправление) MOD2 в загрузочном модуле LMOD2.

На рис. 17.8 представлен пример простого APAR SYSMOD.

```
++APAR(APAR001)             /* SYSMOD type and identifier.   */.
++VER(Z038) FMID(FUN0001)   /* Apply to this product         */
            PRE(UZ00004)    /* at this service level.        */.
++ZAP(MOD2)                 /* Update this module            */
            DISTLIB(AOSFB)  /* in this distribution library. */.
...
... zap control statements
...
```

Рис. 17.8. Пример простого APAR SYSMOD

Предварительным условием для APAR SYSMOD всегда является функциональный SYSMOD; кроме того, APAR SYSMOD может зависеть от установки других PTF или APAR.

17.7 Настройка элемента – USERMOD SYSMOD

Если требуется изменить функциональность продукта, можно выполнить настройку этого элемента системы. IBM предоставляет некоторые модули, которые позволяют настраивать код IBM для соответствия определенным требованиям. После внесения требуемых изменений выполняется добавление модулей в систему путем установки USERMOD SYSMOD. Этот SYSMOD можно использовать для замены или обновления элемента, либо для внедрения совершенно нового пользовательского элемента в систему. В любом случае USERMOD SYSMOD разрабатывается либо для изменения кода IBM, либо для добавления собственного кода в систему.

На рис. 17.9 MOD3 был обновлен посредством установки USERMOD SYSMOD.

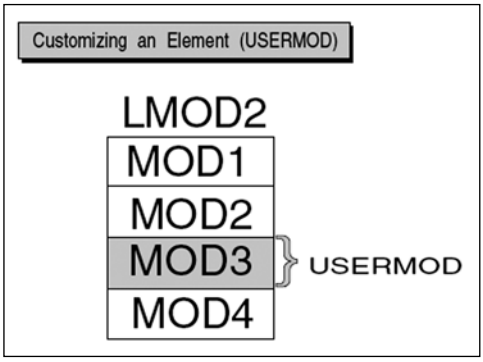


Рис. 17.9. Настройка элемента

На рис. 17.10 представлен пример простого USERMOD SYSMOD.

```
++USERMOD(USRMOD1)              /* SYSMOD type and identifier.   */.
++VER(Z038) FMID(FUN0001)       /* Apply to this product         */
            PRE(UZ00004)        /* at this service level.        */.
++SRCUPD(JESMOD3)               /* Update this source module     */
            DISTLIB(AOSFB)      /* in this distribution library. */.
...
... update control statements
...
```

Рис. 17.10. Пример простого USERMOD SYSMOD

Для USERMOD SYSMOD предварительными условиями является установка функционального SYSMOD и, возможно, установка других PTF, APAR или USERMOD SYSMOD.

17.7.1 Предварительные условия и сопутствующие условия SYSMOD

Как мы говорили, для PTF, APAR и USERMOD SYSMOD предварительным условием является функциональный SYSMOD. Помимо их зависимостей от функционального SYSMOD:

- PTF SYSMOD могут зависеть от других PTF SYSMOD.
- APAR SYSMOD могут зависеть от PTF SYSMOD и других APAR SYSMOD.
- USERMOD SYSMOD могут зависеть от PTF SYSMOD, APAR SYSMOD и других USERMOD SYSMOD.

Иногда PTF или даже APAR взаимно зависят от других PTF SYSMOD, называемых *сопутствующими условиями* (*corequisites*).

Рассмотрим сложность этих зависимостей. Если умножить сложность взаимосвязей на сотни загрузочных модулей в десятках библиотек, необходимость инструмента вроде SMP/E становится очевидной.

Рассмотрим воздействие этих связей на обслуживание программного обеспечения в среде z/OS.

17.8 Отслеживание элементов системы

Важность слежения за элементами системы и их модификациями становится очевидной при рассмотрении процесса обслуживания z/OS. Часто PTF выполняет замену множества элементов.

В примере, представленном на рис. 17.11, PTF1 содержит замены для двух модулей, MOD1 и MOD2. И хотя загрузочный модуль LMOD2 содержит четыре модуля, выполняется замена только двух из этих модулей.

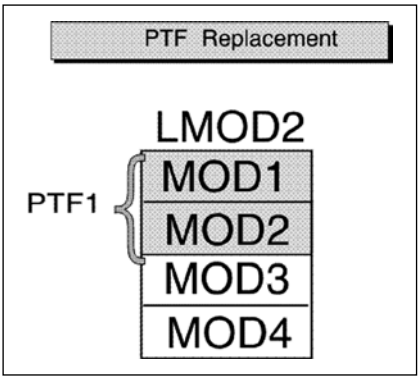


Рис. 17.11. Замена PTF

Но что произойдет, если второй PTF заменит некоторый код в модуле, который был заменен PTF1? Посмотрим на рис. 17.12.

В этом примере PTF2 содержит замены для MOD2 и MOD3. Для того чтобы MOD1, MOD2 и MOD3 могли успешно взаимодействовать, PTF1 должен быть установлен до PTF2. Поэтому MOD3, поставляемый в PTF2, может зависеть от MOD1, входящего в PTF1. Эта зависимость и составляет предварительное условие. Предварительные условия SYSMOD определяются в разделе управляющих операторов модификации (modification control statements, MCS) пакета SYSMOD, рассмотренных в разделе «Изменение элементов системы».

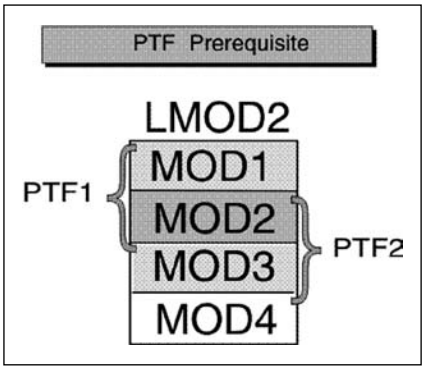


Рис. 17.12. Предварительное условие для PTF

Помимо отслеживания предварительных условий существует еще одна важная причина для отслеживания системных элементов. Один и тот же модуль часто может входить в разные загрузочные модули. Рассмотрим пример на рис. 17.13.

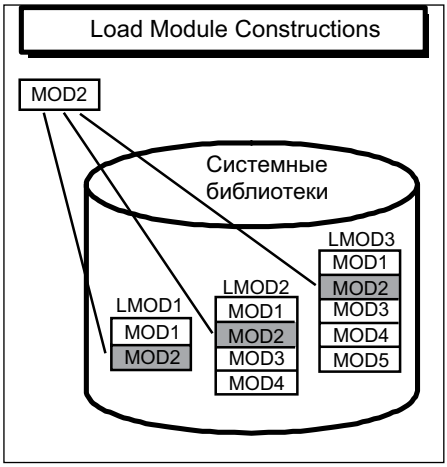


Рис. 17.13. Содержание загрузочных модулей

На рис. 17.13 один и тот же модуль MOD2 входит в LMOD1, LMOD2 и LMOD3. При внедрении PTF, заменяющего элемент MOD2, этот модуль необходимо заменить во всех загрузочных модулях, в которых он есть. Поэтому важно отслеживать все загрузочные модули и входящие в них модули.

Теперь можно оценить, насколько сложным может быть отслеживание системных элементов и их уровней модификации. Далее мы кратко рассмотрим использование возможностей отслеживания SMP/E.

17.9 Отслеживание и управление предварительными условиями

Для успешного отслеживания и управления элементами все элементы и их модификации должны быть четко определены в SMP/E. Для этого в SMP/E используются идентификаторы модификации. Существует три вида идентификаторов модификации, связанных с каждым элементом.

- *Идентификаторы функциональных изменений* (*Function Modification Identifiers*, FMID) идентифицируют функциональный SYSMOD, внедряющий элемент в систему.
- *Идентификаторы изменений замены* (*Replacement Modification Identifiers*, RMID) идентифицируют последний SYSMOD (в большинстве случаев, PTF SYSMOD) для замены элемента.
- *Идентификаторы изменений обновления* (*Update Modification Identifiers*, UMID) идентифицируют SYSMOD, представляющий обновление элемента со времени его последней замены.

SMP/E использует эти идентификаторы модификаций для отслеживания всех SYSMOD, установленных в системе. Это обеспечивает правильную последовательность их установки. Теперь, когда мы рассмотрели необходимость отслеживания элементов, и знаем, что отслеживает SMP/E, перейдем к рассмотрению того, каким образом SMP/E выполняет функцию отслеживания.

17.10 Как работает SMP/E?

Давайте обсудим установку функций в системе. Начнем с элементов, таких как модули, макросы и исходный код. Эти элементы обрабатываются такими утилитами, как ассемблер или компоновщик, для создания загрузочных модулей. Загрузочные модули содержат машинный исполняемый код.

Рабочая система в среде z/OS содержит операционную систему z/OS и весь код, необходимый для выполнения повседневной работы. Это понятно, но где же все хранится, и как все организовано? Давайте посмотрим.

17.10.1 Дистрибутивные библиотеки и целевые библиотеки

Для корректного выполнения обработки, SMP/E должен обрабатывать множество информации о структуре, содержимом и состоянии модификации управляемого им программного обеспечения. Всю информацию, обрабатываемую в SMP/E, можно рассматривать, как если бы вся эта информация находилась в общественной библиотеке.

В общественной библиотеке можно увидеть полки, заполненные книгами, и картотеку с ящиками, содержащими карточки для каждой книги в библиотеке. Эти карточки содержат информацию о заголовке, авторе, датах публикации, типе книги и указателе для книги на полке.

В среде SMP/E существует два разных типа «книжных полок». Они называются дистрибутивными библиотеками и целевыми библиотеками. Так же как книжные полки в общественной библиотеке содержат библиотечные книги, дистрибутивные библиотеки и целевые библиотеки содержат элементы системы.

Дистрибутивные библиотеки (*distribution libraries*) содержат все элементы, такие как модули и макросы, используемые как входные данные для работы системы. Одним очень важным предназначением дистрибутивных библиотек является резервное копирование. При возникновении серьезной ошибки с элементом в рабочей системе, его можно заменить стабильным элементом из дистрибутивной библиотеки.

Целевые библиотеки (*target libraries*) содержат исполняемый код, необходимый для работы системы.

17.10.2 Консолидированная база данных состава программного обеспечения (CSI)

Рассматривая аналогию с общественной библиотекой, можно увидеть, что мы не рассмотрели один важный фрагмент. Общественная библиотека содержит картотеку, помогающую найти книгу или требуемую информацию. SMP/E содержит тот же тип механизма отслеживания в виде консолидированной базы данных состава программного обеспечения (*consolidated software inventory* – CSI).

CSI – набор данных SMP/E, содержащий информацию о структуре пользовательской системы

Наборы данных CSI содержат всю информацию, необходимую SMP/E для отслеживания дистрибутивных и целевых библиотек. Так же как картотека содержит карточку для каждой книги в библиотеке, CSI содержит запись для каждого элемента в своих библиотеках. Записи CSI содержат имя элемента, его тип, историю, сведения о внедрении элемента в систему и указатель на элемент в дистрибутивных и целевых библиотеках. CSI содержит не сам элемент, а только описание представляемого элемента.

Посмотрим, как эти записи организованы в CSI.

Зоны SMP/E

Карточки в картотеке общественной библиотеки организованы в алфавитном порядке по фамилии автора, а также по теме и заголовку книги. В CSI записи элементов в дистрибутивных библиотеках и целевых библиотеках группируются в соответствии с их состоянием установки. Другими словами, записи, представляющие элементы из дистрибутивных библиотек, находятся в дистрибутивной зоне. Записи, представляющие элементы из целевых библиотек, находятся в целевой зоне. Обе эти зоны имеют такое же назначение, как и ящики в картотеке общественной библиотеки.

Помимо дистрибутивной зоны и целевой зоны, SMP/E CSI содержит также глобальную зону. На рис. 17.14 представлена связь между зонами SMP/E и библиотеками.

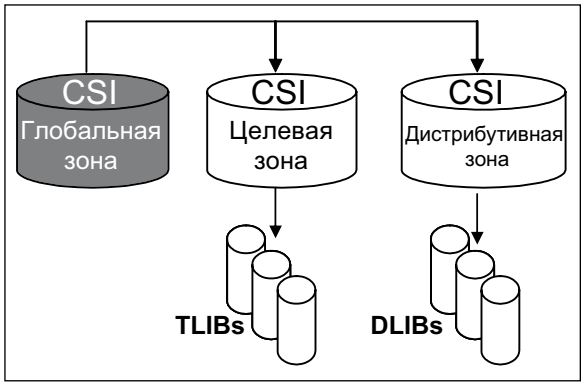


Рис. 17.14. Связь между зонами SMP/E и библиотеками

Глобальная зона содержит:

- записи, необходимые для идентификации и описания каждой целевой и дистрибутивной зоны в SMP/E;

- информацию об опциях обработки SMP/E;
- информацию о статусе всех SYSMOD, обработка которых была начата в SMP/E;
- исключения для SYSMOD, требующие специальной обработки или содержащие ошибки.

В SMP/E, когда говорится об исключениях, обычно подразумеваются HOLDDATA. HOLDDATA часто поставляются для продукта, чтобы указать, какой SYSMOD не следует устанавливать. Это может иметь следующие причины:

- PTF содержит ошибку, и его не следует устанавливать до тех пор, пока ошибка не будет исправлена (ERROR HOLD).
- Перед установкой SYSMOD может быть необходимо выполнить некоторые системные действия (SYSTEM HOLD).
- Пользователю может требоваться выполнить некоторые действия перед установкой SYSMOD (USER HOLD).

Теперь вы имеете представление о согласовании всех элементов системы, их установке, модификации и отслеживании с использованием SMP/E.

17.11 Работа с SMP/E

После ознакомления с SMP/E и его возможностями, вы можете спросить, что нужно знать для того, чтобы начать работать с SMP/E. Вся обработка в SMP/E выполняется с применением трех простых команд: RECEIVE, APPLY и ACCCEPT. Ниже мы рассмотрим эти команды.

17.11.1 Использование команды RECEIVE

Команда RECEIVE позволяет получить SYSMOD, находящийся вне SMP/E, и добавить его в библиотечный домен SMP/E, в результате чего начинается построение описывающих его записей CSI. Это позволяет осуществлять запросы ввода в последующих процессах. В последнее время в качестве источников SYSMOD часто используются веб-сайты, хотя обычно он копируется с магнитной ленты или с носителя стороннего производителя.

Этот процесс выполняет несколько задач (рис. 17.15):

- Построение записей в глобальной зоне для описания SYSMOD.
- Проверка правильности SYSMOD, в частности проверка синтаксиса управляющих операторов модификации (MCS), связанных с продуктами, установленными в CSI.
- Установка SYSMOD в библиотеках. Пример: библиотека временного хранения PTF.
- Доступ к HOLDDATA, чтобы убедиться в том, что не будет выполнено внедрение ошибок.

Во время выполнения команды RECEIVE выполняется копирование MCS для каждого SYSMOD в область временного хранения SMP/E, представляющую собой набор данных SMPPTS, содержащий замену или обновление элемента для данного SYSMOD.

Существуют также RELFILE, в которых упаковываны элементы в виде «относительных файлов» отдельно от MCS, используемых, главным образом, функциональными SYSMOD. Относительные файлы хранятся в другой области временного хранения, называемой наборами данных SMPTLIB.

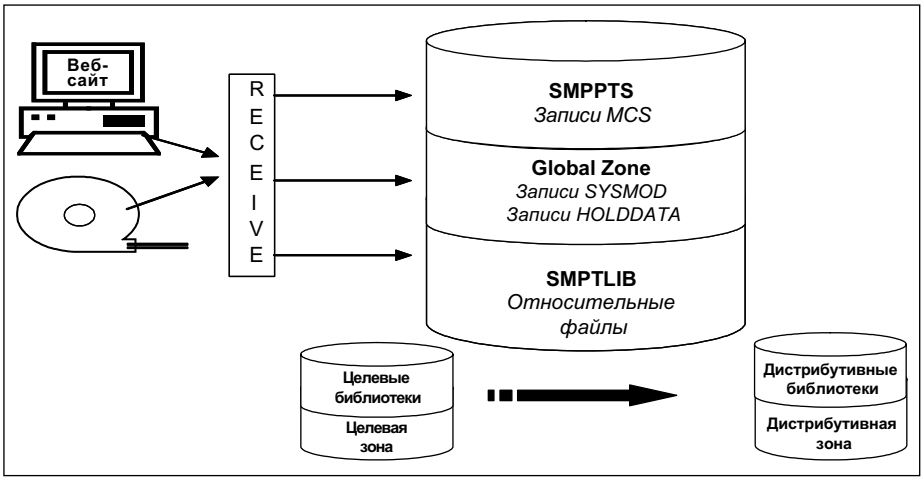


Рис. 17.15. Обработка команды RECEIVE в SMP/E

SMP/E обновляет глобальную зону информацией о полученных SYSMOD:

При обслуживании системы требуется выполнить установку исправлений и обработку связанных HOLDDATA. Например, предположим, что IBM предоставила вам магнитную ленту с исправлениями (например, магнитную ленту CBPDO или ESO), и вам требуется установить ее в системе. Первым делом нужно получить SYSMOD и HOLDDATA, содержащиеся на ленте, путем ввода следующих команд:

```
SET BDY(GLOBAL).
RECEIVE.
```

После ввода этих команд, SMP/E получает все SYSMOD и HOLDDATA с магнитной ленты.

Примеры команд RECEIVE

Для того чтобы получить только HOLDDATA, требующие специальной обработки или содержащие информацию об ошибках, используется следующая команда:

```
SET BDY(GLOBAL).
RECEIVE HOLDDATA.
```

Для получения только SYSMOD, предназначенных для установки в глобальной зоне, используется следующая команда:

```
SET BDY(GLOBAL).
RECEIVE SYSMODS.
```

Для получения всех SYSMOD, включая HOLDDATA, для определенного продукта (например, для WebSphere Application Server), используется следующая команда:

```
SET BDY(GLOBAL).
RECEIVE FORFMID(H28W500).
```

17.11.2 Использование команды APPLY

Команда APPLY определяет, какие из полученных SYSMOD нужно выбрать для установки в целевых библиотеках (target libraries, TLIB). SMP/E также проверяет, установлены ли все требуемые SYSMOD (предварительные условия), или устанавливаются ли они параллельно и в правильной последовательности. Источником элементов являются наборы данных SMPTLIB, набор данных SMPPTS или отдельные библиотеки, в зависимости от того, как была выполнена упаковка. На этом этапе обработки SMP/E выполняются следующие действия:

- Запуск требуемой утилиты для установки SYSMOD в целевой библиотеке в зависимости от типа входного текста и изменяемого целевого модуля.
- Обеспечение корректной связи нового SYSMOD с другими SYSMOD в целевой зоне.
- Модификация CSI с отображением обновленных модулей.

Команда APPLY обновляет системные библиотеки, и ее следует осторожно использовать в реальной рабочей системе. Рекомендуется сначала использовать копию рабочих целевых библиотек и зон.

Целевая зона отражает содержимое целевых библиотек. Поэтому после завершения работы утилиты и обновления зоны она точно отражает состояние этих библиотек.

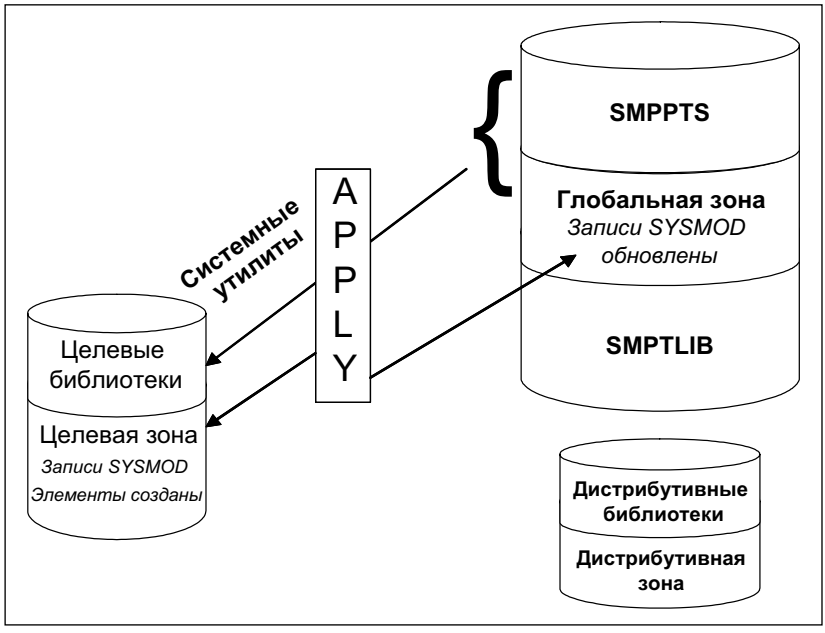


Рис. 17.16. Обработка команды APPLY в SMP/E

Обновление целевой зоны происходит при обработке команды APPLY (рис. 17.16):

- Все записи SYSMOD в глобальной зоне обновляются таким образом, чтобы отразить применение SYSMOD к целевой зоне.
- Целевая зона точно отражает каждую примененную запись SYSMOD. Также в целевой зоне создаются записи элементов (такие как MOD и LMOD).
- Записи BACKUP создаются в наборе данных SMPSCDS, чтобы можно было выполнить восстановление SYSMOD, если это будет необходимо.

Подобно процессу RECEIVE, команда APPLY имеет множество разных операндов, обеспечивающих гибкость выбора SYSMOD, которые требуется установить в целевых библиотеках, и обеспечивает разнообразные выходные данные. Используемые директивы указывают SMP/E, что нужно установить.

Чтобы установить только PTF SYSMOD, введите команду, подобную следующей:

```
SET BDY(ZOSTGT1).
APPLY PTFS.
```

Чтобы выбрать PTF SYSMOD, укажите их в директивах, например:

```
SET BDY(ZOSTGT1).
APPLY SELECT(UZ00001, UZ00002).
```

Иногда может потребоваться установить только корректирующие исправления (APAR) или пользовательские модификации (USERMOD) в целевую библиотеку, например:

```
SET BDY(ZOSTGT1).
APPLY APARS
USERMODS.
```

В других случаях, может потребоваться обновить выбранный продукт с дистрибутивной ленты:

```
SET BDY(ZOSTGT1).
APPLY PTFS
FORFMID(H28W500).
```

или

```
SET BDY (ZOSTGT1).
APPLY FORFMID(H28W500).
```

В этих двух примерах, SMP/E применяет все применимые PTF для FMID. Если не указано иное, PTF является используемым по умолчанию типом SYSMOD.

Использование APPLY CHECK

Иногда может требоваться просмотреть, какие SYSMOD есть в наличии, прежде чем их устанавливать. Для этого следует включить операнд CHECK в команды, например, следующим образом:

```
SET BDY(MVSTGT1).
APPLY PTFS
APARS
FORFMID(HOP1)
GROUPEXTEND CHECK.
```

После выполнения этих команд, можно просмотреть отчет о статусе SYSMOD, какие SYSMOD были бы установлены, если бы не был задан операнд CHECK. Если вас устраивают результаты проверки, можно ввести команды повторно без операнда CHECK, чтобы действительно установить SYSMOD.

17.11.3 Использование команды ACCEPT

После установки и тестирования SYSMOD в целевой библиотеке изменения принимаются командой ACCEPT. Она устанавливает выбранные SYSMOD в соответствующих дистрибутивных библиотеках.

В команде ACCEPT указываются операнды, сообщающие, какой из полученных SYSMOD нужно выбрать для установки. На этом этапе SMP/E также убеждается в том, что выбран правильный функциональный уровень каждого элемента.

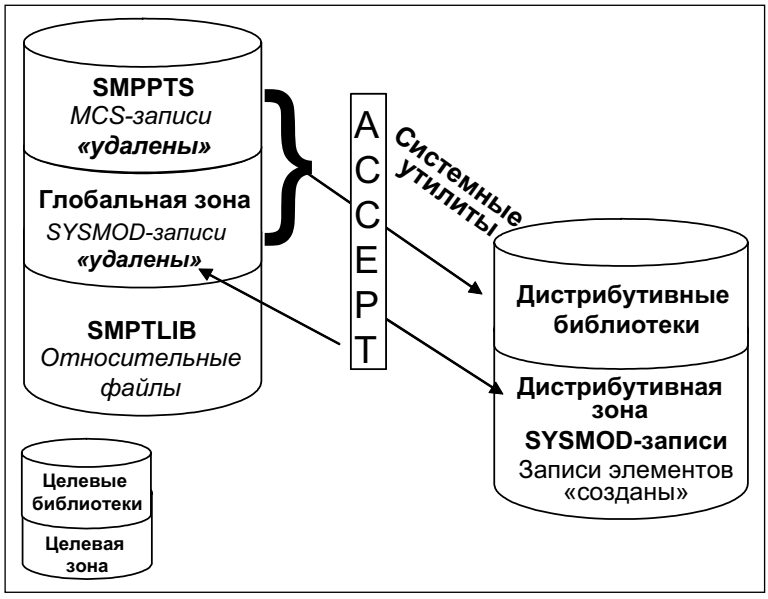


Рис. 17.17. Обработка команды ACCEPT в SMP/E

Команда ACCEPT выполняет следующие задачи (рис. 17.17):

- Обновляет записи CSI, указывая целевые элементы в дистрибутивной зоне.
- Выполняет перекомпоновку или создание целевых элементов в дистрибутивных библиотеках, используя содержимое SYSMOD в качестве входных данных.

- Проверяет записи CSI целевой зоны для затрагиваемых модулей и SYSMOD на согласованность с содержимым библиотеки.
- Выполняет обслуживание устаревших элементов. При обработке команды ACCEPT удаляются записи CSI глобальной зоны, разделы PTS и SMPTLIB для затрагиваемых SYSMOD. Например, ACCEPT удаляет записи SYSMOD глобальной зоны и операторы MCS в наборе данных SMPPTS для тех SYSMOD, которые были приняты в дистрибутивной зоне.

Есть возможность пропустить очистку глобальной зоны в SMP/E. В этом случае SMP/E сохраняет эту информацию.

SMP/E может остановить обработку команды ACCEPT. Это позволяет убедиться в том, что все предварительные требования выполнены, перед установкой SYSMOD. Это позволяет посмотреть, что произойдет, в действительности не модифицируя дистрибутивные библиотеки (что помогает выявлять проблемы).

Важно! Если SYSMOD содержит ошибку, не следует его принимать (командой ACCEPT). Процесс RESTORE выбирает модули, обновленные заданным SYSMOD, и выполняет перекомпоновку их копий в целевых библиотеках, используя в качестве входных данных определенные модули в дистрибутивных библиотеках. Кроме того, RESTORE обновляет записи целевой зоны CSI, чтобы отразить удаление SYSMOD. После завершения обработки команды ACCEPT, невозможно отменить изменения, так что изменения становятся перманентными.

После применения SYSMOD в целевой зоне можно указать SMP/E, что нужно установить только приемлемые PTF SYSMOD в дистрибутивной зоне:

```
SET BDY(ZOSDLB1).
ACCEPT PTFS.
```

Чтобы установить определенные PTF SYSMOD:

```
SET BDY(ZOSDLB1).
ACCEPT SELECT(UZ00001,UZ00002).
```

Иногда возникают ситуации, когда требуется обновить определенный продукт с использованием всех SYSMOD:

```
SET BDY(ZOSDLB1).
ACCEPT PTFS
FORFMID(H28W500).
```

или

```
SET BDY(ZOSDLB1).
ACCEPT FORFMID(H28W500).
```

Примечание. В двух вышеперечисленных случаях, SMP/E принимает все применимые PTF для продукта с FMID H28W500, расположенного в дистрибутивной зоне ZOSDLB1.

Принятие предварительных SYSMOD

При установке SYSMOD может быть неизвестно, имеются ли предварительные требования (это может быть связано с тем, что ERROR SYSMOD не устанавливаются). В таких ситуациях можно указать SMP/E, что необходимо проверить доступность аналогичного (или замещающего) SYSMOD, указав операнд GROUPEXTEND:

```
SET BDY(ZOSDLB1).
ACCEPT PTFS
FORFMID(H28W500)
GROUPEXTEND.
```

Примечание. Если SMP/E не может найти требуемый SYSMOD, он ищет и использует SYSMOD, замещающий требуемый.

Хороший способ посмотреть, какие SYSMOD включены, прежде чем их устанавливать, заключается в использовании операнда CHECK:

```
SET BDY(ZOSTGT1).
ACCEPT PTFS
FORMFMID(H28W500)
GROUPEXTEND
CHECK.
```

Отчеты ACCEPT

После завершения последнего этапа, следующие отчеты позволяют оценить результаты:

- SYSMOD Status Report (отчет состояния SYSMOD) – содержит обзор обработки, выполненной для каждого SYSMOD, на основании операндов, заданных в команде ACCEPT.

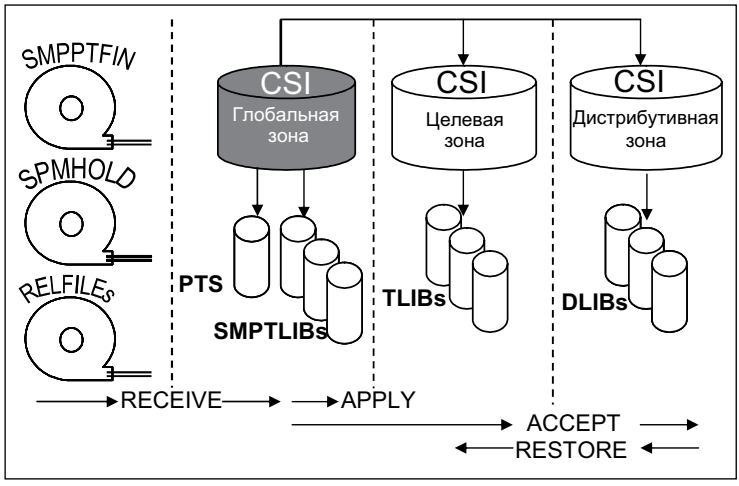


Рис. 17.18. Обзор процесса обработки SMP/E

```
//DEFINE   JOB   'accounting info',MSGLEVEL=(1,1)
//STEP01   EXEC  PGM=IDCAMS
//CSIVOL   DD    UNIT=3380,VOL=SER=volid1,DISP=SHR
//SYSPRINT DD    SYSOUT=A
//SYSIN    DD    *
  DEFINE CLUSTER(                          -
                 NAME(SMPE.SMPCSI.CSI)     -
                 FREESPACE(10 5)           -
                 KEYS(24 0)                -
                 RECORDSIZE(24 143)        -
                 SHAREOPTIONS(2 3)         -
        VOLUMES(volid1)            -
               )                           -
           DATA(                           -
                 NAME(SMPE.SMPCSI.CSI.DATA)  -
                 CONTROLINTERVALSIZE(4096)   -
                 CYLINDERS(250 20)           -
               )                           -
          INDEX(                           -
                 NAME(SMPE.SMPCSI.CSI.INDEX) -
                 CYLINDERS(5 3)              -
               )                           -
         CATALOG(user.catalog)
/*
```

Рис. 17.19. Пример определения кластера CSI VSAM

```
//SMPJOB   JOB 'accounting info',MSGLEVEL=(1,1)
//SMPSTEP  EXEC SMPPROC
//SMPPTFIN DD  ... points to the file or data set that contains
//*             the SYSMODs to be received
//SMPHOLD  DD  ... points to the file or data set that contains
//*             the HOLDDATA to be received
//SMPTLIB  DD  UNIT=3380,VOL=SER=TLIB01
//SMPCNTL  DD  *
  SET      BDY(GLOBAL)        /* Set to global zone       */.
  RECEIVE  SYSMOD             /* receive SYSMODS and      */
           HOLDDATA           /* HOLDDATA                 */
           SOURCEID(MYPTFS)   /* Assign a source ID       */
                              /*                          */.
  LIST     MCS                /* List the cover letters   */
           SOURCEID(MYPTFS)   /* for the SYSMODs          */
                              /*                          */.
  SET      BDY(TARGET1)       /* Set to target zone       */.
  APPLY    SOURCEID(MYPTFS)   /* Apply the SYSMODs        */
                              /*                          */.
  LIST     LOG                /* List the target zone log */.
/*
```

Рис. 17.20. Пример пакетного задания SMP

- Element Summary Report (обзорный отчет элементов) – содержит подробное описание каждого элемента, затронутого обработкой команды ACCEPT, и указывает библиотеку, в которой он находится.
- Causer SYSMOD Summary Report (обзорный отчет причинных SYSMOD) – содержит список SYSMOD, вызвавших отказы других SYSMOD, и описывает ошибки, которые нужно исправить для успешной обработки.
- File Allocation Report (отчет о распределении файлов) – содержит список наборов данных, используемых для принятия (командой ACCEPT) обработки, и предоставляет информацию об этих наборах данных.

17.11.4 Прочие средства SMP/E

Вся информация, расположенная в глобальной зоне (рис. 17.18), в сочетании с информацией, находящейся в целевой и дистрибутивной зонах, составляют данные, требуемые SMP/E для установки и отслеживания программного обеспечения системы, и часто они составляют большой объем данных. Эту информацию можно вывести, используя следующие средства SMP/E:

- Команда LIST создает распечатку с информацией о системе.
- Команда REPORT проверяет, сравнивает и генерирует распечатку с информацией о содержимом зоны.
- Диалоги QUERY в ISPF.
- SMP/E CSI API можно использовать для создания приложений, запрашивающих содержимое системы.

17.12 Наборы данных, используемые SMP/E

Возвратимся к обсуждению того, как SMP/E хранит информацию о системе. Когда SMP/E обрабатывает SYSMOD, он устанавливает элементы в соответствующих библиотеках и обновляет собственные записи о выполненной обработке. SMP/E устанавливает элементы программы в библиотеках двух типов.

- *Целевые библиотеки (target libraries)* содержат исполняемый код, необходимый для работы системы (например, библиотеки, с которых происходит запуск рабочей системы или тестовой системы).
- *Дистрибутивные библиотеки (distribution libraries, DLIB)* содержат главные копии каждого элемента системы. Они используются как входные данные SMP/E-команды GENERATE или процесса генерации системы для построения целевых библиотек новой системы. Кроме того, они используются в SMP/E для резервного копирования, когда необходимо выполнить замену или обновление элементов в целевых библиотеках.

Для установки элементов в этих библиотеках, SMP/E использует базу данных, состоящую из нескольких типов наборов данных.

- Наборы данных SMPCSI (CSI) представляют собой наборы данных VSAM, используемые для управления процессом установки и записи результатов обработки. CSI можно разделить на несколько разделов, используя ключевую структуру VSAM. Каждый раздел называется *зоной* (*zone*).

Существует три типа зон.

- Одиночная *глобальная зона* (*global zone*) используется для записи информации о SYSMOD, полученных в наборе данных SMPPTS. Глобальная зона также содержит информацию, позволяющую SMP/E осуществлять доступ к другим двум типам зон, информацию о системных утилитах, вызываемых SMP/E для установки элементов из SYSMOD, и информацию, позволяющую настраивать обработку SMP/E.
- Одна или несколько *целевых зон* (*target zones*) используются для записи информации о состоянии и структуре (целевых) библиотек операционной системы. Каждая целевая зона также указывает на соответствующую дистрибутивную зону, которая может использоваться при работе команд APPLY, RESTORE и LINK, когда SMP/E обрабатывает SYSMOD, и требуется проверить уровень элементов в дистрибутивных библиотеках.
- Одна или несколько *дистрибутивных зон* (*distribution zones*) используются для записи информации о состоянии и структуре дистрибутивных библиотек (DLIB). Каждая DLIB-зона также указывает на соответствующую целевую зону, которая используется, когда SMP/E принимает SYSMOD, и требуется проверить, был ли SYSMOD уже применен.

Набор данных SMPCSI может содержать несколько зон (в действительности, набор данных может содержать до 32 766). Например, набор данных SMPCSI может содержать глобальную зону, несколько целевых зон и несколько дистрибутивных зон. Зоны также могут находиться в отдельных наборах данных SMPCSI. Один набор данных SMPCSI может содержать только глобальную зону, другой набор данных SMPCSI содержит целевые зоны, и третий набор данных SMPCSI содержит дистрибутивные зоны.

- Набор данных SMPPTS (PTS) представляет собой набор данных для временного хранения SYSMOD, ожидающих установки. PTS используется только как набор данных для хранения SYSMOD. Команда RECEIVE сохраняет SYSMOD непосредственно в PTS без каких-либо изменений в информации SMP/E. PTS связан с глобальной зоной, так как оба набора данных содержат информацию о полученных SYSMOD. Только один PTS можно использовать для заданной глобальной зоны. Поэтому можно рассматривать глобальную зону и PTS как пару наборов данных, обрабатываемых (например, удаляемых, сохраняемых или модифицируемых) параллельно.
- Набор данных SMPSCDS (SCDS) содержит резервные копии записей целевой зоны, модифицируемых во время обработки команды APPLY. Таким образом, каждый SCDS непосредственно связан с определенной целевой зоной, и каждая целевая зона должна иметь собственный SCDS. Наборы данных SCDS используются в SMP/E для хранения резервных копий записей целевой зоны, модифицируемых во время обработки команды APPLY. Таким образом, каждый SCDS непосредственно связан с определенной целевой зоной, и каждая целевая зона должна иметь собственный SCDS.

SMP/E также использует следующие наборы данных:

- Набор данных SMPMTS (MTS) представляет собой библиотеку, в которой SMP/E хранит копии макросов во время установки, если не задана другая целевая

библиотека макросов. Таким образом, MTS связан с определенной целевой зоной, и каждая целевая зона должна иметь собственный набор данных MTS.

- Набор данных SMPSTS (STS) представляет собой библиотеку, в которой SMP/E хранит копии исходного кода во время установки, если не задана другая целевая библиотека исходного кода. Таким образом, STS связан с определенной целевой зоной, и каждая целевая зона должна иметь собственный набор данных STS.
- Набор данных SMPLTS (LTS) представляет собой библиотеку, содержащую базовую версию загрузочного модуля. Загрузочный модуль в этой библиотеке определяет распределение SYSLIB для явного включения модулей. Таким образом, LTS связан с определенной целевой зоной, и каждая целевая зона должна иметь собственный набор данных LTS.
- Другие наборы данных утилит и рабочие наборы данных.

SMP/E использует информацию в наборах данных CSI для выбора требуемых уровней устанавливаемых элементов, для определения того, какие библиотеки должны содержать те или иные элементы, и какие системные утилиты нужно вызывать для установки.

Системные программисты могут также использовать наборы данных CSI для получения последней информации о структуре, содержимом и состоянии системы. SMP/E предоставляет эту информацию в отчетах, листингах и диалогах для выполнения следующего:

- изучения функциональных и сервисных уровней;
- представления о пересечениях и связях SYSMOD (установленных или ожидающих установки);
- построения потоков заданий для обработки в SMP/E.

17.13 Заключение

Как системный программист z/OS, вы будете отвечать за обеспечение правильной установки всех программных продуктов и их модификаций в системе. Основным средством установки и обновления программного обеспечения в z/OS является SMP/E.

SMP/E можно запустить либо используя пакетные задания, либо используя диалоги в ISPF/PDF (Interactive System Productivity Facility/Program Development Facility). Используя диалоги SMP/E, можно осуществлять интерактивное опрашивание базы данных SMP/E, а также создание и передачу заданий для обработки команд SMP/E.

Программное обеспечение, устанавливаемое с использованием SMP/E, должно быть упаковано как системные модификации (SYSMOD), совмещающие обновленный элемент с управляющей информацией. Эта информация описывает элементы и связи программного обеспечения с другими продуктами или службами, которые также могут быть установлены в системе.

На большом предприятии системный программист часто использует SMP/E JCL и команды, однако MCS-инструкции SMP/E редко составляются тем же системным программистом. Упаковка продукта и SYSMOD включает необходимые MCS-операторы.

Критически важной обязанностью системного программиста является работа с группой устранения дефектов IBM при возникновении проблем в z/OS или дополнительных продуктах IBM. Разрешение проблем требует от системного программиста получения и применения исправлений в системе предприятия.

Основные термины в этой главе

Авторизованный отчет об анализе программы (APAR)	Консолидированная база данных состава программного обеспечения (CSI)	Дистрибутивная библиотека (DLIB)
Дистрибутивная зона	Глобальная зона	Временное программное исправление (PTF)
SYSMOD	Целевая библиотека	Целевая зона

17.14 Контрольные вопросы

Чтобы проверить понимание материала этой главы, ответьте на следующие вопросы:

1. Какое назначение SMP/E имеет в системе z/OS?
2. Какие два типа функциональных SYSMOD существуют? В чем их различия?
3. Какая информация хранится в наборе данных CSI?
4. В чем различие между APAR и PTF?
5. Заполните пробел: В SMP/E команда _____ используется для установки выбранных SYSMOD в соответствующих дистрибутивных библиотеках.

17.15 Темы для дальнейшего обсуждения

В чем состоит важность организованного процесса управления изменениями в предприятии, использующем большие системы? Каким образом установка и обслуживание обеспечивают основу для высокой доступности в среде z/OS?

18

Безопасность в z/OS

Цель. При работе с z/OS нужно понимать важность безопасности и знать средства, используемые для реализации безопасности в z/OS. Данные и приложения инсталляции относятся к наиболее важным ресурсам. Они должны быть защищены от несанкционированного доступа как изнутри (от сотрудников), так и извне (от клиентов, бизнес-партнеров и хакеров).

После завершения работы над этой главой вы сможете:

- объяснить понятия безопасности и целостности;
- описать RACF и его интерфейс с операционной системой;
- выполнить авторизацию программы;
- объяснить концепцию целостности системы;
- описать важность управления изменениями;
- описать понятие оценки рисков.

18.1 Зачем нужна защита?

В последние годы стало гораздо легче создавать компьютерную информацию и осуществлять к ней доступ. Доступ к системе больше не является привилегией кучки высококвалифицированных программистов. Информация теперь может создаваться и быть доступной почти для любого, кто найдет немного времени для ознакомления с современными, простыми в использовании высокоуровневыми языками запросов.

Все больше и больше людей становятся зависимыми от компьютерных систем и информации, которую они хранят в этих системах. По мере роста компьютерной грамотности и увеличения количества людей, использующих компьютеры, необходимость защиты данных вышла на новый уровень важности. Теперь недостаточно просто надеяться, что никто не знает, как получить доступ к данным, и считать при этом, что данные находятся в безопасности.

Кроме того, обеспечение безопасности данных не только означает необходимость сделать конфиденциальные данные недоступными для тех, кто их не должен видеть. Это также означает недопущение неумышленного повреждения файлов людьми, которые могут даже не знать, как правильно обращаться с данными.

Считается, что операционная система обладает системной целостностью когда она проектируется, реализуется и обслуживается таким образом, чтобы обеспечить защиту от несанкционированного доступа, и при этом средства безопасности, определенные в системе, нельзя скомпрометировать. В частности, для z/OS это означает, что ни одна несанкционированная программа, используя любой системный интерфейс, определенный или неопределенный, не должна быть способна сделать следующее:

- получить управление в авторизованном состоянии;
- обойти защиту памяти от несанкционированного доступа;
- обойти проверку пароля.

18.2 Средства безопасности в z/OS

В последующих разделах рассматриваются средства z/OS, обеспечивающие высокий уровень безопасности и целостности.

Данные о клиентах являются ценным ресурсом, который можно продать конкурентам. Поэтому целью любой политики безопасности является предоставление пользователям минимального требуемого уровня доступа и недопущение доступа неавторизованных пользователей. Это одна из причин, почему аудиторы предпочитают, чтобы пользователям и группам назначались специальные права доступа, а не предоставлялись средства универсального доступа. Традиционный фокус безопасности мэйнфреймов заключался в том, чтобы не допустить подключения неавторизованных пользователей к системе, и чтобы убедиться в том, что пользователи могут осуществлять доступ только к тем данным, которые им нужны. Когда мэйнфреймы начали использовать в качестве Интернет-серверов, потребовалась дополнительная безопасность. Существуют внешние угрозы, такие как хакеры, вирусы и «троянские кони»; сервер безопасности содержит инструменты для борьбы с ними.

Однако основной угрозой для данных компании всегда была угроза изнутри. Сотрудник компании имеет больше возможностей получить данные, чем кто-либо снаружи. Хорошо продуманная политика безопасности всегда является первой линией защиты.

Кроме того, z/OS содержит множество средств обеспечения целостности, позволяющих минимизировать преднамеренное или случайное повреждение данных другими программами. Во многих инсталляциях запущено несколько копий z/OS, и обычным пользователям TSO/ISPF часто не разрешается осуществлять доступ к производственным системам. Если средства безопасности z/OS настроены должным образом, они могут защитить производственную среду и не допустить (злоумышленного или случайного) воздействия пользователя TSO/ISPF на важные рабочие данные.

18.3 Роли, связанные с безопасностью

В прошлом безопасностью занимался системный программист, который совместно с руководством разрабатывал общую политику и процедуры безопасности. Современные компании требуют более высокого уровня безопасности, поэтому они назначают отдельного менеджера безопасности. Системный программист может непосредственно не отвечать за безопасность, а только давать менеджеру безопасности рекомендации по поводу новых продуктов. Разделение обязанностей необходимо для недопущения неконтролируемого доступа к системе со стороны любого лица.

Системный администратор назначает пользовательские идентификаторы и изначальные пароли, а также обеспечивает надежность, случайность и частое изменение паролей. Так как идентификаторы и пароли пользователей критически важны, следует обращать особое внимание на защиту файлов, которые их содержат.

18.4 IBM Security Server

Во многих инсталляциях используется пакет IBM Security Server, который часто еще называют RACF, по названию самого известного компонента.

Средства безопасности в z/OS выполняют следующие функции:

- контроль доступа пользователей к системе (с использованием идентификатора и пароля пользователя);
- ограничение действий, которые авторизованный пользователь может выполнять над системными файлами данных и программами.

Для тех студентов, которые хотят знать больше об инструментах, доступных для администратора безопасности z/OS, ниже приведен список компонентов безопасности z/OS, которые вместе называются Security Server:

- DCE Security Server.

  Этот сервер представляет собой полнофункциональный сервер безопасности уровня OSF DCE 1.1, выполняющийся в z/OS.
- Lightweight Directory Access Protocol (LDAP) Server.

Этот сервер основан на клиент-серверной модели, обеспечивающей клиентский доступ к LDAP-серверу. LDAP-каталог обеспечивает простой способ обработки информации каталогов в центральном месте для хранения, обновления, извлечения и обмена.

- z/OS Firewall Technologies.
  Представляет собой программу сетевого брандмауэра безопасности IPV4 для z/OS. По существу, брандмауэр z/OS содержит традиционные функции брандмауэра, а также поддержку виртуальных частных сетей.
  Включение брандмауэра означает, что мэйнфрейм можно напрямую подключить к Интернету без использования промежуточного оборудования, и что мэйнфрейм может обеспечить требуемые уровни безопасности для защиты важных данных компании. Используя технологию VPN, можно создавать надежно зашифрованные туннели, проходящие через Интернет от клиента к мэйнфрейму.
- Network Authentication Service for z/OS.
  Обеспечивает службы безопасности Kerberos, не требуя приобретения или использования программного продукта промежуточного уровня, такого как Distributed Computing Environment (DCE).
- Enterprise Identity Mapping (EIM).
  Обеспечивает новый подход к тому, чтобы недорогие системы могли с легкостью управлять несколькими пользовательскими реестрами и пользовательскими идентификационными данными на предприятии.
- PKI Services.
  Позволяет создать инфраструктуру открытых ключей и служит в качестве центра сертификации для внутренних и внешних пользователей, выдавая и администрируя цифровые сертификаты в соответствии с собственными политиками организации.
- Resource Access Control Facility (RACF).
  Основной компонент z/OS Security Server; работает в тесном взаимодействии с z/OS, обеспечивая защиту критически важных ресурсов.

Теме безопасности можно посвятить отдельную книгу. В этой книге мы рассмотрим компонент RACF и покажем, каким образом его средства используются для реализации безопасности z/OS.

18.4.1 RACF

В компьютерной среде доступом называется возможность сделать что-либо с компьютерным ресурсом (например, использовать, изменить или что-либо просмотреть). Управление доступом является методом явного разрешения или ограничения этой возможности. Компьютерные средства доступа называются *средствами управления логическим доступом* (*logical access controls*). Они представляют собой механизмы защиты, ограничивающие доступ пользователей к информации таким образом, чтобы они могли осуществлять доступ только к требуемой информации.

Средства управления логическим доступом часто встроены в операционную систему или могут быть частью логики приложений или основных утилит, например, систем управления базами данных. Они могут также быть реализованы в дополнительных пакетах безопасности, устанавливаемых в операционной системе; такие пакеты доступны для множества систем, включая персональные компьютеры и мэйнфреймы. Кроме того, средства управления логическим доступом могут быть реализованы в специализированных компонентах, управляющих связями между компьютерами и сетями.

Компонент, называемый средством управления доступом к ресурсам (Resource Access Control Facility – RACF) представляет собой дополнительный программный продукт, обеспечивающий базовую безопасность мэйнфрейм-системы. Существуют другие программные пакеты безопасности, например ACF2 или Top Secret (оба производства Computer Associates).

RACF защищает ресурсы, назначая доступ только авторизованным пользователям защищенных ресурсов. RACF сохраняет информацию о пользователях, ресурсах и полномочиях доступа в специальных структурах, называемых *профилями* (*profiles*), в своей базе данных, и обращается к этим профилям, когда определяет, какому пользователю следует разрешить доступ к защищенным системным ресурсам.

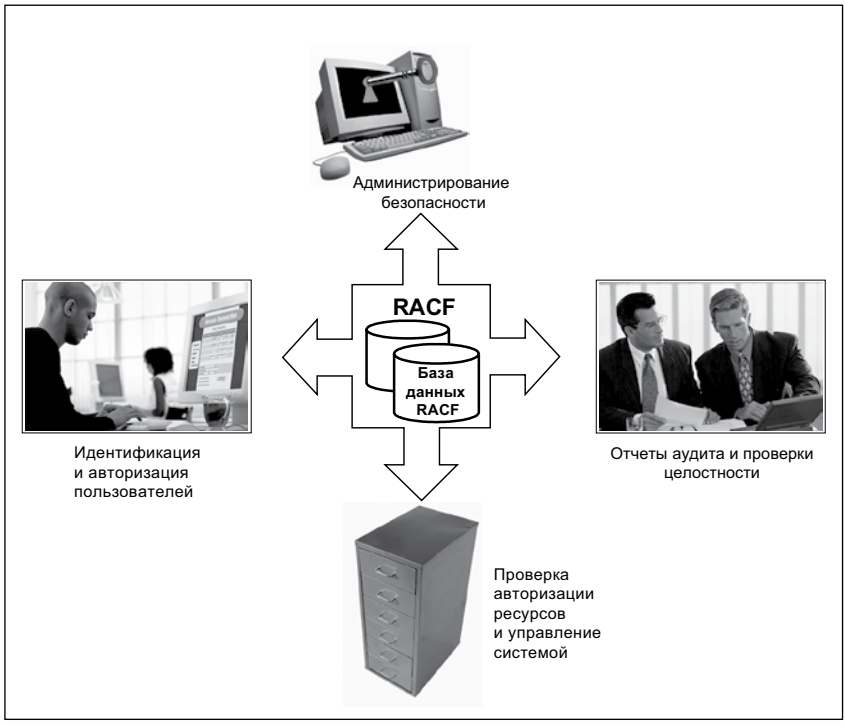


Рис. 18.1. Обзор функций RACF

Для достижения этих целей, RACF позволяет:

- осуществлять идентификацию и аутентификацию пользователей;
- осуществлять авторизацию пользователей для доступа к защищенным ресурсам;

- регистрировать и сообщать о попытках несанкционированного доступа к защищенным ресурсам;
- управлять средствами доступа к ресурсам;
- предоставлять приложениям возможность использования макросов RACF.

На рис. 18.1 показано простое представление функций RACF.

RACF использует идентификатор пользователя и зашифрованный пароль для выполнения идентификации и проверки пользователя. Идентификатор пользователя идентифицирует человека в системе как пользователя RACF. Пароль удостоверяет личность пользователя. Часто используются программы выхода для усиления политики паролей, например, для проверки соответствия требованию минимальной длины, отсутствия повторяющихся символов или символов, соответствующих соседним клавишам, а также для принудительного использования цифровых и буквенных символов. Популярные слова, такие как «password» или использование идентификатора пользователя часто запрещены.

Другой важной политикой является частота изменения пароля. Если идентификатор пользователя не использовался длительное время, он может быть аннулирован, и для его использования может потребоваться специальное действие. Когда кто-либо уходит из компании, должна выполняться специальная процедура, обеспечивающая удаление идентификаторов пользователей.

18.4.2 Средство авторизации системы (SAF)

Средство авторизации системы (system authorization facility, SAF) является частью операционной системы z/OS и предоставляет интерфейсы к вызываемым службам, осуществляющим аутентификацию, авторизацию и ведение журналов.

SAF не требует использования какого-либо продукта в качестве предварительного условия, но общие функции безопасности системы значительно расширяются и дополняются при использовании совместно с RACF. Основным элементом SAF является SAF-маршрутизатор. Этот маршрутизатор устанавливается в любом случае, даже если не установлен RACF.

SAF-маршрутизатор является общим центральным компонентом для всех продуктов, осуществляющих управление ресурсами. Этот центральный компонент содействует использованию общих функций управления, разделяемых между продуктами и системами. Компоненты и подсистемы управления ресурсами обращаются к маршрутизатору z/OS при выполнении некоторых функций принятия решений во время работы, таких как проверка при управлении доступом и проверка авторизации. Эти функции называются *контрольными точками* (*control points*).

Средство SAF в определенных ситуациях передает управление в RACF (если RACF установлен) и/или пользовательской программе обработки при получении запроса от менеджера ресурсов.

18.5 Администрирование безопасности

Безопасность данных предполагает защиту данных от случайного или преднамеренного несанкционированного просмотра, изменения или повреждения. Исходя из этого определения, очевидно, что все инсталляции, осуществляющие обработку дан-

ных, имеют, по меньшей мере, потенциальные проблемы безопасности или управления. Пользователи знают на своем опыте, что меры безопасности данных могут оказывать значительное воздействие на работу с точки зрения административных задач и требований к конечному пользователю.

RACF назначает пользователю с атрибутом SPECIAL (администратору безопасности) много обязанностей как на уровне системы, так и на уровне группы. Администратор безопасности является центральной фигурой в планировании безопасности инсталляции, и ему необходимо:

- определить, какие функции RACF следует использовать;
- определить уровень защиты RACF;
- определить, какие данные должен защищать RACF;
- определить административные структуры и пользователей.

18.5.1 RACF Remote Sharing Facility (RRSF)

RACF Remote Sharing Facility (RRSF) позволяет администрировать и обслуживать базы данных RACF, распределенные на предприятии. Это средство осуществляет улучшение системной производительности, управления системой, доступности системы и практичности. RRSF помогает обеспечить целостность данных при отказах и простоях систем или сети. Это средство позволяет узнать, когда произошли основные события, и возвращает выходные данные для просмотра в удобное для вас время.

18.5.2 RACF с промежуточным программным обеспечением

Крупные подсистемы, такие как CICS и DB2, используют средства RACF для защиты транзакций и файлов. Большая часть работы по конфигурированию RACF-профилей для этих подсистем выполняется системными программистами CICS и DB2. Поэтому требуется, чтобы люди, выполняющие эти роли, имели глубокое понимание RACF и его связи с управляемым программным обеспечением.

18.6 Безопасность консоли оператора

Можно рассмотреть воздействие средств безопасности z/OS на системные функции на примере консолей оператора. Под безопасностью консоли понимается контроль над тем, какие команды могут вводить операторы на своих консолях для мониторинга и управления z/OS. Определение команд, разрешенных для ввода с консоли, или управление подключением операторов позволяет планировать безопасность операций в системе z/OS или сисплексе. Так как в сисплексе оператор на одной системе может вводить команды, влияющие на обработку на другой системе, требуемые меры безопасности усложняются, и требуется выполнить соответствующее планирование.

В консолях MCS можно использовать следующие средства для управления вводом операторами команд с консоли:

- ключевое слово AUTH в операторе CONSOLE в разделе CONSOLxx;
- ключевое слово LOGON в операторе DEFAULT, а также команды и профили RACF;

В консолях EMCS можно контролировать, что авторизованный пользователь SDSF или TSO/E может делать в сеансе консоли. Так как консоль EMCS можно связать с идентификатором пользователя TSO/E, а не с физической консолью, может потребоваться использовать RACF, чтобы определить не только то, какие команды z/OS может вводить пользователь, но и с каких терминалов TSO/E пользователь может вводить команды.

18.7 Целостность

z/OS содержит множество функций и средств, специально разработанных для защиты от преднамеренного или случайного воздействия одной программы на другую. Поэтому операционная система z/OS известна благодаря как безопасности, так и целостности системы.

В этом разделе рассматриваются:

- Средство авторизации программ (Authorized Program Facility, APF).
- Защита памяти.
- Межпространственная связь.

18.7.1 Авторизованные программы

z/OS содержит средство авторизации программ (Authorized Program Facility, APF), позволяющее определенным программам осуществлять доступ к системным функциям, требующим защиты. Средство APF было разработано таким образом, чтобы не допустить нарушений целостности. Инсталляция определяет, какие библиотеки содержат специальные функции или программы. Такие библиотеки называются APF-библиотеками.

APF-авторизованная программа может делать практически все, что ей потребуется. Она, в сущности, представляет собой расширение операционной системы. Она может перейти в супервизорный режим или в режим работы с системным ключом. Она может изменять управляющие блоки системы. Она может выполнять привилегированные инструкции (находясь в супервизорном режиме). Она может отключать ведение журналов, чтобы скрыть свои действия. Очевидно, что такие полномочия следует назначать очень редко, и для них нужно осуществлять тщательный мониторинг.

APF можно использовать для идентификации системных или пользовательских программ, которые могут использовать уязвимые системные функции. В частности, APF:

- ограничивает использование чувствительных (критичных) системных SVC-программ (и уязвимых пользовательских SVC-программ, если они вам нужны) в APF-авторизованных программах;
- позволяет системе использовать все модули задачи шага авторизованного задания только из авторизованных библиотек, чтобы программы не могли подделать модуль в потоке модулей задачи шага авторизованного задания.

Многие системные функции, в частности, вызовы супервизора (supervisor calls, SVC) или специальные пути выполнения через SVC, являются чувствительными. Доступ к этим функциям должны иметь только авторизованные программы, чтобы не допустить нарушения безопасности и целостности системы.

Система считает задачу авторизованной, если выполняющаяся программа имеет следующие свойства:

- Она выполняется в супервизорном режиме (15-й бит слова состояния программы (PSW) равен нулю). Более подробно PSW описывается в главе 3 «Общие сведения о z/OS».
- Она выполняется с PSW-ключом от 0 до 7 (биты 8–11 слова состояния программы (PSW) содержат значение в диапазоне от 0 до 7).
- Все предыдущие программы, выполнявшиеся в той же задаче, были APF-программами.

Библиотеки, содержащие авторизованные программы, называются авторизованными библиотеками. APF-авторизованные программы должны находиться в одной из нижеперечисленных авторизованных библиотек:

- SYS1.LINKLIB;
- SYS1.SVCLIB;
- SYS1.LPALIB;
- авторизованные библиотеки, определенные инсталляцией.

18.7.2 Защита памяти

Оборудование мэйнфрейма имеет функцию *защиты памяти* (*storage protection*). Обычно она используется для предотвращения несанкционированного изменения памяти. Также ее можно использовать для предотвращения несанкционированного чтения областей памяти, хотя z/OS защищает таким способом только небольшие области памяти. Защита памяти работает с 4-килобайтовыми страницами. Она не работает с виртуальной памятью, работает только с основной. Когда страница виртуальной памяти копируется с диска в свободную страницу основной памяти, z/OS также устанавливает в этой странице основной памяти соответствующий ключ защиты памяти.

Защита памяти имела гораздо более важное значение до того, как начали использовать системы с множеством адресных пространств. Когда множество пользователей и задач работали в одном адресном пространстве (или в основной памяти во времена, когда еще не было виртуальной памяти), защита пользовательской памяти от повреждения (или несанкционированного просмотра) была критически важна. С появлением z/OS, основным способом защиты памяти каждого пользователя является изоляция, обеспечиваемая благодаря использованию множества адресных пространств.

Приложения не могут изменять ключи защиты памяти. Используя функцию защиты памяти, обычное приложение (т. е. не являющееся *авторизованной программой*) не может защитить часть своей виртуальной памяти от других частей приложения в том же адресном пространстве.

Дополнительный бит защиты памяти (для каждой 4-килобайтовой страницы основной памяти) является *битом защиты страницы*. Он не позволяет даже системным программам (работающим с ключом 0, которые обычно могут выполнять сохранение в любом месте памяти) выполнять сохранение в этой странице. Этот бит обычно используется для защиты страниц LPA от случайного повреждения системными программами.

18.7.3 Взаимодействие адресных пространств

При правильном управлении таблицей страниц со стороны операционной системы, пользователи и приложения, работающие в разных адресных пространствах, полностью изолированы друг от друга. Исключением является общая область. Другим исключением является *взаимодействие адресных пространств* (*cross-memory communication*).

При корректной настройке в операционной системе программа в одном стандартном адресном пространстве может осуществлять связь с программами в других адресных пространствах. Существует множество возможностей взаимодействия адресных пространств, но чаще всего используются следующие две:

- возможность вызова программы, находящейся в другом адресном пространстве;
- возможность доступа (чтения, записи) к виртуальной памяти в другом адресном пространстве.

В первом случае используется инструкция *вызова программы* (*program call*, PC). После корректной настройки z/OS, для вызова программы, находящейся в другом адресном пространстве, нужна только одна аппаратная инструкция. В качестве примера можно привести DB2, основной продукт для управления базами данных компании IBM. Различные фрагменты DB2 занимают до четырех адресных пространств. С точки зрения DB2 пользователями могут быть пользователи TSO, пакетные задания и другое промежуточное программное обеспечение (например, веб-сервер). Когда эти пользователи передают SQL-инструкции для DB2, SQL-интерфейс приложения использует инструкцию *вызов программы* для получения обслуживания в других адресных пространствах DB2.

Межпространственное программирование является достаточно сложным и должно быть согласовано со средствами безопасности z/OS. На практике межпространственные связи чаще всего используются основными программными продуктам промежуточного уровня и редко напрямую используются стандартными приложениями.

При стандартном программировании приложений редко затрагивается эта сфера. Как аппаратная архитектура мэйнфрейма, так и внутренняя структура z/OS защищают эти функции от некорректного использования, и не возникало серьезных проблем безопасности или целостности, связанных с возможностями межпространственной связи.

18.7.4 Технологии брандмауэра z/OS

Традиционный брандмауэр выступает в качестве преграды между вашей интрасетью (безопасной внутренней частной сетью) и другой (небезопасной) сетью в Интернете. Назначение брандмауэра состоит в том, чтобы не допустить несанкционированной входящей или исходящей связи в безопасной сети. Брандмауэр выполняет следующие две функции:

- Он дает пользователям внутренней сети возможность использовать авторизованные ресурсы из внешней сети, не компрометируя данные и другие ресурсы своей сети.
- Он не позволяет пользователям, находящимся за пределами сети, подключаться для компрометации или атаки на внутреннюю сеть.

Брандмауэр может защищать сети несколькими способами. Он может осуществлять экранирование, запрещающее или разрешающее доступ на основании имени пользователя, имени узла и TCP/IP-протокола. Брандмауэр может также выполнять различные функции, пропуская авторизованных и не пуская неавторизованных пользователей. Он обеспечивает представление, что все взаимодействие между вашей сетью и сетью Интернет ограничивается брандмауэром, не позволяя внешнему миру видеть структуру внутренней сети.

Для управления доступом между внутренней интрасетью и Интернетом с одновременным осуществлением авторизованных транзакций технологией брандмауэра z/OS (z/OS Firewall Technologies) поддерживают три основных технологии: сетевые серверы, фильтры и трансляцию адресов, а также виртуальную частную сеть.

18.8 Заключение

Обеспечение безопасности данных не только означает необходимость сделать конфиденциальные данные недоступными для тех, кто их не должен видеть. Это также означает недопущение неумышленного повреждения файлов людьми, которые могут даже не знать, как правильно обращаться с данными. Без знаний эффективных методов защиты данных развитие технологий может привести к увеличению вероятности неумышленного или преднамеренного доступа, изменения или повреждения данных неавторизованными пользователями. Security Server представляет собой набор функций z/OS, обеспечивающих реализацию безопасности.

Средство авторизации системы (system authorization facility, SAF) является частью операционной системы z/OS и предоставляет интерфейсы к вызываемым службам, осуществляющим аутентификацию, авторизацию и ведение журналов.

Средства управления доступом к ресурсам (RACF) представляет собой компонент Security Server для z/OS, управляющий доступом ко всем защищенным ресурсам z/OS. RACF защищает ресурсы, разрешая доступ только авторизованным пользователям защищенных ресурсов, и сохраняет информацию о пользователях, ресурсах и полномочиях доступа в специальных профилях.

RACF содержит инструменты и базы данных, позволяющие лицензированным программам z/OS, таким как TSO, CICS и DB2, проверять уровень доступа пользователя, разрешая или запрещая использование наборов данных, транзакций и представлений баз данных.

RACF позволяет организации определять пользователей и группы, использующие систему, защищаемую RACF. Например, что касается секретаря организации, администратор безопасности использует RACF для определения пользовательского профиля, содержащего пользовательский идентификатор секретаря, начальный пароль и прочую информацию.

Для достижения своих целей, RACF позволяет:

- осуществлять идентификацию и аутентификацию пользователей;
- осуществлять авторизацию пользователей для доступа к защищенным ресурсам;
- регистрировать и сообщать о попытках несанкционированного доступа к защищенным ресурсам;

- управлять средствами доступа к ресурсам.

Работа по управлению системой z/OS включает следующее:

- консольные операции, определяющие способ взаимодействия операторов с z/OS для мониторинга или управления аппаратным и программным обеспечением;
- обработку сообщений и команд, составляющую основу взаимодействия оператора с z/OS и основу автоматизации z/OS.

Управление системой z/OS включает управление оборудованием, в частности, процессорами и периферийными устройствами, а также программным обеспечением, в частности, операционной системой z/OS, подсистемой управления заданиями и всеми приложениями, выполняющимися в z/OS.

Под безопасностью консоли понимается контроль над тем, какие команды могут вводить операторы на своих консолях для мониторинга и управления z/OS. Настройка, главным образом, выполняется в RACF и в разделе CONSOLxx библиотеки PARMLIB.

Основные термины в этой главе

Авторизованные библиотеки	Средство авторизации программ (APF)	Шифрование
Брандмауэр	Хакер	Бит защиты страницы
Пароль	Средства управления доступом к ресурсам (RACF)	Политика безопасности
Разделение обязанностей	Целостность системы	Идентификатор пользователя

18.9 Контрольные вопросы

Чтобы проверить понимание материала этой главы, ответьте на следующие вопросы:

1. Истинно ли следующее утверждение?

   Информация доступа в профилях ресурсов может устанавливаться только на уровне группы. Это означает, что один пользователь не может иметь атрибут обновления для определенного набора данных, если группа RACF, к которой относится пользователь, имеет только атрибут чтения.

2. В следующей ситуации, что произойдет с программой, если не использовать авторизованные SVC или специальные функции?

   а) Программа компонуется с AC=0.

   б) Запускается из APF-авторизованной библиотеки.

3. В следующем примере какие могут возникнуть проблемы в программе, запускаемой из библиотеки SYS1.LINKLIB, расположенной на томе MPRES2?

```
D PROG,APF,ENTRY=1
CSV450I 05.24.55 PROG,APF DISPLAY 979
FORMAT=DYNAMIC
ENTRY VOLUME DSNAME
1 MPRES1 SYS1.LINKLIB?
```

18.10 Темы для дальнейшего обсуждения

1. Как осуществляется защита наборов данных или файлов на других платформах? Существует ли способ не допустить выполнения определенного приложения?
2. RACF позволяет назначать пользователям атрибут администратора группы. В этом случае можно реализовать децентрализованное администрирование. Обсудите преимущества и недостатки.

18.11 Упражнения

1. Попробуйте подключиться к TSO после изменения процедуры начального входа с IKJACCNT на IKJACCN1. Ожидаемое сообщение имеет следующий вид:

   ```
   IKJ56483I THE PROCEDURE NAME IKJACCN1 HAS NOT BEEN AUTHORIZED
   FOR THIS USERID
   ```

2. Используя свой идентификатор пользователя TSO (на этот раз используя заданную по умолчанию процедуру входа IKJACCNT), попробуйте удалить набор данных ZPROF.JCL.NOT.DELETE, установленный стандартными заданиями в поставляемом JCL. Этот набор данных является защищенным, и вы можете только просмотреть его содержимое.
3. Выполните следующий образец JCL-кода, чтобы получить отчет утилиты DSMON с текущей структурой групп RACF (доступен в образце JCL-кода как раздел DSMON):

   ```
   //DSMONRPT JOB (POK,999),'DSMONREPORT',MSGLEVEL=(1,1),MSGCLASS=X,
   //  CLASS=A,NOTIFY=&SYSUID
   /*JOBPARM SYSAFF=*
   //*
   //* NOTE:
   //* REMEMBER THAT ICHDSM00 MUST BE RUN BY A USER WITH
   AUDITOR ATTRIBUTE
   //*
   //STEPNAME EXEC PGM=ICHDSM00
   //SYSPRINT DD SYSOUT=A
   //SYSUT2 DD SYSOUT=A
   //SYSIN DD *
   FUNCTION RACGRP
   /*
   ```

4. Проверьте, является ли библиотека SYS1.LINKLIB APF-авторизованной.
   - Используя команду DISPLAY APF для вывода всего списка APF.
   - Используя операнд ENTRY= команды DISPLAY APF.
   - Используя операнд DSNAME= команды DISPLAY APF. Проверьте номер записи в результате выполнения команды в системном журнале (syslog).
5. Следующий пример JCL-кода можно использовать для вызова утилиты ADRDSSU

и вывода сообщения WTOR на консоль. Команда WTOR позволяет вывести сообщение ADR112A на системную консоль. Сообщение ADR112A требует, чтобы оператор выполнил некоторое действие, после чего ввел ответ. Например, WTOR можно использовать для того, чтобы попросить оператора подключить требуемый том или стабилизировать (quiesce) базу данных, перед тем как задание DFSMSdss продолжит обработку (доступно в образце JCL-кода как раздел ADRDSSU).

```
//WTORTEST JOB  (POK,999),'USER',MSGLEVEL=(1,1),MSGCLASS=X,
//  CLASS=A,NOTIFY=&SYSUID
//         EXEC  PGM=ADRDSSU
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
 WTOR  'TEST'
/*
```

DFSMSdss назначает сообщению WTOR следующий код перенаправления :

1. Основное действие оператора.

   DFSMSdss назначает следующий код дескриптора сообщению WTOR:

2. Требуется немедленное действие.

   На главном экране SDSF выберите опцию SR (system requests) и введите любой ответ.

19

Сетевые связи в z/OS

Цель. При работе с сетевыми связями в z/OS вам потребуется взаимодействовать с TCP/IP- и SNA-сетями.

В этой главе мы рассмотрим следующие темы:

- сравнение различных моделей коммуникационных сетей;
- программные компоненты продукта z/OS Communications Server;
- сравнение топологии подобластей SNA и сетевой топологии APPN;
- использование IP-сети для передачи данных между SNA-приложениями;
- часто используемые команды TCP/IP и VTAM.

19.1 Связь в z/OS

Сетевая связь имеет как программные, так и аппаратные аспекты, и разделение программных и аппаратных задач связи распространено на больших предприятиях. Однако опытному эксперту по сетям нужно понимать оба аспекта. В этой главе представлен краткий обзор коммуникационного программного обеспечения мэйнфрейма.

Как системный программист, специалист по сетям должен привнести глубокое понимание коммуникационного программного обеспечения z/OS в любой проект, предполагающий работу с сетью компании. Хотя специалисты по сетевому оборудованию имеют определенные навыки и инструменты для поддержки физической сети, их опыт часто не включает знание коммуникационного программного обеспечения z/OS. Когда, например, разветвленная розничная сеть открывает новый магазин, системные программисты z/OS и специалисты по сетевому оборудованию должны согласовывать свои усилия.

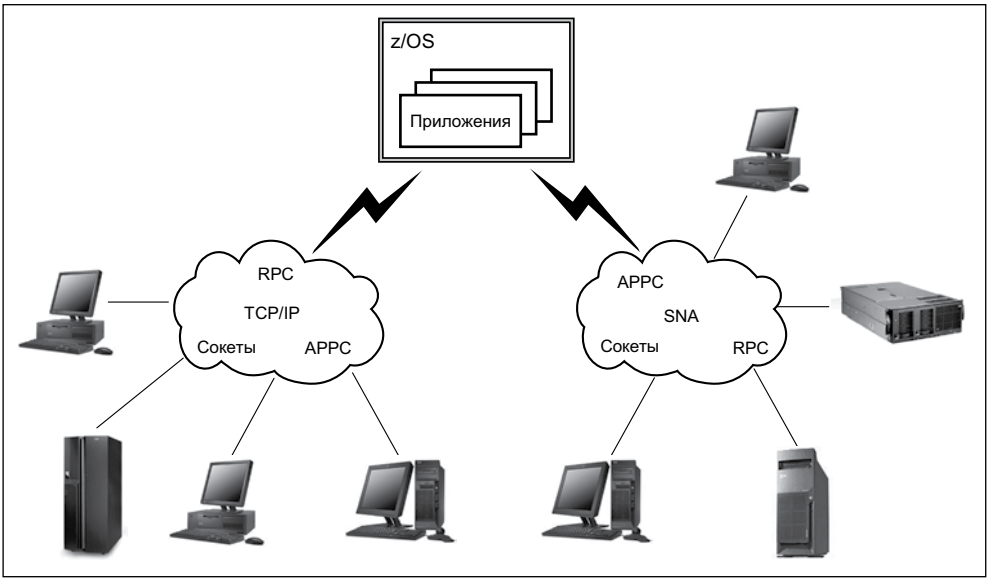


Рис. 19.1. Коммуникационный сервер IBM

z/OS включает полнофункциональный коммуникационный сервер с многопротокольной сетевой обработкой. Эта глава начинается с обзора существующих сетевых технологий в z/OS, после чего в разделе 19.3 «z/OS Communications Server» обсуждаются основные аспекты работы с коммуникационным сервером операционной системы.

19.2 Краткая история сетей данных

Разработанный в 1969 году, TCP/IP в действительности на пять лет старше, чем системная сетевая архитектура (System Network Architecture – SNA). Однако SNA стал сразу доступен общественности, тогда как TCP/IP сначала имел ограниченное использование в военных и исследовательских учреждениях и применялся во взаимосвязанных (сцепленных) сетях – предшественниках Интернета.

Кроме того, SNA включал средства управления сетями, которых изначально не было в TCP/IP, через протокол SDLC (Synchronous Data Link Control). В 1980-х годах протокол SNA имел широкое применение в крупных корпорациях, так как он позволял ИТ-организациям использовать централизованные вычислительные возможности во всемирном масштабе с приемлемой скоростью реагирования и надежностью. Например, широкое использование SNA позволило отрасли розничной торговли предлагать клиентам возможность уплаты в магазинах по кредитным картам.

В 1983 году TCP/IP стал общедоступным с появлением Berkeley BSD UNIX. Уровень развития TCP/IP, его приложения и уровень принятия росли посредством применения механизма Request-For-Comment (RFC), используемого комитетом по открытым стандартам Internet Engineering Task Force (IETF).

Термин *интерсеть* (*internet*) используется как общий термин в сети TCP/IP, и его не следует путать с *Интернетом* (*Internet*), состоящим из больших международных магистральных сетей, связывающих все TCP/IP-хосты, имеющие подключение к магистрали Интернета.

TCP/IP разрабатывался для взаимосвязанных сетей (интерсетей) и был проще в настройке, тогда как протокол SNA имел иерархическую структуру с «централизованным» мэйнфреймом на вершине иерархии. Структура SNA включала управление сетью, управление потоком данных и способность назначать приоритеты «класса обслуживания» определенным рабочим нагрузкам.

Связь между автономными SNA-сетями стала доступна в 1983 году. До этого времени SNA-сети не могли с легкостью взаимодействовать друг с другом. Способность независимых SNA-сетей совместно использовать бизнес-приложение и сетевые ресурсы называется межсетевым взаимодействием SNA (SNA Network Interconnect, SNI).

19.2.1 SNA и TCP/IP в z/OS

Протокол SNA (System Network Architecture) был разработан компанией IBM. SNA позволял корпорациям осуществлять обмен данными между своими узлами по всей стране. Для этого SNA содержал такие средства, как Virtual Telecommunication Access Method (VTAM), Network Control Program (NCP) и контроллеры терминалов, а также протокол SDLC (Synchronous Data Link Control). То, чем TCP/IP и Интернет стал для общества в 1990-х, тем SNA был для больших предприятий в 1980-х.

TCP/IP (Transmission Control Protocol/Internet Protocol) представляет собой непатентованный набор коммуникационных протоколов, являющийся отраслевым стандартом и обеспечивающий надежные подключения между приложениями через взаимосвязанные сети различных типов. TCP/IP получил широкое распространение с развитием сети Интернет, так как он позволял относительно дешево осуществлять доступ к удаленным данным и обработку данных. TCP/IP и Интернет привели к распространению небольших компьютеров и коммуникационного оборудования для поддержки чатов, электронной почты, ведения бизнеса, а также для загрузки и выгрузки данных.

Крупные предприятия, использующие SNA, увидели большой потенциал в том, чтобы расширить доступ к данным и приложениям, доступным через SNA, для растущего числа небольших компьютеров и коммуникационного оборудования из домов клиентов, небольших офисов и т. д.

19.2.2 Многоуровневые сетевые модели

TCP/IP и SNA представляют собой многоуровневые сетевые модели. Каждую из них можно сопоставить с международной сетевой моделью OSI (Open Systems Interconnect); (рис. 19.2).

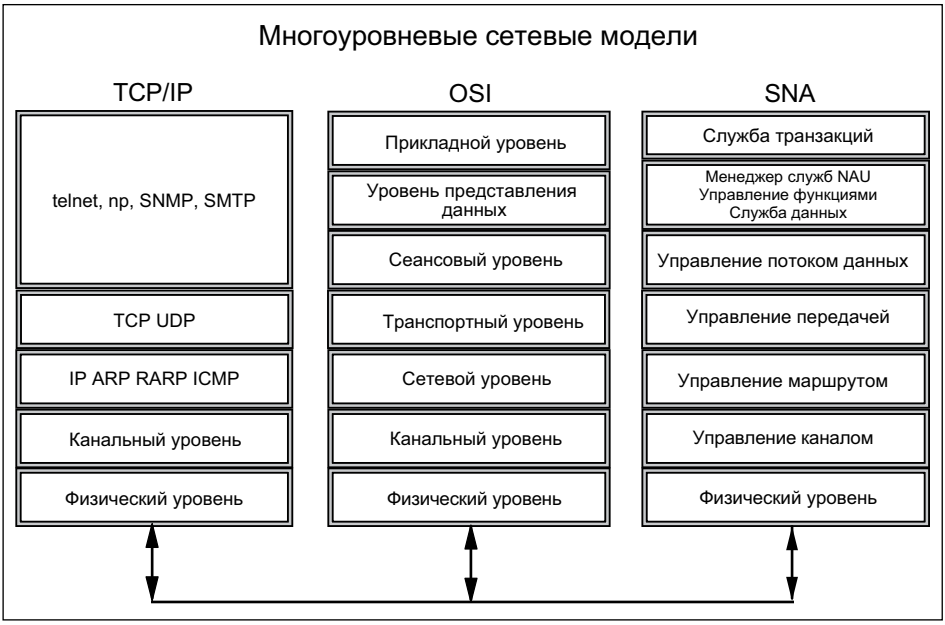


Рис. 19.2. Сетевая модель OSI (Open Systems Interconnect)

Сетевая модель OSI описывает структуру, состоящую из отдельных технологических элементов, осуществляющих межузловую передачу данных. Как показано на рис. 19.2, между сетевой моделью OSI и протоколами SNA и TCP/IP есть много общего. Хотя ни одна технология не имеет прямого соответствия сетевой модели OSI (TCP/IP и SNA появились до того, как была формализована сетевая модель OSI), есть много общего, так как определены уровни моделей.

Сетевая модель OSI разделена на семь уровней. Седьмой уровень модели OSI (прикладной) имеет непрямое соответствие верхним уровням стеков SNA и TCP/IP. Первый и второй уровни модели OSI (физический и канальный) соответствуют нижним уровням стеков SNA и TCP/IP.

В одном стандартном сценарии два географически рассредоточенных программных приложения на конечных узлах взаимодействуют посредством многоуровневой сетевой модели. Данные передаются одним конечным приложением и принимаются другим конечным приложением. Приложения могут размещаться на больших мэйнфреймах, персональных компьютерах, кассовых устройствах, банкоматах, терминалах или контроллерах принтеров. Конечные узлы в SNA называются логическими устройствами (logical units, LU), тогда как в терминологии IP они называются портами приложений (или просто портами).

Рассмотрим, как эту модель можно использовать в сетевых коммуникациях в большой сети продуктовых магазинов. Каждый раз, когда клиент оплачивает продукты на одном из множества кассовых устройств в магазине, многоуровневая сетевая модель используется дважды:

- Приложение кассового устройства находится на верхнем уровне локального многоуровневого сетевого стека.
- Приложение, записывающее сведения о покупке и авторизующее ее выполнение, находится на верхнем уровне удаленного многоуровневого сетевого стека.

Локальный сетевой стек может работать на системе, отличной от мэйнфрейма, с подключенными кассовыми устройствами, тогда как удаленный сетевой стек чаще всего работает на мэйнфрейме, обрабатывая транзакции, полученные со всех узлов магазина. Метод оплаты, список покупок, узел магазина и время записываются мэйнфрейм-приложениями, и разрешение на печать товарного чека возвращается обратно через сетевые стеки для завершения продажи.

Эта транзакционная модель часто называется запрос-серверным или клиент-серверным отношением.

19.2.3 Надежность и доступность сети

Что произойдет, если сеть или подключенный мэйнфрейм из нашего примера с сетью продуктовых магазинов по какой-то причине станет недоступным? Большинство кассовых систем, используемых в настоящее время, способны накапливать транзакции в интеллектуальном кассовом контроллере или небольшом процессоре магазина. После устранения причины отключения, накопленные транзакции передаются на мэйнфрейм одним пакетом.

В предыдущем примере восстановление транзакций является важным условием предотвращения проблем бухгалтерии и учета запасов в магазине и центральном офисе торговой сети. Суммарный эффект от неточных записей может с легкостью развалить бизнес. Поэтому надежность, доступность и удобство обслуживания (reliability, availability, serviceability – RAS) для сети настолько же важны, как и для мэйнфрейма.

19.2.4 Факторы, определяющие популярность SNA

Протокол SNA является стабильным и надежным протоколом, поддерживающим критически важные бизнес-приложения по всему миру. Значительный объем корпоративных данных в мире обслуживается SNA-приложениями под управлением z/OS[1]. Отличительная особенность SNA состоит в том, что этот протокол является протоколом на основе соединений, содержащим множество таймеров и механизмов управления, обеспечивающих надежную доставку данных.

ИТ-организации, использующие мэйнфреймы, часто скептически относятся к переходу с SNA на TCP/IP, несмотря на привлекательность TCP/IP и веб-коммерции. Этот скептицизм часто оправдан. Переделка стабильных, хорошо отлаженных бизнес-приложений под использование TCP/IP-сокетов вместо программных интерфейсов SNA может отнять много времени и средств.

[1] SNA-приложения, работающие под управлением z/OS, также называются VTAM-приложениями.

Многие предприятия предпочитают использовать веб-технологии, чтобы сделать огромное количество централизованных данных доступным в веб-среде, основанной на TCP/IP, продолжая при этом использовать SNA API. Такой подход позволяет взять лучшее из двух технологий и гарантирует использование SNA и VTAM в обозримом будущем.

19.3 z/OS Communications Server

В z/OS входит Communications Server, представляющий собой интегрированный набор программных компонентов, обеспечивающих сетевые коммуникации для приложений, запущенных в z/OS. Communications Server обеспечивает коридор транспортировки данных между внешней сетью и бизнес-приложениями, выполняющимися в z/OS.

z/OS Communications Server содержит набор коммуникационных протоколов, поддерживающих функции одноранговой связи как для локальных, так и для глобальных сетей, включая самую популярную глобальную сеть – Интернет. z/OS Communications Server также обеспечивает высокую производительность, доступную для множества TCP/IP-приложений, и включает множество часто используемых приложений.

Communications Server содержит множество сложных продуктов и функций. Основными службами являются:

- IP – использует TCP/IP (Transmission Control Protocol/Internet Protocol);
- SNA (Systems Network Architecture) – использует VTAM (Virtual Telecommunication Access Method).

Компонент CSM (Communications Storage Manager) поддерживает разделяемый буфер ввода-вывода для потока данных. Функция CSM позволяет авторизованным базовым приложениям совместно использовать данные без их физического перемещения.

Описание z/OS Communications Server (CS)

Интегрированные службы
- — Обеспечивают общие службы в CS для z/OS и OS/390
  - • Подключение к сети
  - • Управление памятью
  - • Высокоскоростная передача данных
- — Интеграция TCP/IP и SNA
  - • TN 3270
  - • Доступ к сети
  - • Внутренняя оптимизация
  - • Enterprise Extender
- — Стандартные TCP/IP-приложения

Мультипротокольные системы
- — Сокетные приложения (TCP/IP)
  - • Службы Unix обеспечивают доступ пользователей zSeries и S/390 к широкому диапазону UNIX-приложений через IP- и SNA-сети

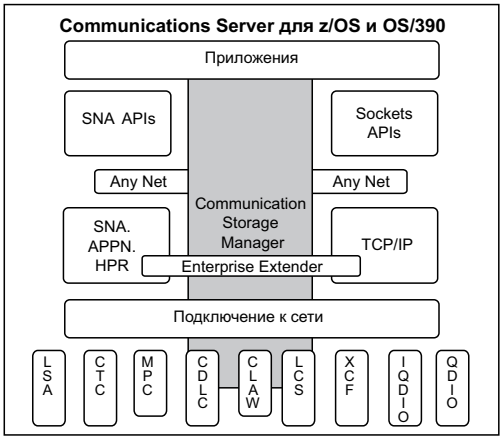


Рис. 19.3. z/OS Communications Server

Communications Server, содержащий функции TCP/IP и SNA, помимо z/OS реализован на многих других платформах, в частности, в AIX, Microsoft® Windows и Linux. В результате, программисты приложений z/OS могут использовать технологические усовершенствования в области коммуникаций (доступ к информации, электронная коммерция и совместная работа) между различными операционными системами.

19.4 Обзор TCP/IP

TCP/IP является общим термином, используемым для описания набора протоколов, составляющих основу Интернета. Впервые он появился в системе UNIX, предлагаемой Калифорнийским университетом в Беркли, и теперь устанавливается практически на всех компьютерах в мире, поддерживающих сетевые функции.

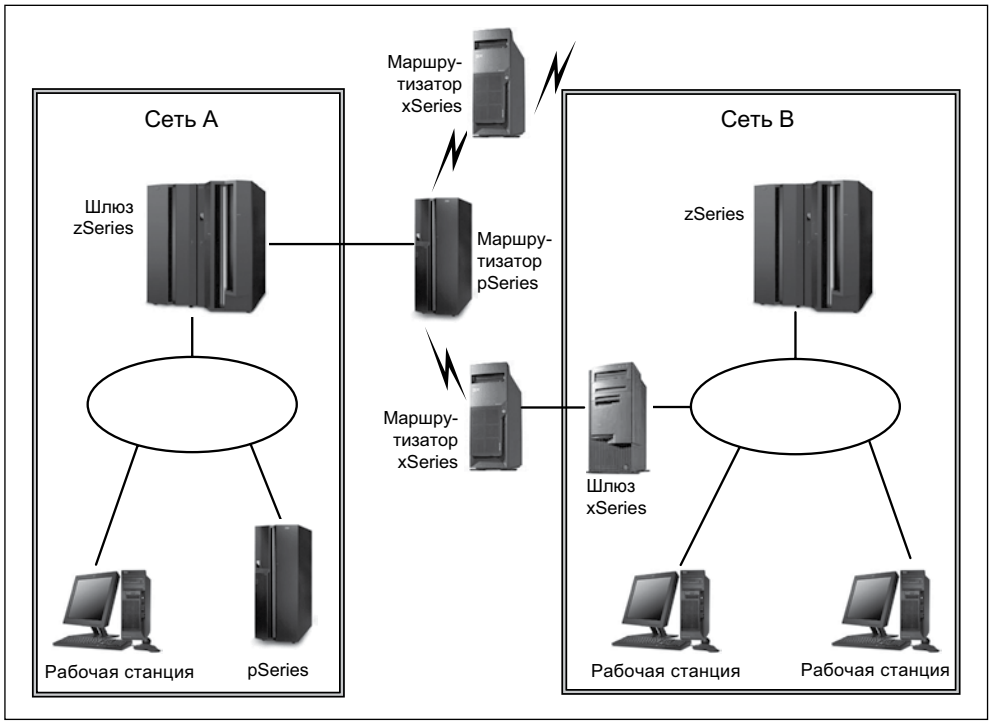


Рис. 19.4. Введение в TCP/IP

Все системы, независимо от размера, выглядят одинаково с точки зрения остальных систем в TCP/IP-сети. TCP/IP можно использовать как на оборудовании для локальных сетей, применяя общие протоколы, так и в глобальных сетях.

В сетевой среде TCP/IP компьютер, на котором запущен TCP/IP называется *хостом* (*host*). TCP/IP-сеть содержит один или несколько хостов, соединенных между собой различными каналами связи. Любой хост может напрямую обращаться ко всем другим хостам для установления соединения. Каналы связи между сетями невидимы для приложения, осуществляющего обмен данными с хостом. При этом неважно, ис-

пользуется ли традиционное CICS-приложение на основе служб MVS или новое установленное приложение, использующее компоненты UNIX. Системным администраторам может потребоваться выбрать, какие компоненты и интерфейсы программирования использовать. Однако с точки зрения приложения используемые компоненты невидимы.

19.4.1 Использование команд для мониторинга TCP/IP

z/OS поддерживает команды TCP/IP, существующие в других операционных системах, в частности:

```
NETSTAT
PING
TRACERTE
NSLOOKUP
```

Эти команды в авторизованном сеансе TSO можно вводить из:

- приглашения TSO Ready;
- командной оболочки ISPF;
- любой командной строки ISPF; перед командой вводится TSO;
- пакетных программ.

В Примере 19.1 показаны возможные результаты выполнения команды NETSTAT при ее вводе из приглашения TSO Ready.

Пример 19.1. Образец выходных данных команды NETSTAT

```
**** NETSTAT ROUTE output ****
MVS TCP/IP NETSTAT CS V1R5      TCPIP Name: TCPIP   21:38:18
Destination  Gateway     Flags  Refcnt  Interface
-----------  -------  -----  ------  ---------
Default      5.12.6.92   UGS    000001  OSA2380LNK
5.12.6.0     0.0.0.0     US     000000  OSA2380LNK
5.12.6.66    0.0.0.0     UH     000000  OSA2380LNK
5.12.6.67    0.0.0.0     UH     000000  STAVIPA1LNK
15.1.100.0   0.0.0.0     US     000000  IQDIOLNK0A016443
15.1.100.4   0.0.0.0     UHS    000000  IQDIOLNK0A016443
15.1.100.42  0.0.0.0     UHS    000000  IQDIOLNK0A016443
**** NETSTAT BYTE output ****
MVS TCP/IP NETSTAT CS V1R5 TCPIP Name: TCPIP 21:41:18
User Id   Conn      Local Socket        Foreign Socket      State
-------  ----  ------------  --------------  -----
DASU8G25  000048BF  0.0.0.0..523        0.0.0.0..0          Listen
D8G2DIST  00000072  0.0.0.0..38062      0.0.0.0..0          Listen
```

```
D8G2DIST   00000031  0.0.0.0..38060      0.0.0.0..0          Listen
FTPMVS1    00000022  5.12.6.66..21       0.0.0.0..0          Listen
FTPOE1     00000024  5.12.6.67..21       0.0.0.0..0          Listen
INETD4     00000083  0.0.0.0..7          0.0.0.0..0          Listen
INETD4     00000081  0.0.0.0..19         0.0.0.0..0          Listen
```

19.4.2 Использование консольных команд для управления TCP/IP

Для управления сетью и TCP/IP-приложениями в z/OS можно использовать команды DISPLAY TCPIP и VARY TCPIP.

Эти команды включают дополнительные опции, в частности:

- Команду DISPLAY TCPIP можно использовать для вывода следующей информации:
  - отчет NETSTAT для стека TCP/IP;
  - информация о OMPROUTE – демоне динамической маршрутизации;
  - использование памяти стека TCP/IP;
  - состояние стеков TCP/IP в сисплексе;
  - информация о сервере TN3270.
- Команда VARY TCPIP включает опции, позволяющие выполнять следующие действия:
  - удаление определенных сокетных подключений;
  - запуск и остановка коммуникационных устройств;
  - настройка трассировки пакетов и сокетов;
  - очистка записей ARP-кеша или кеша соседей;
  - изменение состояния стеков TCP/IP в сисплексе;
  - управление сервером TN3270;
  - изменение конфигурации TCP/IP-сети.

Дополнительные сведения. Дополнительные сведения об IP-командах см. в публикации *Communications Server IP System Administrators Commands*.

19.4.3 Использование VIPA для обеспечения доступности и балансировки нагрузки

TCP/IP в z/OS позволяет сетевым администраторам определять *виртуальные IP-адреса* (*virtual Internet protocol addresses*, VIPA). Управление VIPA-адресами осуществляется низкоуровневым коммуникационным протоколом, реализованным в z/OS. Такой адрес можно автоматически отобразить на несколько систем z/OS одновременно. Его можно динамически передавать с одной системы z/OS на другую, а также с одного IP-приложения на другое, даже если приложение существует на другом хосте z/OS. Все это происходит невидимо для сети и большинства приложений.

Кроме того, VIPA-адрес можно связать со всеми физическими адаптерами на хосте z/OS. Фактически не существует предела количества физических адаптеров, поддерживаемых хостом z/OS.

В то же время, VIPA-адреса совместно со службами управления рабочей нагрузкой в z/OS могут переносить входящие IP-подключения на хост z/OS, имеющий наибольшее количество доступных системных ресурсов.

19.4.4 TN3270 – шлюз в z/OS

В те времена, когда хосты z/OS работали в среде, поддерживающей только SNA, выделенные контроллеры терминалов и «немые» (непрограммируемые) терминалы были основой взаимодействия с z/OS. Протокол данных, используемый для связи между «немым» терминалом и z/OS, называется потоком данных 3270 (дополнительные сведения см. ниже в более полном обсуждении VTAM). Когда TCP/IP стал мировым стандартом, поток данных 3270 был адаптирован для использования в TCP/IP-сети. В результате скрещивания 3270 с существующим стандартом telnet появился стандарт Telnet 3270. В настоящее время TN3270 доработан и стал называться TN3270 Enhanced или TN3270E. Этот стандарт определен в RFC 2355.

Для связи с z/OS посредством протокола TN3270[1], на рабочей станции запускается клиент, например Personal Communications. Этот клиент создает сеанс TN3270, используя TCP. На стороне z/OS сервер TN3270 преобразует протокол TN3270 в протокол SNA, и подключение к приложению выполняется с использованием VTAM. Завершение SNA фрагмента обработки выполняется сервером TN3270, получающим логическое устройство (Logical Unit, LU) для обслуживания клиента TN3270. После этого устанавливается сеанс связи логических устройств (LU-LU) между сервером TN3270 и целевым приложением. Дополнительные сведения о сеансах LU-LU см. в обсуждении VTAM далее в этой главе.

19.5 Обзор VTAM

В z/OS VTAM обеспечивает работу сетевого коммуникационного стека SNA для транспортировки данных между приложениями и конечным пользователем. VTAM управляет ресурсами, определенными в сети SNA, устанавливает сеансы между этими ресурсами и отслеживает сеансовую активность.

VTAM выполняет множество задач в сети, например:

- осуществляет мониторинг и управление активацией и подключением ресурсов;
- устанавливает подключения и управляет потоком и ходом выполнения сеансов;
- предоставляет интерфейсы программирования приложений (например, APPC API для программирования LU 6.2), позволяющие осуществлять доступ к сети пользовательским приложениям и подсистемам IBM;
- обеспечивает поддержку интерактивных терминалов для TSO (Time Sharing Option);
- обеспечивает поддержку как локально, так и удаленно подключенных ресурсов.

[1] В настоящее время практически все клиенты используют TN3270E, однако термин TN3270 используется для описания обоих протоколов.

z/OS поддерживает только одно адресное пространство VTAM. Каждое приложение, использующее VTAM, например, CICS/TS, требует определения в VTAM. Приложение и VTAM используют это определение для установления подключений к другим приложениям или конечным пользователям.

Конечные точки сеанса SNA называются логическими устройствами (logical unit, LU). Логическое устройство представляет собой устройство или программу, используя которое конечный пользователь (приложение, оператор терминала или механизм ввода-вывода) получает доступ к SNA-сети. VTAM-сеансы называются сеансами LU-LU. Например, в SNA-сети CICS/TS считается логическим устройством и обычно имеет множество сеансов связи с другими логическими устройствами: дисплеями, принтерами, кассовыми устройствами и другими удаленными CICS/TS-регионами. Каждому логическому устройству назначается уникальный адресуемый сетевой элемент (network addressable unit, NAU) для обеспечения связи.

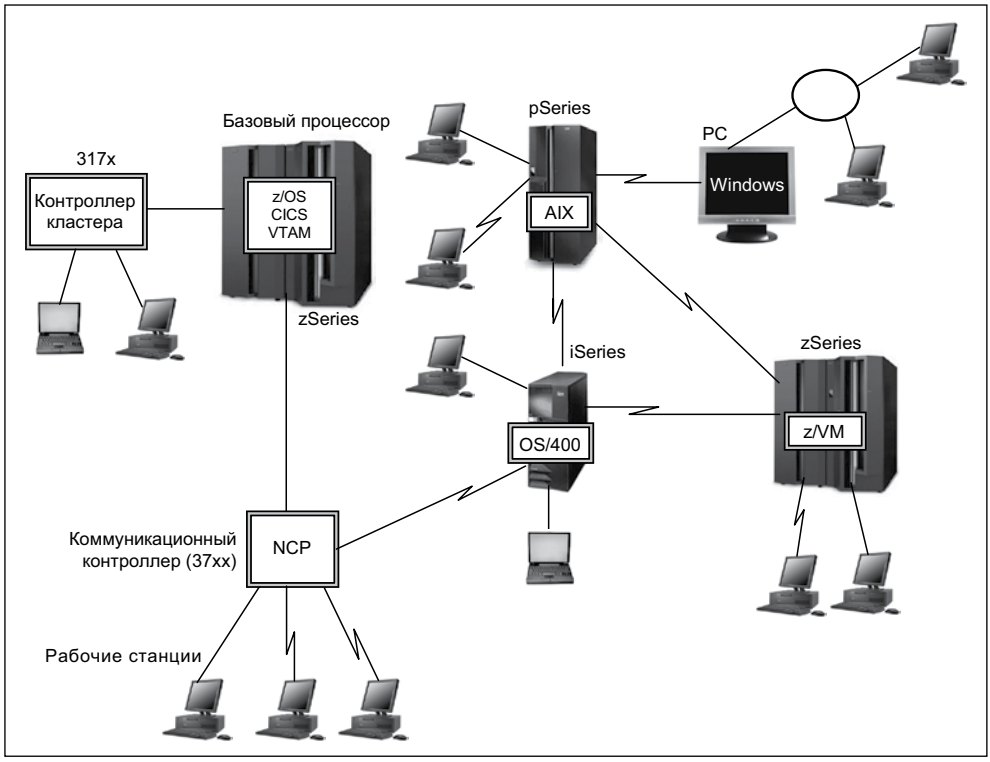


Рис. 19.5. Обзор VTAM – среда SNA

Физическое устройство (physical unit, PU) управляет одним или несколькими логическими устройствами. PU не является буквальным физическим устройством в сети. Оно скорее представляет собой часть устройства (программную и/или аппаратную), выполняющую функции управления устройством, в котором оно находится, и, в некоторых случаях, другими устройствами, подключенными к устройству, содержащему PU.

Физическое устройство присутствует в каждом узле SNA-сети, осуществляя управление и мониторинг ресурсов узла (например, подключенных каналов связи и смежных станций связи).

Физическое устройство находится либо внутри устройства, либо внутри подключенного управляющего устройства. VTAM должен активировать физическое устройство прежде, чем он сможет активировать и владеть каждым логическим устройством, подключенным к физическому устройству.

Даже мэйнфрейм является одним из типов физических устройств с подключенными логическими устройствами, примером одного из которых может служить CICS/TS. Существует три типа физических устройств (PU):

- PU Type 5 – находится на мэйнфрейме.
- PU Type 4 – коммуникационный контроллер глобальной сети.
- PU Type 2 – периферийный коммуникационный контроллер. Может напрямую подключаться к мэйнфрейму или к PU Type 4.

19.5.1 Топологии сетей, поддерживаемые VTAM

Несмотря на то, что самым распространенным способом связи с хостом z/OS является TCP/IP, в некоторых средах все еще используется протокол SNA, и многие среды передают (инкапсулируют) SNA-трафик через UDP/IP. Иерархическая структура протокола SNA реализует потребности больших предприятий в централизованной обработке данных. На вершине этой иерархии находится VTAM. VTAM поддерживает следующие типы топологий сетей:

- топология подобластей (subarea);
- APPN (Advanced Peer-to-Peer Network);
- смешанная топология подобластей/APPN.

Компонент VTAM, управляющий топологией подобластей, называется точкой управления системными службами (System Services Control Point – SSCP). Компонент VTAM, управляющий топологией APPN, называется точкой управления (Control Point – CP).

Сети подобластей VTAM предшествовали сетям APPN. На многих больших предприятиях переход с топологии подобластей на топологию APPN является желательным. Если сеть содержит Enterprise Extender (EE), доступны дополнительные функции, в том числе включение данных SNA-приложений в TCP/IP-пакеты. Это консолидирует старое коммуникационное оборудование, предназначенное для работы с протоколом SNA, направляя потоки данных SNA через существующие TCP/IP-сети. Кроме того, объем задач администрирования VTAM и согласования работы сотрудников по обслуживанию коммуникационного оборудования и программного обеспечения можно значительно сократить, используя топологию APPN, обладающую большей гибкостью, чем сети подобластей.

Все три типа конфигурации VTAM (топология подобластей, APPN и смешанная топология подобластей/APPN) используются на крупных предприятиях мира.

19.5.2 Что такое топология подобластей?

Отличительным свойством сети подобластей VTAM является владение и совместное использование SNA-ресурсов. Подобласть представляет собой набор SNA-ресурсов, управляемых одним узлом PU Type 5 или PU Type 4.

Одиночный VTAM и SNA-ресурсы, которыми он владеет, называются *доменом* (*domain*). Междоменный менеджер ресурсов (Cross-Domain Resource Manager, CDRM) позволяет осуществлять связь между VTAM-доменами в SNA-сети. Когда LU запрашивает создание сеанса с LU в отдельном VTAM-домене, разные VTAM взаимодействуют для создания *междоменного сеанса* (*cross-domain session*).

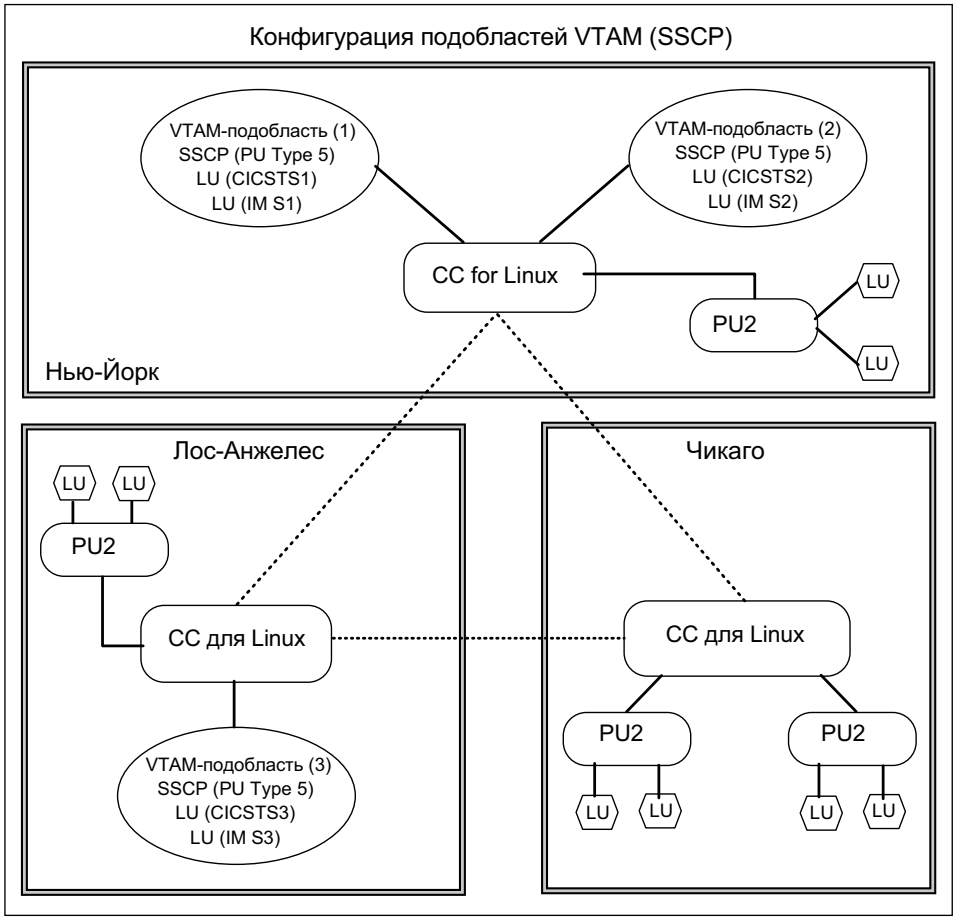


Рис. 19.6. Сеть топологии подобластей VTAM

На рис. 19.6 показана сеть с топологией подобластей VTAM. Эта схема может, например, представлять предприятие, расположенное в Нью-Йорке и имеющее значительное присутствие в Лос-Анжелесе с дальнейшим развитием в Чикаго. На рис. 19.6 изображено три домена VTAM и шесть подобластей. Подобласть Чикаго становится частью домена управляющего VTAM.

В целом желательно осуществить полную миграцию со старой топологии подобластей на топологию APPN, так как это позволяет использовать новую инфраструктуру IP-сетей и сократить расходы, благодаря устранению старого сетевого оборудования для SNA. Это также упрощает управление сетями VTAM, благодаря дополнительным динамическим возможностям.

19.5.3 Что такое топология APPN?

Улучшенный протокол одноранговых сетей (Advanced Peer-to-Peer Networking® – APPN) представляет собой тип связи, направляющий данные в сети между двумя и более системами, не требуя прямого подключения.

В топологии APPN не используется номер подобласти, равно как и не назначается исключительное владение SNA-ресурсами. Каждый VTAM, участвующий в APPN, включен в географически распределенное множество совместно используемых SNA-ресурсов, что устраняет необходимость использования междоменного менеджера ресурсов для установления сеансов.

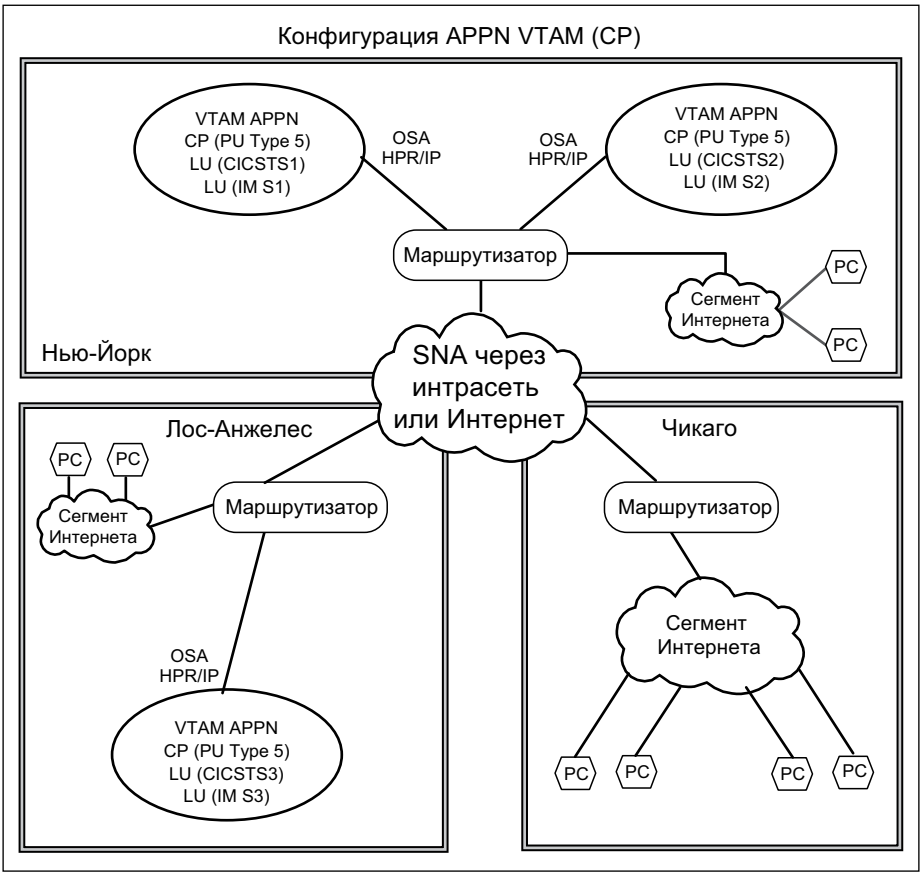


Рис. 19.7. Топология APPN

APPN включает метод высокопроизводительной маршрутизации (High-Performance Routing – HPR), осуществляющий передачу данных SNA-приложения через существующее сетевое оборудование TCP/IP. APPN содержит функцию Enterprise Extender (EE), иногда называемую HPR/IP. EE обеспечивает возможность обслуживания SNA-приложений современными IP-технологиями.

Использование HPR не ограничено исключительно топологиями SNA/APPN в TCP/IP-сетях. HPR скорее является функцией APPN, обеспечивающей высокоскоростную доставку данных через сеть APPN, совмещенную с высокодоступной функцией динамического изменения маршрута сеансов для обхода отказов в сети. Однако HPR поддерживается в большинстве типов APPN-подключений (не только в APPN через TCP/IP). Enterprise Extender (EE) представляет собой функцию APPN/HPR, позволяющую SNA-сеансам и другим функциям APPN (например, HPR) работать через TCP/IP-сеть (вместо собственной SNA-сети).

Предположим, что компания, изображенная на рис. 19.6, позже выполнит миграцию с топологии подобластей на топологию APPN. На рис. 19.7 представлена та же компания после миграции.

19.5.4 Обзор топологий VTAM

VTAM может представлять собой SSCP топологии подобластей, CP топологии APPN или вместе SSCP и CP, обслуживающие смешанную сеть. Желательно использовать более новую топологию APPN, так как она может непосредственно вести обмен данными с существующими IP-инфраструктурами.

Первоначальные SSCP подобластей обычно преобразуются в SSCP смешанной топологии или APPN CP, что позволяет использовать функцию EE HPR/IP и сократить расходы на коммуникационное сетевое оборудование SNA. Это обычно ведет к переносу всех остальных подобластей в топологию APPN, чтобы уменьшить сложность сети.

19.5.5 Использование команд мониторинга VTAM

Ниже приведен небольшой набор команд VTAM, используемых для сбора информации о среде VTAM.

- Вывод состояния ресурсов VTAM командами DISPLAY(D NET,):

D NET,VTAMOPTS	Вывод опций запуска VTAM.
D NET,CSM[,OWNERID=ALL]	Вывод сведений об использовании буфера коммуникаций.
D NET,APPLS	Вывод состояния определенных приложений (ACB).
D NET,MAJNODES	Вывод состояния всех основных узлов, активированных VTAM.
D NET,TOPO,LIST=SUMMARY	Вывод информации о топологии APPN.
D NET,CPCP	Вывод состояния сеансов APPN CP-CP.
D NET,SESSIONS	Вывод состояния сеансов подобластей SSCP-SSCP, сеансов LU-LU (включая сеансы CP-CP), сеансов SSCP-LU и сеансов SSCP-PU.
D NET,CDRMS	Вывод состояния междоменных менеджеров ресурсов подобластей.
D NET,EXIT	Вывод состояния программ-«выходов» VTAM.

- Активация/деактивация ресурсов VTAM командами VARY (V NET,).
- Изменение среды VTAM командами MODIFY (F VTAM,).

Для мониторинга и создания отчетов о состоянии ресурсов VTAM, системные программисты z/OS используют такие продукты, как Tivoli NetView.

Дополнительные сведения. Дополнительные сведения о командах VTAM см. в публикации *Communications Server SNA Operations*.

19.5.6 Общие сведения о потоке данных 3270

То, чем HTML является для веб-приложения и браузера, тем поток данных 3270 является для SNA-приложения и устройства в сеансе LU-LU. Специализированные команды вставляются в данные дисплейных устройств и принтеров. Поток данных 3270 включает эти встраиваемые инструкции и дескрипторы полей данных. Команды потока данных 3270 создаются и считываются SNA-приложениями, контроллерами физических устройств, управляющими дисплеями и принтерами, а также эмуляторами TN3270, доступными в операционных системах AIX и PC.

Одно из наиболее примечательных преимуществ потока данных 3270 состоит в том, что по нажатии клавиши Enter или PF полный экран записей и исправлений данных направляется в принимающее SNA-приложение.

Поток данных 3270 включает адреса полей данных, состоящих из столбца и строки, а также дескрипторы данных, такие как цвет, защищенные области экрана и незащищенные области экрана.

Когда SNA-приложения направляют данные на дисплей, они содержат положение «столбец/строка» для отдельных полей данных, дескрипторы полей данных и позицию курсора на экране. Способность потока данных 3270 осуществлять завершение ввода данных до отправки в SNA-приложения позволяет избежать излишних процессорных прерываний. В то же время каждое нажатие клавиши в сеансе VT100 vi требует обработки центральным процессором. При нажатии клавиш CNTRL-G в VT100 система должна распознать это нажатие и отобразить строку состояния.

Поток данных 3270 является критически важной составляющей способности SNA-сети осуществлять централизованное управление тысячами географически распределенных дисплеев и принтеров.

19.6 Заключение

Сети предприятий можно разрабатывать, настраивать, использовать и поддерживать, применяя сочетание возможностей и функций сетевых уровней SNA и TCP/IP, используя Communications Server в z/OS, AIX, Windows, Linux и Linux для zSeries.

Значительное количество больших предприятий используют приложения 3270 и SNA, не нуждаясь в изменении API для бизнес-приложений. В результате обеспечивается поддержка VTAM, а также его интеграция с такими технологиями как APPN, HPR и EE. Кроме того, TCP/IP использует VTAM для управления памятью, устройствами связи (все IP-устройства работают через VTAM) и обработки сеансов TN3270.

Предприятия могут для некоторых задач SNA использовать продукты Communications Server для замены некоторых старых компонентов инфраструктуры SNA, таких как контроллеры связи IBM 3725/45 (NCP) или другие SNA-контроллеры канального подключения.

Основные термины в этой главе

APPN	Communications Server	Интернет
LU-LU	NCP	OSI
SDLC	SNA	TCP/IP
PU	VTAM	LU
Стек	Сегмент Интернета	Подобласть

19.7 Контрольные вопросы

Чтобы проверить понимание материала этой главы, ответьте на следующие вопросы:

1. Какие компоненты совпадают в многоуровневых сетевых моделях SNA и TCP/IP?
2. Обслуживается ли большая часть корпоративных данных в мире приложениями z/OS SNA?
3. Нужно ли предприятию изменять SNA-приложения для обеспечения доступа через веб?
4. В чем состоит различие между сетью подобластей SNA и топологией APPN?
5. Почему использование топологии APPN является более желательным, чем использование подобластей SNA?
6. Что общего между HTML и потоком данных 3270?
7. Что общего между IP-адресом и адресуемым сетевым элементом (NAU) в SNA?
8. Какие ресурсы z/OS Communications Server совместно используются в TCP/IP и VTAM?
9. Какой компонент z/OS Communications Server обеспечивает разделяемый буферную область ввода-вывода данных для TCP/IP и VTAM?

19.8 Упражнения

1. В SDSF введите TCP/IP-команду /D TCPIP,,NETSTAT,HOME, а в ISPF введите TSO NETSTAT HOME.

   Совпадают ли выходные данные этих команд? Чему равен домашний IP-адрес или адреса данной системы z/OS?

2. В SDSF введите VTAM-команду /D NET,CSM.
   - Сколько пространства TOTAL ALL SOURCES используется (INUSE)?
   - Сколько пространства TOTAL ALL SOURCES свободно (FREE)?
   - Сколько пространства TOTAL ALL SOURCES доступно (AVAILABLE)?

   3. В SDSF введите следующие VTAM-команды:

   ```
   /D NET,APPLS
   ```

```
/D NET,MAJNODES
/D NET,TOPO,LIST=SUMMARY
/D NET,CPCP
/D NET,SESSIONS
/D NET,SESSIONS,LIST=ALL
/D NET,TSOUSER,ID=yourid
```

Запишите свой IP-адрес: ___.___.___.___

Кратко опишите, чем могут быть полезны выходные данные этой команды.

4. В ISPF запустите оболочку z/OS UNIX командой TSO OMVS.
   - Введите `netstat -h`.

   Совпадает ли информация с выходными данными в упражнении 1?

   - Обратите внимание на то, что можно использовать TCP/IP-команды из оболочки z/OS UNIX (префикс `o`).
   - Введите `ping your.ip.addr.ess` (ваш IP-адрес).
   - Введите `traceroute your.ip.addr.ess` (ваш IP-адрес).
   - Выйдите из оболочки z/OS UNIX (**`exit`**).

A

Краткий обзор истории мэйнфреймов IBM

В этом приложении рассматривается развитие мэйнфреймов IBM с 1964 года до настоящего времени, как показано на рис. А.1.

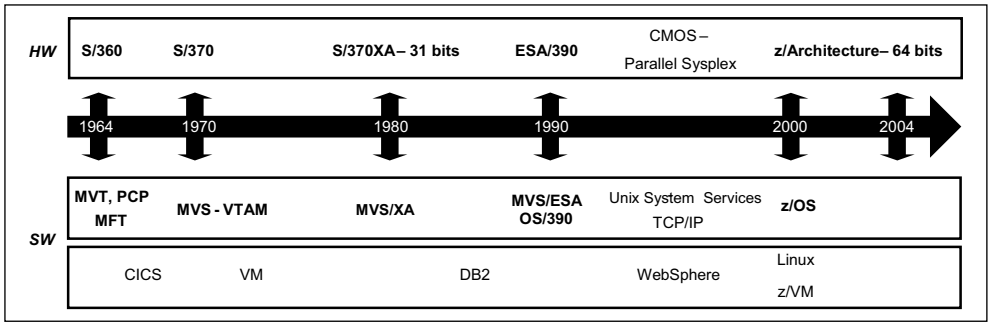


Рис. А.1. Хронологическая шкала развития мэйнфреймов IBM

7 апреля 1964 года компания IBM представила System/360 – семейство, состоящее из пяти мощных компьютеров с одинаковой операционной системой, могущих использовать одинаковые 44 периферийных устройства. Вместе с S/360 появились понятия подсистемы ввода-вывода (назначающей процессоры для передачи данных между памятью и устройствами ввода-вывода) и параллельных каналов (каналов параллельной передачи данных на устройства ввода-вывода).

Рис. А.2. S/360 Model 40

Впервые компании могли запускать критически важные приложения предприятия на высоконадежной платформе.

В 1968 году компания IBM представила систему CICS (Customer Information Control System). Она позволяла сотрудникам вычислительного центра оперативно вводить, изменять и извлекать данные. До сих пор CICS остается одним из наиболее популярных мониторов транзакций в отрасли.

В 1969 году удачная высадка «Аполлона-11» на Луну поддерживалась несколькими мэйнфреймами System 360, Information Management System (IMS) 360 и программным обеспечением компании IBM.

Рис. А.3. S/370™ Model 165

Летом 1970 года компания IBM представила семейство машин с расширенным набором инструкций под названием System/370. Эти машины были способны использовать несколько процессоров в одной системе (изначально было два процессора), совместно использующих память. В 1970-х годах компьютеры стали мощнее и быстрее, и многопроцессорные системы стали распространенными. Система 370 Model 145 была первым компьютером с полностью интегрированной монолитной памятью (схемой, в которой все элементы – резисторы, конденсаторы и диоды – размещались на одной кремниевой пластине) и 128-разрядными биполярными микросхемами. Более 1400 микроскопических схемных элементов были вытравлены на каждой микросхеме размером одна восьмая квадратного дюйма.

System/370 была способна выполнять программы для System/360, что уменьшало нагрузку на клиентов, связанную с обновлением программного обеспечения. Кроме того, System/370 была одной из первых моделей компьютеров, содержавших технологию «виртуальной памяти». Эта технология имела цель расширить возможности компьютера посредством использования пространства на жестком диске для удовлетворения требований программного обеспечения к памяти.

В 1980 году был представлен процессор 3081. Процессор 3081 обеспечивал двойное увеличение внутренней производительности по сравнению с предыдущей моделью процессора для мэйнфреймов – 3033. Он также содержал теплоотводящие модули, значительно сократившие требования к пространству, охлаждению и мощности.

Рис. А.4. Процессорный комплекс 3081

Около 1982 года произошло расширение адресации с 24-разрядной до 31-разрядной (370XA).

В 1984 году компания IBM представила 1-мегабитную микросхему на основе технологии SAMOS (Silicon and Aluminum Metal Oxide Semiconductor, кремниево-алюминиевый металл-оксид-полупроводник). Хотя «мега» обычно означает миллион, микросхема в действительности вмещала 1 048 576 бит информации на площади меньше детского ногтя.

В 1988 году были добавлены расширения, поддерживающие использование множества адресных пространств. В том же 1988 году, используя мэйнфреймы, клиенты смогли использовать базу данных DB2 не только как «систему поддержки решений», но как основную систему обработки транзакций, что обеспечило снижение затрат процессорного времени и значительное повышение параллельности вычислений.

В этот период компания IBM внедрила понятие логического раздела (logical partition, LP), что позволило осуществлять логическое разделение мэйнфрейма на несколько независимых процессоров, использующих общее оборудование.

Некоторые эксперты сомневались, что мэйнфреймы переживут начало 1990-х. Они предсказывали, что стремительно развивающиеся персональные компьютеры и небольшие серверы вытеснят «Большое железо» (отраслевое жаргонное название мэйнфрейма). Однако компания IBM считала, что на серьезные, сверхзащищенные, мощные вычислительные системы всегда будет спрос, в результате чего появился System/390. IBM осталась верной мэйнфреймам, однако переделала их изнутри, добавив в них совершенно новые технологии и снизив их цены.

Рис. А.5. S/390 G5 и G6

IBM представила концепцию системной кластеризации и совместного использования данных и выпустила System/390 Parallel Sysplex, что позволило достичь очень высокого уровня доступности систем.

В среде мэйнфреймов начали применять процессоры на основе технологии CMOS (Complementary Metal Oxide Semiconductor, комплементарный металл-оксид-полупроводник), заменившей биполярную технологию и установившей новое направление для современной технологии производства мэйнфреймов. CMOS-схемы требовали меньше электроэнергии, чем микросхемы, использовавшие только один тип транзистора.

В том же десятилетии компания IBM представила новые каналы, реализовав технологию ESCON (Enterprise System Connectivity), и начала интеграцию сетевого адаптера в мэйнфрейм – Open System Adapter (OSA).

В 1998 году IBM представила новый модуль, способный преодолеть барьер в 1000 миллионов команд в секунду, вследствие чего он стал одним из наиболее мощных

мэйнфреймов в мире. В тот же период концепция логического раздела была расширена таким образом, чтобы поддерживать 15 разделов.

В 1999 году на System/390 была реализована технология CUoD (Capacity Upgrade on Demand, повышение производительности по запросу). CUoD предполагает наличие дополнительных резервных процессоров, которые можно «включить» в случае необходимости. Эта технология представляет собой критически важный инструмент, позволяющий улучшить работу в период пиковых нагрузок и справляться с неожиданными изменениями.

В том же 1999 году компания IBM представила первый промышленный сервер, использующий передовую технологию «copper chip» компании IBM. Благодаря совместным усилиям, удалось расширить способность клиентов обрабатывать миллионы электронных коммерческих транзакций и обеспечить масштабируемость приложений планирования бизнес-ресурсов (ERP). В то же время появилась новая концепция, предполагающая возможность увеличения мощности машин без их остановки.

Появилась новая оптоволоконная технология FICON, мощность которых в восемь раз превышала мощность ESCON-каналов. Кроме того, в 1999 году на System/390 впервые появился Linux.

В октябре 2000 года компания IBM представила первое поколение мэйнфреймов zSeries. z/Architecture является расширением ESA/390 и поддерживает 64-разрядную адресацию. Кроме того, было реализовано управление динамическими каналами, а также специальные криптографические возможности. Мэйнфрейм стал «открытым» и способным работать под управлением Linux; были разработаны специальные процессоры (IFL).

В 2000 году появилась система z900, которая стала первым сервером компании IBM, полностью предназначенным для работы в сфере электронной коммерции.

Рис. А.6. z900

За z900 последовала z990. Система z990 достигла показателя 9000 миллионов команд в секунду; высокая масштабируемость еще больше увеличилась с ростом количества доступных логических разделов с 15 до 30 LPAR. Все еще существует предел

в 256 каналов на один образ операционной системы, однако z990 может содержать 1024 каналов, распределенных в четырех логических канальных подсистемах (Logical Channel SubSystem, LCSS). Современная модель предлагает также IFL – специальный процессор для выполнения Linux, и zAAP для обработки Java.

Рис. А.7. z990

zSeries основан на 64-разрядной архитектуре z/Architecture, разработанной таким образом, чтобы уменьшить количество узких мест памяти и дисковых систем хранения, и способный автоматически направлять ресурсы на приоритетные задачи посредством Intelligent Resource Director (IRD). IRD является основным средством в z/Architecture. Технологии Parallel Sysplex и IRD совместно обеспечивают гибкость и быстрое реагирование, необходимые для выполнения бизнес-задач по требованию.

z990 имеет модульную (многокнижную) структуру системы, поддерживающую конфигурацию, содержащую от одной до четырех *книг* (*books*). Каждая книга содержит: модуль MCM (Multiple Chip Module) с 12 процессорами, восемь из которых можно настроить как стандартные процессоры; карты памяти, которые могут поддерживать до 64 ГБ памяти на книгу; и высокопроизводительные самосинхронизирующиеся соединения (Self-Timed Interconnects). Максимальное число процессоров, доступных в системе z990, составляет 32.

Для поддержки многокнижной структуры системы с высокой масштабируемостью, канальная подсистема (Channel SubSystem, CSS) была расширена логическими канальными подсистемами (Logical Channel SubSystem, LCSS), позволяющими установить до 1024 ESCON-каналов в трех каркасах ввода/вывода (I/O cages). Поддержка Spanned Channel позволяет реализовать совместное использование HiperSockets™, ICB, ISC-3, OSA-Express и FICON Express в LCSS, что обеспечивает дополнительную гибкость. Высокоскоростные соединения для TCP/IP-связи, называемые HiperSockets, позволяют направлять TCP/IP-трафик между разделами и виртуальными серверами со скоростью памяти, а не со скоростью сети.

Самое последнее поколение мэйнфреймов, IBM System z9 109 (также называемое z9-109), является следующим этапом эволюции семейства мэйнфреймов IBM. Эти

мэйнфреймы используют z/Architecture и набор инструкций (с некоторыми дополнениями) серверов z900 и z990 (эта архитектура, ранее называвшаяся архитектурой ESAME, широко известна как 64-разрядная архитектура, хотя она обеспечивает гораздо больше, чем просто 64-разрядную адресацию). Внешне серверы z9-109 и z990 очень похожи. Однако помимо расширения технологии zSeries, сервер z9-109 содержит усовершенствования производительности, масштабируемости, доступности, безопасности и виртуализации.

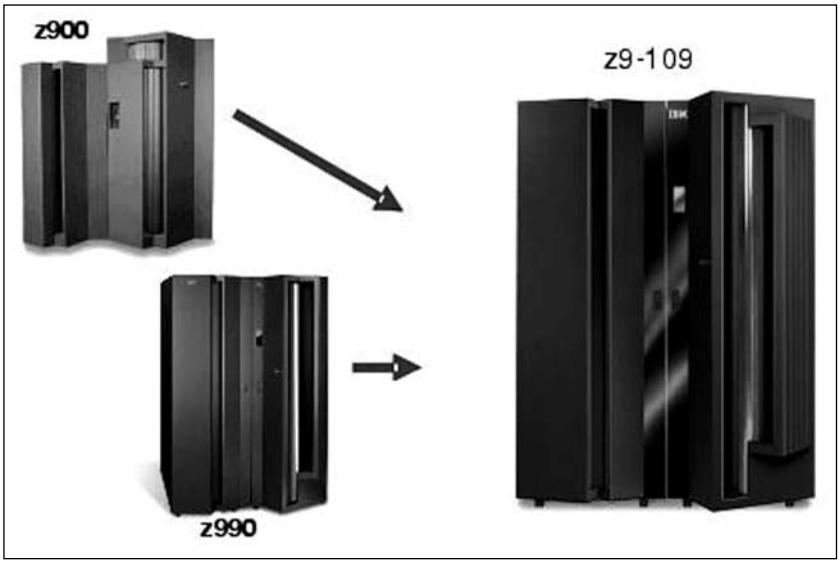


Рис. А.8. z9-109

К примерам дальнейшей эволюции мэйнфреймов, отраженной в z9-109, относятся:

- модульная многокнижная структура, поддерживающая от одной до четырех книг и до 54 процессорных устройств (используемых клиентами) на сервер;
- полная поддержка 64-разрядной основной и виртуальной памяти; для любого логического раздела может быть определена 31-разрядная или 64-разрядная адресуемость;
- до 512 ГБ системной памяти;
- до 60 логических разделов.

В предыдущих поколениях мэйнфреймов количество устройств ввода-вывода в системе было ограничено количеством каналов, количеством устройств управления на каждом канале и количеством устройств на каждом канале. Структура адресации также налагала ограничение. Фиксированные трехбайтовые адреса (по одному байту для канала, устройства управления и устройства), использовавшиеся в ранних системах, превратились в четырехбайтовые номера устройств, позволяющие задавать адреса устройств почти до 64 КБ. Сервер z9-109 продолжил это развитие путем реализации технологии Multiple Subchannel Sets (MSS), позволяющей задавать адреса устройств почти до 128 КБ.

Производительность каналов прошла развитие от параллельных каналов до ESCON-каналов и до FICON-каналов. Сервер z9-109 продолжил это развитие путем реализации значительно более высокопроизводительной опции программирования каналов.

Нагрузка на сервер была частично снята посредством использования отдельных процессоров, например, SAP, ICF, IFL и zAAP. Сервер z9-109 улучшает управление путем предоставления отдельных пулов для обработки разделяемых ICF, IFP и zAAP в PR/SM.

Базовые «реальные» системы развились в виртуальные системы, и это развитие относится к системам, процессорам, памяти, устройствам ввода-вывода, интерфейсам локальной сети и т. д. Сервер z9-109 продолжает развитие в этом направлении путем использования новых инструкций, повышающих производительность QDIO-операций виртуальной машины. Это достигается путем создания архитектуры транзитной передачи, позволяющей уменьшить издержки, связанные с программированием хостов, избегая остановки обработки при возникновении прерываний адаптера.

В последних поколениях мэйнфреймов был расширен набор инструкций; он включает инструкции, совместимые с другими платформами (например, инструкции двоичных вычислений с плавающей точкой), инструкции для более эффективной реализации популярных языков (например, инструкции обработки текстовых строк в C/C++), инструкции для более эффективного использования реестра (например, относительные инструкции и инструкции с непосредственными адресами, а также инструкции с длинным смещением) и т. д. Сервер z9-109 продолжил это развитие путем реализации новых и измененных инструкций.

Криптографические аппаратные средства были доступны и в ранних системах, но в последних моделях серверов они были значительно доработаны. Сервер z9-109 продолжает развитие криптографического оборудования путем расширения функций основных криптографических инструкций, а также путем консолидации опций (сопроцессора защиты и ускорителя) в единое устройство. Эти две опции могут быть отдельно определены в устройстве.

Прозрачное аппаратное восстановление всегда было основой структуры мэйнфрейма и получило развитие во многих направлениях. Сервер z9-109 продолжил это развитие путем расширения функций прозрачного восстановления таким образом, чтобы они включали пути от каркасов ввода-вывода к системной памяти.

Параллельное обслуживание является основной целью проектирования современных мэйнфреймов и часто включает поиск баланса между реплицируемыми компонентами и дополнительной интеграцией микросхем и MCM. Сервер z9-109 позволяет осуществлять параллельное удаление и переустановку одной книги в многокнижной конфигурации во время обновления или ремонта.

B

Примеры таблиц DB2

Большинство примеров в главе 12 «Системы управления базами данных в z/OS» ссылаются на таблицы из этого приложения. Вместе эти таблицы содержат информацию о сотрудниках и отделах и составляют образец приложения, иллюстрирующий большинство функций DB2.

Таблица отделов (DEPT)

Таблица отделов описывает каждый отдел предприятия, определяет, кто является менеджером отдела, и указывает отдел, перед которым он отчитывается. Таблица находится в табличном пространстве DSN8D81A.DSN8S81D и создается следующим кодом:

```
CREATE TABLE DSN8810.DEPT
      (DEPTNO    CHAR(3)          NOT NULL,
       DEPTNAME  VARCHAR(36)      NOT NULL,
       MGRNO     CHAR(6)                  ,
       ADMRDEPT  CHAR(3)          NOT NULL,
       LOCATION  CHAR(16)                 ,
       PRIMARY KEY (DEPTNO)               )
  IN DSN8D81A.DSN8S81D
  CCSID EBCDIC;
```

Так как таблица является автореферентной и при этом также является частью цикла зависимостей, необходимо добавить к ней внешние ключи, используя следующие операторы:

```
ALTER TABLE DSN8810.DEPT
      FOREIGN KEY RDD (ADMRDEPT) REFERENCES DSN8810.DEPT
               ON DELETE CASCADE;

ALTER TABLE DSN8810.DEPT
      FOREIGN KEY RDE (MGRNO) REFERENCES DSN8810.EMP
               ON DELETE SET NULL;
```

Столбцы таблицы отделов

Столбец	Имя столбца	Описание
1	DEPTNO	Идентификатор отдела, первичный ключ
2	DEPTNAME	Имя, описывающее основные функции отдела
3	MGRNO	Номер сотрудника (EMPNO) для менеджера отдела
4	ADMRDEPT	Идентификатор отдела, перед которым отчитывается этот отдел; отдел высшего уровня отчитывается перед самим собой
5	LOCATION	Имя местоположения

Индексы таблицы отделов

Имя	По столбцу	Тип индекса
DSN8810.XDEPT1	DEPTNO	Первичный, возрастающий
DSN8810.XDEPT2	MGRNO	Возрастающий
DSN8810.XDEPT3	ADMRDEPT	Возрастающий

Содержимое таблицы отделов

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	----------
B01	PLANNING	000020	A00	----------
C01	INFORMATION CENTER	000030	A00	----------
D01	DEVELOPMENT CENTER	------	A00	----------
E01	SUPPORT SERVICES	000050	A00	----------
D11	MANUFACTURING SYSTEMS	000060	D01	----------
D21	ADMINISTRATION SYSTEMS	000070	D01	----------
E11	OPERATIONS	000090	E01	----------
E21	SOFTWARE SUPPORT	000100	E01	----------
F22	BRANCH OFFICE F2	------	E01	----------
G22	BRANCH OFFICE G2	------	E01	----------
H22	BRANCH OFFICE H2	------	E01	----------
I22	BRANCH OFFICE I2	------	E01	----------
J22	BRANCH OFFICE J2	------	E01	----------

Связь с другими таблицами

Таблица является автореферентной: для указания руководящего отдела используется идентификатор отдела. Данная таблица является родительской для:

- Таблицы сотрудников, через внешний ключ в столбце WORKDEPT.
- Таблицы проектов, через внешний ключ в столбце DEPTNO. Эта таблица является зависимой от таблицы сотрудников через внешний ключ в столбце MGRNO.

Таблица сотрудников (EMP)

Таблица сотрудников идентифицирует всех сотрудников по номеру сотрудника и содержит основные сведения о персонале. Таблица находится в многораздельном табличном пространстве DSN8D81A.DSN8S81E. Так как она имеет внешний ключ, ссылающийся на DEPT, в первую очередь необходимо создать таблицу DEPT и индекс по ее первичному ключу. Затем создается таблица EMP с использованием следующего кода:

```
CREATE TABLE DSN8810.EMP
      (EMPNO     CHAR(6)                          NOT NULL,
       FIRSTNME  VARCHAR(12)                      NOT NULL,
       MIDINIT   CHAR(1)                          NOT NULL,
       LASTNAME  VARCHAR(15)                      NOT NULL,
       WORKDEPT  CHAR(3)                                  ,
       PHONENO   CHAR(4)          CONSTRAINT NUMBER CHECK
                 (PHONENO >= '0000' AND
                  PHONENO <= '9999')                      ,
       HIREDATE  DATE                                     ,
       JOB       CHAR(8)                                  ,
       EDLEVEL   SMALLINT                                 ,
       SEX       CHAR(1)                                  ,
       BIRTHDATE DATE                                     ,
       SALARY    DECIMAL(9,2)                             ,
       BONUS     DECIMAL(9,2)                             ,
       COMM      DECIMAL(9,2)                             ,
       PRIMARY KEY (EMPNO)                                ,
       FOREIGN KEY RED (WORKDEPT) REFERENCES DSN8810.DEPT
                 ON DELETE SET NULL                       )
  EDITPROC  DSN8EAE1
  IN DSN8D81A.DSN8S81E
  CCSID EBCDIC;
```

Столбцы таблицы сотрудников

Столбец	Имя столбца	Описание
1	EMPNO	Номер сотрудника (первичный ключ)
2	FIRSTNAME	Имя сотрудника
3	MIDINIT	Инициал, соответствующий второму имени (отчеству)
4	LASTNAME	Фамилия сотрудника
5	WORKDEPT	Идентификатор отдела, в котором работает сотрудник
6	PHONENO	Номер телефона сотрудника
7	HIREDATE	Дата приема на работу
8	JOB	Должность сотрудника
9	EDLEVEL	Количество лет обучения в учебных заведениях

Столбец	Имя столбца	Описание
10	SEX	Пол сотрудника (M или F)
11	BIRTHDATE	Дата рождения
12	SALARY	Годовой оклад в долларах
13	BONUS	Годовая премия в долларах
14	COMM	Годовое комиссионное вознаграждение в долларах

Индексы таблицы сотрудников

Имя	По столбцу	Тип индекса
DSN8810.XEMP1	EMPNO	Первичный, многораздельный, возрастающий
DSN8810. XEMP2	WORKDEPT	Возрастающий

Связь с другими таблицами

Данная таблица является родительской для:

- Таблицы отделов через внешний ключ в столбце MGRNO.
- Таблицы проектов через внешний ключ в столбце RESPEMP. Эта таблица является зависимой от таблицы отделов через внешний ключ в столбце WORKDEPT.

C

Утилиты

Не существует какого-то формального определения относительно того, что является утилитой z/OS в настоящее время, однако в практическом употреблении только некоторые программы, поставляемые с z/OS, называются *утилитами* (*utilities*). С другой стороны, в среде UNIX многие стандартные команды считаются утилитами. К ним относят компиляторы, программы резервного копирования, фильтры и многие другие типы программ. В среде z/OS эти средства называют *приложениями* или *программами*, но не *утилитами*[1]. Различие состоит только лишь в терминологии, однако новых пользователей z/OS это может сбить с толку.

В z/OS утилиты являются пакетными программами (хотя команды ALLOC позволяют выполнять их в оперативном режиме TSO) и обычно имеют соответствующие требования к JCL. Это включает DD-операторы SYSPRINT, SYSIN, SYSUT1 и SYSUT2. Большинство пользователей z/OS знакомы с IEFBR14, IEBGENER и IEBCOPY. Пользователи VSAM должны быть знакомы с IDCAMS.

Из широкого диапазона функций и возможностей z/OS только небольшое количество является системными утилитами. Это привело к появлению большого числа утилит, написанных клиентами (хотя большинство пользователей воздерживаются от того, чтобы называть их *утилитами*), многие из которых широко используются сообществом пользователей. Независимые изготовители программного обеспечения также предлагают множество подобных продуктов (платных). Некоторые из них можно отнести к утилитам; некоторые из них конкурируют с утилитами IBM, тогда как другие предоставляют функции, не реализованные в утилитах IBM.

[1] В z/OS UNIX термин «утилита» имеет значение, общее для UNIX.

Большинство рассматриваемых здесь основных и системных утилит описывается в публикации *z/OS DFSMSdfp Utilities*. В этом приложении приводится обзор доступных утилит вместе с простыми примерами самых основных утилит.

Основные утилиты

Некоторые утилиты (в традиционном значении этого термина) широко используются в пакетных заданиях. Ниже они описываются более подробно.

IEFBR14

Единственная функция этой программы состоит в том, чтобы выдавать нулевой код завершения (0). Она используется в качестве безопасного средства «выполнения JCL». Само понятие выполнения JCL считается некорректным, однако оно очень хорошо передает идею. Например, рассмотрим следующее задание:

```
//OGDEN1 JOB 1,BILL,MSGCLASS=X
// EXEC PGM=IEFBR14
//A DD DSN=OGDEN.LIB.CNTL,DISP=(NEW,CATLG),VOL=SER=WORK02,
// UNIT=3390,SPACE=(CYL,(3,1,25)
//B DD DSN=OGDEN.OLD.DATA,DISP=(OLD,DELETE)
```

Это полезное задание, хотя выполняемая программа (IEFBR14) не делает ничего. При подготовке к запуску задания инициатор распределяет OGDEN.LIB.CNTL и сохраняет набор данных по завершении задания. Он также удаляет OGDEN.OLD.DATA по завершении задания. DD-имена A и B не имеют особого значения и используются потому, что синтаксис оператора DD требует указания DD-имени.

Те же операции создания одного набора данных и удаления другого можно было выполнить, например, через ISPF, но эти действия может потребоваться выполнить в составе крупной последовательности пакетных заданий.

> **Примечание.** Имя IEFBR14 представляет интерес. Одна группа в IBM, создававшая первоначальный код для OS/360, использовала префикс IEF для всех своих модулей. В ассемблере BR означает переход (Branch) к адресу в регистре (Register). Переход к адресу в общем регистре 14 является стандартным способом завершения программы. Хотя, выбирая имя, разработчики не проявили особой изобретательности, практически все специалисты в z/OS с легкостью запоминают имя IEFBR14.

Программа IEFBR14 не является утилитой в том смысле, что она не включена в руководство *Utilities*. Однако не существует другой практической категории для этой полезной программы, поэтому мы произвольно отнесли ее в категории утилит.

IEBGENER

Утилита IEBGENER копирует один последовательный набор данных в другой (помните, что раздел библиотечного набора данных можно использовать как последовательный набор данных). Она может также выполнять фильтрацию данных, изменять LRECL и BLKSIZE, генерировать записи и выполнять некоторые другие функции.

Однако чаще всего она используется просто для копирования наборов данных. Типичное задание может иметь следующий вид:

```
//OGDEN2 JOB 1,BILL,MSGCLASS=X
// PGM=IEBGENER
//SYSIN DD DUMMY
//SYSPRINT DD SYSOUT=X
//SYSUT1 DD DISP=SHR,DSN=BILL.SEQ.DATA
//SYSUT2 DD DISP=(NEW,CATLG),DSN=BILL.COPY.DATA,UNIT=3390,
// VOL=SER=WORK02,SPACE=(TRK,3,3))
```

IEBGENER требует использования четырех DD-операторов с DD-именами, описываемыми в примере. Оператор SYSIN DD используется для чтения управляющих параметров; при простых вариантах управляющие параметры не нужны, и можно использовать DD DUMMY. Оператор SYSPRINT используется для выдачи сообщений, получаемых из IEBGENER. Оператор SYSUT1 предназначен для ввода, а оператор SYSUT2 – для вывода. Этот пример считывает существующий набор данных и копирует его в новый набор данных.

Если выходной набор данных является новым, и если параметры DCB не заданы, IEBGENER копирует параметры DCB из входного набора данных (к параметрам DCB относятся LRECL, RECFM и BLKSIZE, как описывается в разделе 5.8 «Форматы записи наборов данных»).

Другой общий пример может иметь приблизительно следующий вид:

```
//OGDEN2 JOB 1,BILL,MSGCLASS=X
// PGM=IEBGENER
//SYSIN DD DUMMY
//SYSPRINT DD SYSOUT=X
//SYSUT2 DD DISP=OLD,DSN=BILL.TEST.DATA
//SYSUT1 DD *
    Это встроенные данные. Они могут занимать столько места,
    сколько требуется. Для приложения они представляются как
    LRECL=80, RECFM=F, BLKSIZE=80. Вам лучше распределить набор
    данных SYSUT2 с более подходящим размером блока.
/*
```

Этот пример предполагает, что набор данных BILL.TEST.DATA уже был создан. Это задание перезапишет его данными из входного потока SYSUT1. Так как выходной набор данных уже существует, IEBGENER будет использовать существующие атрибуты DCB.

IEBGENER является простейшей программой копирования или вывода, входящей в z/OS. Она существует со времен первой версии OS/360.

IEBCOPY

Эта утилита обычно используется в следующих целях:

- Для копирования выбранных (или всех) разделов из одного библиотечного набора данных в другой.
- Для копирования библиотечного набора данных в уникальный последовательный формат *выгруженного* (*unloaded*) библиотечного набора данных. Как и последовательный набор данных, его можно записать на магнитную ленту, отправить по FTP[1] или обрабатывать как простой последовательный набор данных.
- Для чтения выгруженного библиотечного набора данных (представляющего собой последовательный файл) и повторного создания первоначального библиотечного набора данных. Как вариант, можно использовать только выбранные разделы.
- Для *сжатия* библиотечных наборов данных (на месте) для восстановления потерянного пространства.

Большинство программных продуктов для z/OS распространяются как выгруженные секционированные наборы данных. Опции копирования ISPF (в том числе, опция 3.3) негласно используют IEBCOPY. Перемещение PDS или PDSE с одного тома на другой с легкостью выполняется утилитой IEBCOPY. При возникновении необходимости осуществлять управление библиотечными наборами данных в пакетных заданиях, чаще всего используется IEBCOPY. При выполнении аналогичных операций в среде TSO (через ISPF) утилита IEBCOPY используется косвенным образом.

Простое задание с IEBCOPY может иметь следующий вид:

```
//OGDEN5 JOB 1,BILL,MSGCLASS=X
// EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//SYSUT1 DD DISP=SHR,DSN=OGDEN.LIB.SOURCE
//SYSUT2 DD DISP=(NEW,KEEP),UNIT=TAPE,DSN=OGDENS.SOURCE,
//           VOL=SER=123456
```

Это задание выгружает OGDEN.LIB.SOURCE (который, предположительно, является библиотечным набором данных) и записывает его на магнитную ленту (имя TAPE, предположительно, является *групповым именем*, которое локальная инсталляция связывает с приводами для магнитных лент). По умолчанию, IEBCOPY осуществляет копирование из SYSUT1 в SYSUT2. Обратите внимание на то, что имя набора данных на ленте не совпадает с именем набора данных, используемым на входе (можно было использовать одинаковое имя, но это не является обязательным требованием). Для восстановления PDS на другом томе можно использовать следующее задание:

[1] Выходной набор данных обычно имеет формат V или VB, и существуют дополнительные аспекты, которые следует учитывать при передаче наборов данных V или VB через FTP.

```
//JOE6 JOB 1,JOE,MSGCLASS=X
// EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//SYSUT1 DD DISP=OLD,UNIT=TAPE,DSN=OGDENS.SOURCE,
VOL=SER=123456
//SYSUT2 DD DISP=(NEW,CATLG),DSN=P390Z.LIB.PGMS,UNIT=3390,
//         SPACE=(TRK,(10,10,20)),VOL=SER=333333
```

В этом примере IEBCOPY обнаружит, что входной набор данных является выгруженным библиотечным набором данных. Нам потребовалась внешняя информация, чтобы определить, что набор данных займет около 10 дорожек и будет иметь 20 блоков оглавления.

Вместо DD DUMMY для SYSIN можно было использовать следующее:

```
//SYSIN DD *
COPY OUTDD=SYSUT2,INDD=SYSUT1
SELECT MEMBER=(PGM1,PGM2)
/*
```

Параметры OUTDD и INDD определяют используемые DD-имена. В данном случае мы просто использовали имена по умолчанию, но это не является обязательным требованием.

Восстановление библиотечного набора данных с выгруженной копии автоматически сжимает набор данных (восстанавливая потерянное пространство).

IEBDG

Утилита IEBDG используется для создания записей, в которых можно генерировать поля с различными типами данных. IEBDG обычно используется для создания тестовых данных. Можно генерировать разнообразные поля и данные, и поля для каждой записи можно изменять путем циклического, колебательного, вращательного сдвига, прокрутки и прочих изменений полей. IEBDG может принимать входящие записи данных и выполнять наложение заданных полей входных данных на сгенерированные данные.

Ниже представлен простой пример использования IEBDG:

```
//OGDEN7 JOB 1,BILL,MSGCLASS=X
// EXEC PGM=IEBDG
//SYSPRINT DD SYSOUT=*
//OUT DD DISP=(NEW,CATLG),DSN=OGDEN.TEST.DATA,UNIT=3390,
//         VOL=SER=WORK01,SPACE=(CYL,(10,1)),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000)
//SYSIN DD *
  DSD OUTPUT=(OUT)
```

```
  FD NAME=FIELD1,LENGTH=30,FORMAT=AL,ACTION=RP
  FD NAME=FIELD2,LENGTH=10,PICTURE=10,'TEST DATA '
  FD NAME=FIELD3,LENGTH=10,FORMAT=RA
  CREATE QUANTITY=90000,NAME=(FIELD1,FIELD2,FIELD3)
  END
/*
```

Это задание создает новый набор данных OGDEN.TEST.DATA, содержащий 90000 записей. Каждая запись имеет длину 80 байт, что задано в параметрах DCB в DD-операторе. Управляющие операторы определяют три поля, занимающих первые 50 байт каждой записи. По умолчанию IEBDG заполняет остальные байты двоичными нулями. Этими тремя полями являются:

- Алфавитное поле ('ABCDEF...') длиной 30 байт. Оно циклически сдвигается на один байт влево после генерирования каждой записи.
- Второе поле содержит 10 байт с фиксированной константой 'TEST DATA '.
- Третье поле содержит 10 байт случайных двоичных данных.

Утилита может генерировать более сложные образцы, но приведенный выше пример является типичным примером простого использования. Он также иллюстрирует определение количества дискового пространства, необходимого для хранения данных:

- Нам известно, что дорожка 3390 содержит около 57 КБ и еще меньше из-за пространства, теряемого в промежутках между записями.
- Нам известны параметры DCB (заданные в JCL): LRECL=80, BLKSIZE=8000 и RECFM=FB. Нам не известно, почему были выбраны эти параметры DCB, но, предположительно, они связаны с программой, которая будет использовать тестовые данные.
- Мы можем определить, что шесть блоков по 8000 байт в каждом, вероятно, поместятся на одной дорожке. Такой размер блока не очень эффективен, так как некоторое пространство дорожки будет потеряно, однако он является достаточно практичным.
- Каждый блок содержит 100 записей по 80 байт в каждой. Каждая дорожка содержит 600 записей (в блоке FB-записей не происходит потери пространства).
- Цилиндр содержит 15 дорожек, следовательно цилиндр может содержать 9000 записей.
- Исходя из вышесказанного, нам нужно 10 цилиндров для содержания 90000 записей. Мы задали 10 цилиндров как пространство первичного распределения в JCL и один цилиндр как приращение при вторичном распределении. Нам вряд ли потребуется вторичное распределение, однако это обеспечивает некоторый запас надежности. Мы могли запросить 150 дорожек, а не 10 цилиндров; результат был бы такой же.

IDCAMS

Программа IDCAMS не относится к основному набору утилит z/OS, описанному в руководстве *Utilities*. Программа IDCAMS используется, главным образом, для создания и управления наборами данных VSAM. Она имеет и другие функции (например, изменение каталога), но чаще всего она связана с VSAM. Она содержит множество сложных функций, описание которых заняло бы отдельные руководства. На момент написания данной книги основным руководством от компании IBM является *DFSMS Access Method Services for Catalogs*.

Типичный пример простого использования IDCAMS:

```
//OGDEN12 JOB 1,BILL,MSGCLASS=X
//DEL EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  DELETE OGDEN.DATA.VSAM CLUSTER
/*
//LOAD EXEC PGM=IDCAMS
//SYSPRINT DD *
//DATAIN DD DISP=OLD,DSN=OGDEN.SORTOUT
//SYSIN DD *
  DEFINE CLUSTER (NAME (OGDEN.DATA.VSAM) -
          VOLUMES(WORK02) CYLINDERS(1 1) -
          RECORDSIZE (72 100) KEYS(9 8) INDEXED)
  REPRO INFILE(DATAIN) OUTDATASET(OGDEN.DATA.VSAM) ELIMIT(200)
/*
```

Этот пример иллюстрирует множество моментов.

- Задание содержит два шага. Первый шаг удаляет набор данных, который будет создан вторым шагом. Это называется функцией очистки. Набор данных может не существовать на данном этапе, и первый шаг сгенерирует код завершения, указывающий на ошибку при выполнении этого действия, который в данном случае игнорируется.
- Обратите внимание на отсутствие операторов DD для набора данных VSAM. IDCAMS осуществляет динамическое распределение для создание требуемого JCL.
- Второй шаг выполняет две функции. Сначала он создает набор данных VSAM (командой DEFINE CLUSTER), после чего загружает его из последовательного набора данных (командой REPRO). Последовательный набор данных требует использования оператора DD.
- Команда DEFINE CLUSTER занимает три строки. Используются такие же индикаторы продолжения, что и в TSO-командах.

- Набор данных VSAM находится на томе WORK02 и использует один цилиндр в качестве первичного пространства и один цилиндр для вторичного распределения. Средний размер записи составляет 72 байт, а максимальный размер записи – 100 байт (наборы данных VSAM всегда используют записи переменной длины). Первичный ключ (для доступа к записям в наборе данных) имеет длину 8 байт и начинается со смещения 9 байт в каждой записи.
- Таким образом, записи для загрузки набора данных VSAM должны уже быть упорядочены по ключу.
- Параметр ELIMIT задает количество ошибочных записей, игнорируемых командой REPRO, прежде чем прервать операцию. Ошибочные записи обычно являются результатом дублирования значения ключа.

Многие из функций IDCAMS можно вводить как TSO-команды. Например, DEFINE CLUSTER можно использовать как TSO-команду. Однако обычно не рекомендуется этого делать, так как эти команды могут быть сложными, и могут возникнуть трудности с анализом ошибок. Ввод команд IDCAMS посредством пакетного задания позволяет просматривать команды и результирующие сообщения настолько часто, насколько это нужно, используя SDSF для просмотра выходных данных.

Примечание. Некоторые пользователи произносят имя данной программы как «ид камс» (в два слога), тогда как другие произносят «ай ди камс» (в три слога).

IEBUPDTE

Утилита IEBUPDTE используется для создания нескольких разделов в библиотечном наборе данных или для обновления записей в разделе. Хотя ее можно использовать для других типов записей, ее основное применение состоит в создании или обслуживании библиотек процедур JCL или библиотек макросов на ассемблере. В настоящее время эта утилита используется, главным образом, для распространения и обслуживания лицензированных программ z/OS. Ее редко используют пользователи TSO.

Простой пример выполняет добавление двух JCL-процедур в MY.PROCLIB. Это легко осуществляется через ISPF, но если предположить, что следующее задание было отправлено на магнитной ленте, его полезность становится более очевидной:

```
//OGDEN10 JOB 1,BILL,MSGCLASS=X
// EXEC PGM=IEBUPDTE
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DISP=OLD,DSN=MY.PROCLIB
//SYSUT2 DD DISP=OLD,DSN=MY.PROCLIB
//SYSIN DD DATA
./ ADD    LIST=ALL,NAME=MYJOB1
//STEP1 EXEC=BILLX1
//PRINT DD SYSOUT=A
//         (more JCL for MYJOB1)
```

```
//SYSUDUMP DD SYSOUT=* (last JCL for MYJOB1)
./  REPL   LIST=ALL,NAME=LASTJOB
//LIST EXEC PGM=BILLLIST
//         (дополнительный JCL-код этой процедуры)
//* LAST JCL STATEMENT FOR LASTJOB
./ ENDUP
/*
```

Этот пример требует некоторых комментариев.

- При обновлении библиотеки, SYSUT1 и SYSUT2 должны указывать на эту библиотеку (если они указывают на разные библиотеки, выполняется копирование библиотеки SYSUT1 в библиотеку SYSUT2 с последующим обновлением).
- Формат SYSIN DD DATA указывает, что данные во входном потоке содержат // в столбцах 1 и 2. Эти символы не следует интерпретировать как JCL. Конец входного потока обозначен символами /*.
- Утилита IEBUPDTE использует управляющие операторы, содержащие ./ в первых двух столбцах.
- Раздел MYJOB1 добавляется в MY.PROCLIB; этот раздел не обязательно должен уже существовать в библиотеке.
- Раздел LASTJOB заменяется новым содержимым.

Утилита IEBUPDTE может также добавлять или заменять операторы в разделе на основе *порядковых номеров* в операторах. Это представляет один из примеров использования порядковых номеров в JCL или операторах исходного кода.

Опять же заметим, что IEBUPDTE обычно используется для распространения и обслуживания программ. Например, если программный продукт добавляет 25 JCL-процедур в библиотеку процедур клиента, изготовитель программного продукта может упаковать процедуры как задание IEBUPDTE. Одно из преимуществ этого способа состоит в том, что весь материал имеет исходный формат, и клиент может с легкостью просмотреть его содержимое до запуска задания.

Системно-ориентированные утилиты

Программы, рассматриваемые в этом разделе, выполняют ряд базовых функций для системных администраторов и лишь кратко описываются.

IEHLIST

Утилита IEHLIST используется для вывода оглавления библиотечного набора данных или оглавления дискового тома. Она обычно используется для вывода листингов оглавления тома и предоставляет информацию на уровне битов. В большинстве инсталляций утилита IEHLIST используется редко, однако она необходима в случаях повреждения оглавления тома. Иногда она используется вместе с программой SUPERZAP для исправления поврежденного оглавления тома.

IEHINITT

Утилита IEHINITT используется для записи стандартных меток на магнитные ленты. При необходимости ее можно использовать для разметки одной ленты или для разметки больших пакетов магнитных лент. Многие крупные инсталляции z/OS не позволяют использовать неразмеченные ленты в инсталляции.

IEHPROGM

Утилита IEHPROGM, в сущности, уже устарела. Она используется, главным образом, для управления каталогами, переименования наборов данных и удаления наборов данных через программу, а не через JCL-операции. Она чаще использовалась во время установки системы или крупного программного продукта. Эти функции могут включать десятки (или даже сотни) операций с каталогами и наборами данных. Заблаговременная подготовка команд (в пакетном задании IEHPROGM) гораздо менее подвержена ошибкам, чем динамические подходы. Большая часть функций IEHPROGM доступна в IDCAMS, и эта утилита теперь является более предпочтительной для работы с каталогами и наборами данных.

ICKDSF

Утилита ICKDSF используется, главным образом, для инициализации дисковых томов. Это, как минимум, включает создание записи метки диска и оглавления тома. ICKDSF может также выполнить сканирование тома, чтобы убедиться в том, что он пригоден для использования, переформатировать все дорожки, записать домашние адреса и записи R0 и т. д.

SUPERZAP

Программу SUPERZAP (ее имя менялось несколько раз) можно использовать для исправления дисковых записей. Она понимает формат исполняемых модулей в библиотеках PDS, что необходимо при ее использовании в своем основном назначении при применении обновлений для таких исполняемых модулей. В настоящее время утилита SUPERZAP редко используется для системного обслуживания; она гораздо чаще использовалась в более ранних версиях операционной системы.

Утилита SUPERZAP используется для исправления оглавлений томов, исполняемых программ, а также почти любой дисковой записи. На практике она преимущественно используется для исправления исполняемых программ. Она широко использовалась в прежние времена для установки небольших исправлений в программах.

Рассмотрим, например, новую версию продукта XXX. Новая версия могла быть отправлена на магнитной ленте сотням или тысячам клиентов. После доставки всех этих лент разработчики могли обнаружить небольшую ошибку, которую можно исправить изменением нескольких инструкций. Вместо того чтобы создавать новые дистрибутивные ленты и их доставлять всем клиентам (что является трудоемким и дорогостоящим мероприятием для крупного программного продукта), разработчики могут создать решение на основе SUPERZAP и отправить его по почте/факсу/ftp своим клиентам.

Утилита SUPERZAP работает на уровне битов. Ее использование является целесообразным, когда нужно изменить относительно небольшое число битов или байтов. Пример SUPERZAP-обновления представлен ниже:

```
//OGDEN15 JOB 1,BILL,MSGCLASS=X
// EXEC PGM=AMASPZAP
//SYSPRINT DD SYSOUT=*
//SYSLIB DD DISP=OLD,DSN=OGDEN.LIB.LOAD
//SYSIN DD *
   NAME QSAM1
   VERIFY 004E 4780
   REP 004E 4700
/*
```

Оператор SYSLIB DD должен указывать на набор данных, содержащий модуль, требующий изменения. Управляющий оператор NAME указывает исполняемый модуль (представленный именем раздела PDS), который нужно изменить. Оператор VERIFY указывает, что нужно перейти к адресу со смещением x'004E' в модуле и убедиться в том, что он содержит x'4780'. В этом случае следует изменить модуль таким образом, чтобы он содержал x'4700' по адресу с этим смещением. Это заменит инструкцию Branch Equal на No Operation и, предположительно, изменит логику программы.

Мы смогли создать SUPERZAP-обновление таким образом, потому что у нас был листинг программы на ассемблере, и мы могли увидеть точное смещение в модуле, содержащем инструкцию, которую требовалось изменить. Без листинга это было бы гораздо сложнее, хотя листинг можно было восстановить путем чтения шестнадцатеричных дампов памяти и воссоздания машинных операций с дампов. Обратите внимание на то, что исполняемые программы на диске имеют сложный формат и не являются простым образом программы, загруженной в память (частично эта сложность обуславливается перемещением данных, внешними символами и оптимизированным форматом дисковой загрузки). SUPERZAP понимает этот дисковый формат и позволяет пользователям обновить исполняемую программу, как если бы она являлась образом в памяти.

Мы обсуждаем SUPERZAP, однако в данном примере используется программа AMASPZAP. Это текущее имя данной программы, хотя она все еще широко известна как SUPERZAP.

Утилиты уровня приложений

Существует множество приложений, которые можно считать утилитами. Мы кратко рассмотрим некоторые наиболее часто используемые из них. Они более сложны в применении, чем простые программы, которые мы рассматривали выше, и мы не приводим примеры их использования.

ADRDSSU

Эта программа является основной программой *создания дампов диска* и *восстановления дисков*, поставляемой с z/OS. Она способна выполнять фильтрацию и выбор наборов данных, подлежащих включению в дамп или восстановлению, однако чаще всего она используется для создание полного дампа диска. Цель создания дампа диска обычно состоит в создании резервной копии содержимого, с которой, при необходимости, можно выполнить восстановление. Часто с помощью этой программы создаются дампы полных томов, но восстанавливается только определенный набор данных, который был случайно поврежден.

Резервная копия обычно записывается на магнитную ленту, однако может быть записана и в дисковый набор данных. Дамп диска можно создавать по дорожкам (физический дамп) или по наборам данных (логический дамп). При выполнении логического дампа, несколько экстентов наборов данных совмещаются в один экстент, библиотечные наборы данных сжимаются, и все свободное пространство остается в одном экстенте.

RMF

Средство RMF (Resource Measurement Facility) является дополнительной лицензированной программой компании IBM, используемой для измерения различных аспектов производительности системы. Разные модули RMF осуществляют долгосрочный сбор статистики, фиксацию мгновенных данных, долгосрочное ведение отчетов, создание отчетов пакетного типа, создание TSO-ориентированных отчетов и т. д. Аппаратная система ввода-вывода поддерживает статистические счетчики времени нахождения в очереди для каждого устройства ввода-вывода, количества операций на устройство и прочей низкоуровневой информации. RMF осуществляет доступ к этим аппаратным счетчикам и включает их в свои отчеты.

D

Таблица EBCDIC - ASCII

Hx	Dec	E	A	Hx	Dec	E	A	Hx	Dec	E	A	Hz	Dec	E	A
00	00	NUL	20	32	SP	40	64	SP	@	60	96	-	'		
01	01	21	33	!	41	65	A	61	97	/	a				
02	02	22	34	"	42	66	B	62	98	b					
03	03	23	35	#	43	67	C	63	99	c					
04	04	24	36	$	44	68	D	64	100	d					
05	05	25	37	%	45	69	E	65	101	e					
06	06	26	38	&	46	70	F	66	102	f					
07	07	27	39	'	47	71	G	67	103	g					
08	08	28	40	(	48	72	H	68	104	h					
09	09	29	41	)	49	73	I	69	105	i					
0A	10	2A	42	*	4A	74	^	J	6A	106	|	j			
0B	11	2B	43	+	4B	75	.	K	6B	107	,	k			
0C	12	2C	44	,	4C	76	<	L	6C	108	%	l			
0D	13	2D	45	-	4D	77	(	M	6D	109	_	m			
0E	14	2E	46	.	4E	78	+	N	6E	110	>	n			
0F	15	2F	47	/	4F	79	|	O	6F	111	?	o			
10	16	30	48	0	50	80	&	P	70	112	p				
11	17	31	49	1	51	81	Q	71	113	q					
12	18	32	50	2	52	82	R	72	114	r					
13	19	33	51	3	53	83	S	73	115	s					
14	20	34	52	4	54	84	T	74	116	t					
15	21	35	53	5	55	85	U	75	117	u					
16	22	36	54	6	56	86	V	76	118	v					

Hx	Dec	E	A	Hx	Dec	E	A	Hx	Dec	E	A	Hz	Dec	E	A
17	23	37	55	7	57	87	W	77	119	w					
18	24	38	56	8	58	88	X	78	120	x					
19	25	39	57	9	59	89	Y	79	121	y					
1A	26	3A	58	:	5A	90	!	Z	7A	122	:		z		
1B	27	3B	59	;	5B	91	$	[	7B	123	#		{		
1C	28	3C	60	<	5C	92	*	\	7C	124	@		|		
1D	29	3D	61	=	5D	93	)	]	7D	125	'		}		
1E	30	3E	62	>	5E	94	;	^	7E	126	=		~		
1F	31	3F	63	?	5F	95	not	_	7F	127	"				
80	128	A0	160	C0	192	{	E0	224	\						
81	129	a	A1	161	C1	193	A	E1	225						
82	130	b	A2	162	s	C2	194	B	E2	226	S				
83	131	c	A3	163	t	C3	195	C	E3	227	T				
84	132	d	A4	164	u	C4	196	D	E4	228	U				
85	133	e	A5	165	v	C5	197	E	E5	229	V				
86	134	f	A6	166	w	C6	198	F	E6	230	W				
87	135	g	A7	167	x	C7	199	G	E7	231	X				
88	136	h	A8	168	y	C8	200	H	E8	232	Y				
89	137	i	A9	169	z	C9	201	I	E9	233	Z				
8A	138	AA	170	CA	202	EA	234								
8B	139	AB	171	CB	203	EB	235								
8C	140	AC	172	CC	204	EC	236								
8D	141	AD	173	[	CD	205	ED	237							
8E	142	AE	174	CE	206	EE	238								
8F	143	AF	175	CF	207	EF	239								
90	144	B0	176	D0	208	}	F0	240	0						
91	145	j	B1	177	D1	209	J	F1	241	1					
92	146	k	B2	178	D2	210	K	F2	242	2					
93	147	l	B3	179	D3	211	L	F3	243	3					
94	148	m	B4	180	D4	212	M	F4	244	4					
95	149	n	B5	181	D5	213	N	F5	245	5					
96	150	o	B6	182	D6	214	O	F6	246	6					
97	151	p	B7	183	D7	215	P	F7	247	7					
98	152	q	B8	184	D8	216	Q	F8	248	8					
99	153	r	B9	185	D9	217	R	F9	249	9					
9A	154	BA	186	DA	218	FA	250								
9B	155	BB	187	DB	219	FB	251								
9C	156	BC	188	DC	220	FC	252								
9D	157	BD	189	]	DD	221	FD	253							
9E	158	BE	190	DE	222	FE	254								
9F	159	BF	191	DF	223	FF	255								

E

Программа для практической работы

Все приведенные здесь примеры работают с файлом (или базой данных) сотрудников; этот файл идентифицирует всех сотрудников по номеру сотрудника и содержит основную информацию о персонале.

Упражнение использует номер отдела как входные данные, выбирает все записи по этому отделу, после чего вычисляет сумму полей оклада в этих записях. В результате, выводится средний оклад.

Следующие упражнения написаны на разных языках, выполняются в разных средах и используют разные источники данных, но все они выполняют функции, описанные выше. Упражнения содержат код, подготовительные задания и инструкции.

Предполагается, что студенты установили соответствующий эмулятор 3270 и имеют требуемые полномочия в TSO, CICS, DB2 и WebSphere для z/OS. Обратите внимание на системные определения (в частности, на HLQ, имя базы данных DB2 и т. д.), которые могут потребоваться в каждом упражнении.

Программа COBOL-CICS-DB2

Исходный код

Определение карты

Это определение находится в разделе TMAP01 библиотеки LUISM.TEST.SAMPLIB.

```
       PRINT       NOGEN
TMAPSET    DFHMSD TYPE=&SYSPARM, x
                  LANG=COBOL, x
                  MODE=INOUT, x
                  TERM=3270-2, x
                  CTRL=FREEKB, x
                  STORAGE=AUTO, x
                  TIOAPFX=YES
TMAP01     DFHMDI SIZE=(24,80), x
                  LINE=1, x
                  COLUMN=1, x
                  MAPATTS=COLOR
           DFHMDF POS=(1,1), x
                  LENGTH=9, x
                  ATTRB=(NORM,PROT), x
                  COLOR=BLUE, x
                  INITIAL='ABCD txid'
           DFHMDF POS=(1,26), x
                  LENGTH=28, x
                  ATTRB=(NORM,PROT), x
                  COLOR=GREEN, x
                  INITIAL='Average salary by department'
           DFHMDF POS=(9,1), x
                  LENGTH=41, x
                  ATTRB=(NORM,PROT), x
                  INITIAL='Type a department number and press enter.'
           DFHMDF POS=(11,1), x
                  LENGTH=18, x
                  ATTRB=(NORM,PROT), x
                  COLOR=GREEN, x
                  INITIAL='Department number:'
DPTONO     DFHMDF POS=(11,20), x
                  LENGTH=3, x
```

```
                ATTRB=(NORM,UNPROT,IC), x
                COLOR=TURQUOISE, x
                INITIAL='___'
          DFHMDF POS=(11,24), x
                LENGTH=1, x
                ATTRB=ASKIP
          DFHMDF POS=(13,1), x
                LENGTH=18, x
                ATTRB=(NORM,PROT), x
                COLOR=GREEN, x
                INITIAL='Average salary($):'
AVGSAL    DFHMDF POS=(13,20), x
                LENGTH=11, x
                ATTRB=(NORM,PROT), X
                COLOR=TURQUOISE
MSGLINE   DFHMDF POS=(23,1), x
                LENGTH=78, x
                ATTRB=(BRT,PROT), x
                COLOR=BLUE
          DFHMDF POS=(23,79), x
                LENGTH=1, x
                ATTRB=(DRK,PROT,FSET), x
                INITIAL=' '
          DFHMDF POS=(24,1), x
                LENGTH=7, x
                ATTRB=(NORM,PROT), x
                COLOR=RED, x
                INITIAL='F3=Exit'
          DFHMSD TYPE=FINAL
END
```

Код программы

Эта программа находится в разделе XYZ2 библиотеки LUISM.TEST.SAMPLIB.

```
IDENTIFICATION DIVISION.
    *-------------------------------------------------------------
    * COBOL-CICS-DB2 PROGRAM ZSCHOLAR RESIDENCY
    * OBTAINS THE AVERAGE SALARY OF EMPLOYEES OF A GIVEN DEPART.
    *-------------------------------------------------------------
    *-----------------------
    PROGRAM-ID. XYZ2.
```

```
/
ENVIRONMENT DIVISION.
*--------------------
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
DATA DIVISION.
*-------------
FILE SECTION.
/
*-------------
WORKING-STORAGE SECTION.
************************************************
* WORKAREAS *
************************************************
01 SWITCH.
05 DATA-IS PIC X VALUE 'Y'.
    88 DATA-IS-O VALUE 'Y'.
05 SEND-IND PIC X.
    88 SEND-IND-ERASE VALUE '1'.
    88 SEND-IND-DATAO VALUE '2'.
    88 SEND-IND-ALARM VALUE '3'.
01 COMMUNICATION-AREA PIC X.
01 MSGLINET.
   02 MSGSQLC PIC X(8).
   02 FILLER PIC X.
   02 MSGREST PIC X(69).
************************************************
* DB2 HOST VARIABLES DECLARATION *
************************************************
01 WORKDEPT-HV PIC X(3).
01 SALARY-HV PIC X(11).
01 SALARY-IN PIC S9(4) COMP-5.
************************************************
* SQLCA DECLARATION *
************************************************
       EXEC SQL INCLUDE SQLCA END-EXEC.
************************************************
* DFHAID *
************************************************
```

```
COPY DFHAID.
*****************************************************
* MAP COPY *
*****************************************************
COPY MAPONL.
*****************************************************
* DECLARE OF DB2 TABLE *
*****************************************************
       EXEC SQL
         DECLARE DSN8810.EMP TABLE
         (EMPNO CHAR(6) NOT NULL,
         FIRSTNAME VARCHAR(12) NOT NULL,
         MIDINIT CHAR(1) NOT NULL,
         LASTNAME VARCHAR(15) NOT NULL,
         WORKDEPT CHAR(3) ,
         PHONENO CHAR(4) ,
         HIREDATE DATE ,
         JOB CHAR(8) ,
         EDLEVEL SMALLINT ,
         SEX CHAR(1) ,
         BIRTHDATE DATE ,
         SALARY DECIMAL(9,2) ,
         BONUS DECIMAL(9,2) ,
         COMM DECIMAL(9,2) )
       END-EXEC.
************************************************************
LINKAGE SECTION.
************************************************************
01 DFHCOMMAREA PIC X.
/
PROCEDURE DIVISION USING DFHCOMMAREA.
************************************************************
* MAIN ROGRAM ROUTINE
************************************************************
MAINLINE.
************************************************************
* 2000-PROCESS
************************************************************
2000-PROCESS.
       EVALUATE TRUE
```

```
           WHEN EIBCALEN = ZERO
             MOVE LOW-VALUE TO TMAP01O
             SET SEND-IND-ERASE TO TRUE
             PERFORM 2000-10-SEND
           WHEN EIBAID = DFHCLEAR
             MOVE LOW-VALUE TO TMAP01O
             SET SEND-IND-ERASE TO TRUE
             PERFORM 2000-10-SEND
           WHEN EIBAID = DFHPA1 OR DFHPA2 OR DFHPA3
             CONTINUE
           WHEN EIBAID = DFHPF3
               EXEC CICS RETURN
             END-EXEC
             GOBACK
           WHEN EIBAID = DFHENTER
             PERFORM 2000-00-PROCESS
           WHEN OTHER
             MOVE LOW-VALUE TO TMAP01O
             MOVE 'WRONG KEY' TO MSGLINEO
             SET SEND-IND-ALARM TO TRUE
             PERFORM 2000-10-SEND
           END-EVALUATE.
*
           EXEC CICS RETURN TRANSID('ABCD')
             COMMAREA(COMMUNICATION-AREA)
           END-EXEC.
           GOBACK.
       2000-00-PROCESS.
           EXEC CICS RECEIVE MAP('TMAP01')
                 MAPSET('TMAPSET')
                 INTO(TMAP01I)
           END-EXEC.
           IF DPTONOL = ZERO OR DPTONOI = SPACE
             MOVE 'N' TO DATA-IS
             MOVE 'ENTER A VALID DEPARTMENT NUMBER' TO MSGLINEO
           END-IF.
           IF DATA-IS-O
             MOVE DPTONOI TO WORKDEPT-HV
             PERFORM 2000-01-DB2
           END-IF.
```

```
           IF DATA-IS-O
             SET SEND-IND-DATAO TO TRUE
             PERFORM 2000-10-SEND
           ELSE
             SET SEND-IND-ALARM TO TRUE
             PERFORM 2000-10-SEND
           END-IF.
*
  2000-01-DB2.
           EXEC SQL SELECT CHAR(DECIMAL(SUM(SALARY),9,2))
             INTO :SALARY-HV :SALARY-IN
             FROM DSN8810.EMP
             WHERE WORKDEPT=:WORKDEPT-HV END-EXEC.
           IF SQLCODE = 0
           THEN
             IF SALARY-IN = -1
             THEN
               MOVE 'N' TO DATA-IS
               MOVE 'NO EMPLOYEES EXIST IN THIS DEPARTMENT' TO MSGLINEO
               MOVE SPACES TO AVGSALO
             ELSE
               MOVE SALARY-HV TO AVGSALO
               MOVE SPACES TO MSGLINEO
             END-IF
             ELSE
               MOVE '0' TO DATA-IS
               MOVE SPACES TO AVGSALO
               MOVE 'SQLSTATE' TO MSGSQLC
               MOVE SQLSTATE TO MSGREST
               MOVE MSGLINET TO MSGLINEO
             END-IF.
*
    2000-10-SEND.
           EVALUATE TRUE
           WHEN SEND-IND-ERASE
             EXEC CICS SEND MAP('TMAP01')
               MAPSET('TMAPSET')
               FROM (TMAP01O)
               ERASE
             END-EXEC
```

```
WHEN SEND-IND-DATAO
   EXEC CICS SEND MAP('TMAP01')
      MAPSET('TMAPSET')
      FROM (TMAP01O)
      DATAONLY
   END-EXEC
WHEN SEND-IND-ALARM
   EXEC CICS SEND MAP('TMAP01')
      MAPSET('TMAPSET')
      FROM (TMAP01O)
      DATAONLY
     ALARM
END-EXEC
END-EVALUATE.
```

Подготовительные задания

Ассемблирование и компоновка карты

Это задание находится в разделе MAPASSEM библиотеки LUISM.TEST.SAMPLIB. Обе вызываемые процедуры находятся в SYS1.PROCLIB.

```
//LUISM01 JOB (999,POK),'BMS Compilation',
// CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1)
//*************************************************************
//*       ASSEMBLE MAP SET *
//*************************************************************
//STEP01   EXEC PROC=DFHASMV1,PARM.ASSEM='SYSPARM(MAP)'
//SYSLIN   DD DSN=LUISM.OBJETO,DCB=(LRECL=80),
//         SPACE=(2960,(10,10)),UNIT=SYSDA,DISP=(NEW,PASS)
//SYSIN    DD DSN=LUISM.TEST.SAMPLIB(TMAP01),DISP=SHR
/*
//*************************************************************
//*       LINK EDIT *
//*************************************************************
//STEP02   EXEC PROC=DFHLNKV1,PARM='LIST,LET,XREF'
//SYSLIN   DD DSN=LUISM.OBJETO,DISP=(OLD,DELETE)
//         DD *
           MODE RMODE(ANY)
           NAME TMAPSET(R)
/*
```

Генерирование файла копии карты

Это задание находится в разделе MAPCOPYGM библиотеки LUISM.TEST.SAMPLIB.

```
//LUISM02  JOB (999,POK),'BMS COPY',
//         CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1)
//*************************************************************
//*        MAP COPY GENERATION *
//*************************************************************
//STEP01   EXEC PROC=DFHASMV1,PARM.ASSEM='SYSPARM(DSECT)'
//SYSLIN   DD DSN=LUISM.TEST.SAMPLIB(MAPCOPY),DISP=OLD
//SYSIN    DD DSN=LUISM.TEST.SAMPLIB(TMAP01),DISP=SHR
/*
```

Подготовка программы

Это задание находится в разделе CICSDB2P библиотеки LUISM.TEST.SAMPLIB.

```
//LUISM03  JOB (999,POK),'Cobol-CICS-DB2',
//         CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1)
//*************************************************************
//* DB2 precompile, CICS translation, COBOL compile, pre-link,
//* and link edit. Also DB2 Bind. *
//*************************************************************
//*************************************************************
//*        DB2 Precompile *
//*************************************************************
//PC       EXEC PGM=DSNHPC,PARM='HOST(IBMCOB)'
//SYSIN    DD DSN=LUISM.TEST.SAMPLIB(XYZ2),DISP=SHR
//DBRMLIB  DD DISP=SHR,
//            DSN=DB8HU.DBRMLIB.DATA(XYZ2)
//STEPLIB  DD DISP=SHR,DSN=DB8H8.SDSNEXIT
//         DD DISP=SHR,DSN=DB8H8.SDSNLOAD
//SYSCIN   DD DSN=&&DSNHOUT,DISP=(MOD,PASS),UNIT=SYSDA,
//            SPACE=(800,(500,500))
//SYSLIB   DD DISP=SHR,DSN=DB8HU.SRCLIB.DATA
//SYSPRINT DD SYSOUT=*
//SYSTERM  DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSUT1   DD SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT2   DD SPACE=(800,(500,500),,,ROUND),UNIT=VIO
/*
//*************************************************************
//* CICS Translator *
//*************************************************************
```

```
//TRN EXEC PGM=DFHECP1$,
// COND=(4,LT,PC)
//STEPLIB  DD DSN=CICSTS23.CICS.SDFHLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSPUNCH DD DSN=&&SYSCIN,
//            DISP=(MOD,PASS),UNIT=SYSDA,
//            DCB=BLKSIZE=400,
//            SPACE=(400,(400,100))
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD DSN=&&DSNHOUT,DISP=(OLD,DELETE)
//*
//*************************************************************
//* Compile *
//*************************************************************
//COB      EXEC PGM=IGYCRCTL,
//         PARM=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)'),
//         COND=(4,LT,TRN)
//SYSPRINT DD SYSOUT=*
//SYSLIB   DD DSN=CICSTS23.CICS.SDFHCOB,DISP=SHR
//         DD DSN=CICSTS23.CICS.SDFHMAC,DISP=SHR
//         DD DSN=CICSTS23.CICS.SDFHSAMP,DISP=SHR
//         DD DSN=LUISM.TEST.SAMPLIB,DISP=SHR
//SYSTERM  DD SYSOUT=*
//SYSLIN   DD DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=VIO,
//         SPACE=(800,(500,500))
//SYSIN    DD DSN=&&SYSCIN,DISP=(OLD,DELETE)
//SYSUT1   DD SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT2   DD SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT3   DD SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT4   DD SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT5   DD SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT6   DD SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT7   DD SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//*************************************************************
//* Prelink *
//*************************************************************
//PLKED    EXEC PGM=EDCPRLK,COND=(4,LT,COB)
//STEPLIB  DD DISP=SHR,DSN=CEE.SCEERUN
//SYSMSGS  DD DISP=SHR,
//            DSN=CEE.SCEEMSGP(EDCPMSGE)
```

```
//SYSIN    DD DSN=&&LOADSET,DISP=(OLD,DELETE)
//SYSMOD   DD DSN=&&PLKSET,UNIT=SYSDA,DISP=(MOD,PASS),
//            SPACE=(32000,(30,30)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSDEFSD DD DUMMY
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTERM  DD SYSOUT=*
//*************************************************************
//* Linkedit *
//*************************************************************
//LKED     EXEC PGM=IEWL,PARM='LIST,XREF',
//              COND=(4,LT,PLKED)
//SYSLIB   DD DISP=SHR,DSN=CEE.SCEELKED
//         DD DISP=SHR,DSN=DB8H8.SDSNLOAD
//         DD DISP=SHR,DSN=CICSTS23.CICS.SDFHLOAD
//         DD DISP=SHR,DSN=ISP.SISPLOAD
//         DD DISP=SHR,DSN=GDDM.SADMMOD
//SYSLMOD  DD DSN=CICSTS23.CICS.SDFHLOAD(XYZ2),
//            DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD SPACE=(1024,(50,50)),UNIT=VIO
//SYSLIN   DD DSN=&&PLKSET,DISP=(OLD,DELETE)
//         DD DDNAME=SYSIN
//CICSLOAD DD DSN=CICSTS23.CICS.SDFHLOAD,
//            DISP=SHR
//SYSIN    DD *
           INCLUDE CICSLOAD(DSNCLI)
           MODE RMODE(ANY)
           NAME XYZ2(R)
/*
//*************************************************************
//* Bind *
//*************************************************************
//BIND     EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT,LKED)
//STEPLIB  DD DSN=DB8H8.SDSNLOAD,DISP=SHR
//DBRMLIB  DD DSN=DB8HU.DBRMLIB.DATA,DISP=SHR
//SYSUDUMP DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
```

```
//SYSIN DD *
  GRANT BIND, EXECUTE ON PLAN XYZP TO PUBLIC;
//SYSTSIN DD *
DSN SYSTEM(DB8H)
BIND PACKAGE (DSN8CC81) MEMBER(XYZ2) -
    ACT(REP) ISO(CS) ENCODING(EBCDIC)
BIND PLAN(XYZP) PKLIST(DSN8CC81.*) -
    ACT(REP) ISO(CS) ENCODING(EBCDIC)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA81) -
    LIB('DB8HU.RUNLIB.LOAD')
END
/*
```

Определения CICS

Все ресурсы CICS определяются в оперативном режиме через транзакцию CEDA. Группа имеет имя PAZSGROU.

Ресурс	Тип
ABCD	Транзакция
XYZ2	Программа
TMAPSET	Программа (карта)
TMAPSET	Набор карт
XYZE	Запись DB2 (сопоставляет транзакцию ABCD и имя плана XYZP)

Выполнение программы

Введите «ABCD» в экране CICS и нажмите Enter. Затем введите номер отдела и нажмите Enter. Для выхода из программы нажмите PF3.

```
ABCD txid Average salary by department
Type a department number and press Enter.
Department number: ___
Average salary($):
F3=Exit
```

Программа COBOL-Batch-VSAM

Код программы

Эта программа находится в разделе XYZ3 библиотеки LUISM.TEST.SAMPLIB

```
IDENTIFICATION DIVISION.
*-------------------------------------------------------------
* COBOL VSAM PROGRAM ZSCHOLAR RESIDENCY
```

```
*-------------------------------------------------------------
PROGRAM-ID. XYZ3.
/
ENVIRONMENT DIVISION.
*--------------------
CONFIGURATION SECTION.
SPECIAL-NAMES.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT I-FILE
    ASSIGN TO KSDATA
    ORGANIZATION IS INDEXED
    ACCESS IS DYNAMIC
    RECORD KEY IS I-FILE-RECORD-KEY
    ALTERNATE RECORD KEY IS I-FILE-ALTREC-KEY
    FILE STATUS IS FSTAT-CODE VSAM-CODE.
    SELECT DPTONO
    ASSIGN TO SYSIN
    ORGANIZATION IS SEQUENTIAL
    ACCESS IS SEQUENTIAL
    FILE STATUS IS DPTONO-CODE.
DATA DIVISION.
*-------------
FILE SECTION.
FD I-FILE
    RECORD CONTAINS 101 CHARACTERS.
01 I-FILE-RECORD.
    05 I-FILE-RECORD-KEY PIC X(6).
    05 FILLER PIC X(32).
    05 I-FILE-ALTREC-KEY PIC X(3).
    05 FILLER PIC X(42).
    05 SALARY PIC S9(7)V9(2) COMP-3.
    05 FILLER PIC X(13).
FD DPTONO
    RECORDING MODE F
    BLOCK 0 RECORDS
    RECORD 80 CHARACTERS
    LABEL RECORD STANDARD.
01 DPTONO-RECORD PIC X(80).
/
```

```
WORKING-STORAGE SECTION.
01 STATUS-AREA.
   05 FSTAT-CODE PIC X(2).
     88 I-O-OKAY VALUE ZEROES.
   05 VSAM-CODE.
     10 VSAM-R15-RETURN-CODE PIC 9(2) COMP.
     10 VSAM-FUNCTION-CODE PIC 9(1) COMP.
     10 VSAM-FEEDBACK-CODE PIC 9(3) COMP.
77 DPTONO-CODE PIC XX.
01 WS-DPTONO-RECORD.
   05 DPTONO-KEYED PIC X(3).
   05 FILLER PIC X(77).
01 WS-SALARY PIC S9(7)V9(2) COMP-3 VALUE 0.
01 WS-SALARY-EDITED PIC $ZZ,ZZZ,ZZ9.99.
/
PROCEDURE DIVISION.
    OPEN INPUT DPTONO.
    READ DPTONO INTO WS-DPTONO-RECORD.
    DISPLAY DPTONO-KEYED.
    OPEN INPUT I-FILE.
    IF FSTAT-CODE NOT = «00»
       DISPLAY «OPEN INPUT VSAMFILE FS-CODE: « FSTAT-CODE
       PERFORM VSAM-CODE-DISPLAY
       STOP RUN
    END-IF.
    MOVE DPTONO-KEYED TO I-FILE-ALTREC-KEY.
    PERFORM READ-FIRST.
    IF FSTAT-CODE = «02»
       PERFORM READ-NEXT UNTIL FSTAT-CODE = «00»
    END-IF.
    IF FSTAT-CODE = «23»
       DISPLAY «NO RECORDS EXISTS FOR THIS DEPARTMENT»
    END-IF.
    MOVE WS-SALARY TO WS-SALARY-EDITED.
    DISPLAY WS-SALARY-EDITED.
    CLOSE DPTONO.
    CLOSE I-FILE.
    STOP RUN.
READ-NEXT.
    READ I-FILE NEXT.
```

```
    IF FSTAT-CODE NOT = «00» AND FSTAT-CODE NOT = «02»
      DISPLAY «READ NEXT I-FILE FS-CODE: « FSTAT-CODE
      PERFORM VSAM-CODE-DISPLAY
    ELSE
      ADD SALARY TO WS-SALARY
    END-IF.
READ-FIRST.
    READ I-FILE RECORD KEY IS I-FILE-ALTREC-KEY.
    IF FSTAT-CODE NOT = «00» AND FSTAT-CODE NOT = «02»
      DISPLAY «READ I-FILE FS-CODE: « FSTAT-CODE
      PERFORM VSAM-CODE-DISPLAY
    ELSE
      ADD SALARY TO WS-SALARY
    END-IF.
VSAM-CODE-DISPLAY.
    DISPLAY «VSAM CODE -->»
          « RETURN: « VSAM-R15-RETURN-CODE,
          « COMPONENT: « VSAM-FUNCTION-CODE,
          « REASON: « VSAM-FEEDBACK-CODE.
```

Подготовительные задания

Создание среды VSAM

Это задание находится в разделе VSAMDEF библиотеки LUISM.TEST.SAMPLIB.

Задание выполняет следующие действия:

- Выгружает DB2-таблицу сотрудников в последовательный файл.
- Удаляет/определяет файл VSAM KSDS.
- Определяет альтернативный индекс (по номеру отдела).
- Определяет путь.
- Выполняет копирование (repro) файла VSAM из последовательного файла (шаг 1).
- Выполняет BLDINDEX.

```
//LUISM06  JOB (999,POK),'UNLTAB/DEFVSAM/REPRO',
//         CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1)
//*************************************************************
//*        UNLOAD DB2 TABLE *
//*************************************************************
//STEP00   EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD DSN=DB8H8.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
```

```
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSREC00 DD  DSN=LUISM.EMP.TABLE.UNLOAD,
//             SPACE=(TRK,(1,1)),DISP=(,CATLG)
//SYSPUNCH DD  DSN=LUISM.EMP.TABLE.SYSPUNCH,
//             SPACE=(TRK,(1,1)),DISP=(,CATLG),
//             RECFM=FB,LRECL=120
//SYSIN    DD *
DSN8810.EMP
/*
//SYSTSIN DD *
DSN SYSTEM(DB8H)
RUN PROGRAM(DSNTIAUL) PLAN(DSNTIB81) -
    LIB('DB8HU.RUNLIB.LOAD')
END
/*
//*************************************************************
//*        DELETE THE KSDS FILE *
//*************************************************************
//STEP01   EXEC PGM=IDCAMS,COND=(4,LT,STEP00)
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
DELETE LUISM.KSDATA
/*
//*************************************************************
//*        DEFINE A KSDS FILE *
//*************************************************************
//STEP02   EXEC PGM=IEFBR14,COND=(4,LT,STEP01)
//DEFINE   DD DSN=LUISM.KSDATA,DISP=(NEW,KEEP),
//            SPACE=(TRK,(1,1)),AVGREC=U,RECORG=KS,
//            KEYLEN=6,KEYOFF=0,LRECL=101
/*
//*************************************************************
//*        DEFINE ALTERNATE INDEX *
//*************************************************************
//STEP03   EXEC PGM=IDCAMS,COND=(4,LT,STEP02)
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
DEFINE ALTERNATEINDEX -
    (NAME(LUISM.ALTINDEX) -
```

```
    RELATE(LUISM.KSDATA) -
    NONUNIQUEKEY -
    KEYS(3 38) -
    RECORDSIZE(23 150) -
    VOLUMES(TOTSSI) -
    KILOBYTES(100 100) -
    UPGRADE)
/*
//*************************************************************
//*        DEFINE PATH *
//*************************************************************
//STEP04   EXEC PGM=IDCAMS,COND=(4,LT,STEP03)
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
DEFINE PATH -
    (NAME(LUISM.PATH) -
    PATHENTRY(LUISM.ALTINDEX))
/*
//*************************************************************
//* REPRO INTO THE KSDS FROM DB2 UNLOAD SEQ FILE *
//*************************************************************
//STEP05   EXEC PGM=IDCAMS,COND=(4,LT,STEP04)
//SEQFILE  DD DSN=LUISM.EMP.TABLE.UNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
REPRO  INFILE(SEQFILE) -
       OUTDATASET(LUISM.KSDATA) -
       REPLACE
/*
//*************************************************************
//* BLDINDEX *
//*************************************************************
//STEP06   EXEC PGM=IDCAMS,COND=(4,LT,STEP05)
//BASEDD   DD DSN=LUISM.KSDATA,DISP=SHR
//AIXDD    DD DSN=LUISM.ALTINDEX,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
BLDINDEX  INFILE(BASEDD) -
          OUTFILE(AIXDD) -
          SORTCALL
/*
```

Подготовка программы

Это задание находится в разделе BATVSAMP библиотеки LUISM.TEST.SAMPLIB.

```
//LUISM07  JOB (999,POK),'Cobol-VSAM',
//         CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1)
//*************************************************************
//* Compile the IBM COBOL program *
//*************************************************************
//COB      EXEC PGM=IGYCRCTL,
//         PARM=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)')
//SYSPRINT DD SYSOUT=*
//SYSTERM  DD SYSOUT=*
//SYSLIN   DD DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=VIO,
//            SPACE=(800,(500,500))
//SYSIN    DD DSN=LUISM.TEST.SAMPLIB(XYZ3),DISP=SHR
//SYSUT1   DD SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT2   DD SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT3   DD SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT4   DD SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT5   DD SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT6   DD SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT7   DD SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//*************************************************************
//* PRELINK STEP. *
//*************************************************************
//PLKED    EXEC PGM=EDCPRLK,COND=(4,LT,COB)
//STEPLIB  DD DISP=SHR,DSN=CEE.SCEERUN
//SYSMSGS  DD DISP=SHR,
//            DSN=CEE.SCEEMSGP(EDCPMSGE)
//SYSIN    DD DSN=&&LOADSET,DISP=(OLD,DELETE)
//SYSMOD   DD DSN=&&PLKSET,UNIT=SYSDA,DISP=(MOD,PASS),
//            SPACE=(32000,(30,30)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSDEFSD DD DUMMY
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTERM  DD SYSOUT=*
//*************************************************************
//* Linkedit *
//*************************************************************
//LKED     EXEC PGM=IEWL,PARM='LIST,XREF',
//         COND=(4,LT,PLKED)
//SYSLIB   DD DISP=SHR,DSN=CEE.SCEELKED
```

```
//         DD DISP=SHR,DSN=ISP.SISPLOAD
//         DD DISP=SHR,DSN=GDDM.SADMMOD
//SYSLMOD  DD DSN=LUISM.TEST.LOADLIB(XYZ3),
//            DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD SPACE=(1024,(50,50)),UNIT=VIO
//SYSLIN   DD DSN=&&PLKSET,DISP=(OLD,DELETE)
//         DD DDNAME=SYSIN
//SYSIN    DD *
           MODE RMODE(ANY)
           NAME XYZ3(R)
/*
```

Задание на выполнение программы

Выполнение программы находится в разделе RUNXYZ3 библиотеки LUISM.TEST.SAMPLIB.

```
//LUISM08  JOB (999,POK),'EJEC. COB-VSAM',
// CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1)
//*************************************************************
//STEP01   EXEC PGM=XYZ3
//STEPLIB  DD DSN=LUISM.TEST.LOADLIB,DISP=SHR
//KSDATA   DD DSN=LUISM.KSDATA,DISP=SHR
//KSDATA1  DD DSN=LUISM.PATH,DISP=SHR
//OUTPUTFI DD SYSOUT=*
//SYSIN    DD *
E01
/*
```

Ниже представлены выходные данные для отдела E01:

```
************************ TOP OF DATA ************************
E01
$ 40,175.00
********************** BOTTOM OF DATA ************************
```

Выходные данные для отдела, не имеющего сотрудников, имеют следующий вид:

```
************************ TOP OF DATA ************************
A01
READ I-FILE FS-CODE: 23
VSAM CODE --> RETURN: 08 COMPONENT: 2 REASON: 016
NO RECORDS EXISTS FOR THIS DEPARTMENT
$ 0.00
********************** BOTTOM OF DATA ************************
```

Утилита DSNTEP2

Эта PL/I-программа динамически выполняет SQL-операторы, считываемые из SYSIN. Это приложение может также выполнять операторы, отличные от SELECT.

Задание на выполнение

Это задание на выполнение находится в разделе DSNTEP2 библиотеки LUISM.TEST.SAMPLIB.

```
//LUISM04  JOB (999,POK),'Dsntep2',
//         CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1)
//*************************************************************
//* DSNTEP2 *
//*************************************************************
//DSNTEP2  EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD DSN=DB8H8.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
DSN SYSTEM(DB8H)
RUN PROGRAM(DSNTEP2) PLAN(DSNTEP81) -
    LIB('DB8HU.RUNLIB.LOAD')
END
/*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD *
   SELECT  CHAR(DECIMAL(SUM(SALARY),9,2))
   FROM    DSN8810.EMP
   WHERE   WORKDEPT='A00'
/*
Following is the output:
PAGE 1
***INPUT STATEMENT:
   SELECT  CHAR(DECIMAL(SUM(SALARY),9,2))
   FROM    DSN8810.EMP
   WHERE   WORKDEPT='A00'
+-------------+
| |
+-------------+
1_| 0204250.00 |
+-------------+
SUCCESSFUL RETRIEVAL OF 1 ROW(S)
```

Пакетное выполнение QMF

Это упражнение показывает пакетное выполнение QMF-процедуры/запроса/формы. Запрос EMPQRY содержит SQL-оператор нашей программы для практической работы. Процедура EMPPRO инициирует выполнение запроса и распечатку отчета. Задание вызывает QMF-процедуру и передает ей номер отдела; кроме того, задается режим выполнения (пакетный, M=B) и подсистема DB2.

Задание находится в разделе QMFBATCH библиотеки LUISM.TEST.SAMPLIB.

QMF вызывается из ISPF-опции Q7 в системе SC47TS с указанием ISPQMF71 в поле COMMAND.

Задание на выполнение

```
//LUISM10  JOB (999,POK),'QMF in batch',
//         CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1)
/*JOBPARM SYSAFF=SC47
//*************************************************************
//QMFBAT   EXEC PGM=DSQQMFE,
// PARM='M=B,I=LUISM.EMPPRO(&&DEP=''A00''),S=DB7D'
//STEPLIB  DD DISP=SHR,DSN=DB7DU.SDSQLOAD
//         DD DISP=SHR,DSN=DB7D7.SDSNLOAD
//         DD DISP=SHR,DSN=DB7D7.SDSNEXIT
//ADMGGMAP DD DSN=DB7DU.DSQMAPE,DISP=SHR
//DSQPRINT DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=1330)
//DSQDEBUG DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=121,BLKSIZE=1210)
//DSQUDUMP DD SYSOUT=A,DCB=(RECFM=VBA,LRECL=125,BLKSIZE=1632)
//DSQSPILL DD DSN=&&SPILL,DISP=(NEW,DELETE),
//     UNIT=VIO,SPACE=(TRK,(100),RLSE),
//     DCB=(RECFM=F,LRECL=4096,BLKSIZE=4096)
//*
```

QMF-процедура

```
RUN QUERY EMPQRY (&&D=&DEP FORM=EMPFORM
PRINT REPORT
```

QMF-запрос

```
SELECT CHAR(DECIMAL(SUM(SALARY),9,2))
FROM DSN8710.EMP
WHERE WORKDEPT=&D
```

Пакетная программа доступа к DB2 на языке C

Исходный код

Эта программа находится в разделе CDB2 библиотеки GMULLER.TEST.C.

Пример E.1. Исходный код программы доступа к DB2 на языке C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
EXEC SQL INCLUDE SQLCA;
EXEC SQL INCLUDE SQLDA;
EXEC SQL
    DECLARE    DSN8810.EMP TABLE
    (EMPNO     CHAR(6) NOT NULL,
    FIRSTNAME  VARCHAR(12) NOT NULL,
    MIDINIT    CHAR(1) NOT NULL,
    LASTNAME   VARCHAR(15) NOT NULL,
    WORKDEPT   CHAR(3) ,
    PHONENO    CHAR(4) ,
    HIREDATE   DATE ,
    JOB        CHAR(8) ,
    EDLEVEL    SMALLINT ,
    SEX        CHAR(1) ,
    BIRTHDATE  DATE ,
    SALARY DECIMAL(9,2) ,
    BONUS DECIMAL(9,2) ,
    COMM DECIMAL(9,2) );
EXEC SQL BEGIN DECLARE SECTION;
    long sum;
    long count;
    char deptno[4];
EXEC SQL END DECLARE SECTION;
int avg_sal(char*);
int record_read(FILE*,char*);
void main()
{
    FILE* cardin; /* for DD card CARDIN */
    int avgsal;
    char dept[4];
    cardin = fopen(«DD:CARDIN»,«rb,recfm=FB,lrecl=80,type=record»);
    if(cardin == NULL) {
          printf(«Error opening DD CARDIN\n»);
          exit(-2);
    }
```

```
    while(record_read(cardin, dept) != 0)  {
          avgsal = avg_sal(dept);
          if(avgsal > 0)
          printf(«Average salary of %s is %d\n»,dept, avgsal);
    }
    fclose(cardin);
}
int avg_sal(char* dept)
{
    int avgsal;
    count = 0;
    strncpy(deptno, dept, 3);
    deptno[3] = 0;
    EXEC SQL SELECT SUM(SALARY), COUNT(*) INTO :sum, :count
          FROM DSN8810.EMP
          WHERE WORKDEPT = :deptno;
    if(count != 0) {
          avgsal = sum/count;
          return avgsal;
    } else {
          printf(«DEPT %s does not exist\n», deptno);
          return -1;
    }
}
int record_read(FILE* file, char* dept)
{
    int readbytes;
    char linebuf[81], linebuf2[80];
    readbytes = fread(linebuf, 1, 81, file);
    strncpy(dept, linebuf, 3); /* first 3 bytes are dept. number */
    dept[3]=0; /* terminate string */
    return readbytes;
}
```

Подготовка программы

Этот JCL находится в разделе CDB2 библиотеки GMULLER.TEST.CNTL.

Пример E.2. GMULLER.TEST.CNTL(CDB2)

```
//GMULLERC JOB 1,GEORG,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//* PRECOMPILE AND COMPILE THE SAMPLE C FILE
//PROCLIB JCLLIB ORDER=DB8HU.PROCLIB
/*JOBPARM SYSAFF=SC04
//STEP1 EXEC PROC=DSNHC,MEM=CDB2,
//  PARM.PC=('HOST(C),CCSID(1047)')
//PC.DBRMLIB  DD DSN=DB8HU.DBRMLIB.DATA(CDB2),DISP=SHR
```

```
//PC.SYSLIB   DD DSN=GMULLER.TEST.C,DISP=SHR
//PC.SYSIN    DD DSN=GMULLER.TEST.C(&MEM),DISP=SHR
//LKED.SYSLMOD DD DSN=GMULLER.TEST.LOAD(&MEM),DISP=SHR
//LKED.SYSIN  DD *
INCLUDE SYSLIB(DSNELI)
//*************************************************
//* BIND AND RUN THE PROGRAM *
//*************************************************
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//STEPLIB  DD DSN=DB8H8.SDSNLOAD,DISP=SHR
//DBRMLIB  DD DSN=DB8HU.DBRMLIB.DATA,DISP=SHR
//SYSUDUMP DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//CARDIN   DD *
D11
XYZ
A00
/*
//SYSIN DD *
GRANT BIND,EXECUTE ON PLAN CDB2 TO PUBLIC;
//SYSTSIN DD *
DSN SYSTEM(DB8H)
BIND PACKAGE (CDB2PAK) MEMBER(CDB2) -
    ACT(REP) ISO(CS) ENCODING(EBCDIC)
BIND PLAN(CDB2) PKLIST(CDB2PAK.*) -
    ACT(REP) ISO(CS) ENCODING(EBCDIC)
RUN PROGRAM(CDB2) PLAN(CDB2) LIB('GMULLER.TEST.LOAD')
END
/*
```

- Это задание требует PDS GMULLER.TEST.LOAD с RECFM=U.
- Оператор «/*JOBPARM SYSAFF=SC04» указывает на систему, в которой запущен DB2; этот оператор нужно изменить (или удалить, если не используется сисплекс).
- DB8H нужно заменить именем локальной DB2.
- HLQ для библиотек DB2 могут отличаться.

Выходные данные

Пример E.3. Выходные данные CDB2

```
Average salary of D11 is 25147
DEPT XYZ does not exist
Average salary of A00 is 40850
```

Выполнение программы

Этот JCL находится в разделе RUNJCL библиотеки GMULLER.TEST.CNTL.

Пример E.4. GMULLER.TEST.CNTL(RUNJCL)

```
//GMULLERR JOB 1,GEORG,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//* PRECOMPILE AND COMPILE THE SAMPLE C FILE
/*JOBPARM SYSAFF=SC04
//*************************************************
//* RUN THE PROGRAM *
//*************************************************
//BIND     EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//STEPLIB  DD DSN=DB8H8.SDSNLOAD,DISP=SHR
//DBRMLIB  DD DSN=DB8HU.DBRMLIB.DATA,DISP=SHR
//SYSUDUMP DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//CARDIN   DD DISP=SHR,DSN=GMULLER.TEST.CNTL(CARDIN)
//SYSTSIN  DD *
DSN SYSTEM(DB8H)
RUN PROGRAM(CDB2) PLAN(CDB2) LIB('GMULLER.TEST.LOAD')
END
/*
```

- Этот пример требует раздел CARDIN в библиотеке GMULLER.TEST.CNTL
- HLQ для библиотек DB2 могут отличаться.

Входные данные

Пример E.5. GMULLER.TEST.CNTL(CARDIN)

```
D11
A00
XYZ
C01
ABC
E21
```

Выходные данные

Пример E.6. Выходные данные RUNJCL

```
Average salary of D11 is 25147
Average salary of A00 is 40850
DEPT XYZ does not exist
Average salary of C01 is 29722
DEPT ABC does not exist
Average salary of E21 is 24086
```

Доступ Java-сервлета к DB2

Исходный код сервлета

Пример E.7. SalaryServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;
public class SalaryServlet extends HttpServlet {
    private DataSource ds;
    private boolean dbProblem = false;
    public void init() throws ServletException {
          super.init();
          try { // get DataSource from Container
          Context context = new InitialContext();
          ds = (DataSource) context.lookup("jdbc/DB8H");
    } catch (NamingException e) {
          e.printStackTrace();
          this.dbProblem = true;
    }
}
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
          throws ServletException, IOException {
    resp.setContentType("text/html");
    String deptno = req.getParameter("deptno");// get from request string
    PrintWriter out = resp.getWriter();
    out.println("<html>\n<head>\n <title>Average
Salary</title>\n</head>\n<body>");
    out.println("<h1>Average Salary</h1>");
    out.println("<form action=\"salary\" method=\"get\">");
    out.println("Dept. No.: <input type=\"text\" name=\"deptno\" />");
```

```
    out.println(" <input type=\"submit\" />\n</form>");
    if (deptno != null) {
          try {
                int avgSal = getAvgSal(deptno);
                out.println("The average salary of <b>" + deptno
                      + "</b> is <b>$ " + avgSal
                      + "</b><br>");
          } catch (Exception e) {
              out.println("<b>Error: " + e.getMessage() +
                          "</b><br>");
          }
}
out.println("</html>");
}
private int getAvgSal(String deptno) throws Exception {
  String sqlStatement = "SELECT SUM(salary), COUNT(*) "
    + "FROM DSN8810.EMP WHERE WORKDEPT = '" + deptno + "'";
  // Connect to database
  Connection con = null;
  try {
    con = ds.getConnection();
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(sqlStatement);// Execute SQL
    // statement
    rs.next(); // get Values from result set
    int sum = rs.getInt(1);
    int count = rs.getInt(2);
    if (count == 0)
      throw new Exception("Department " + deptno
        + " does not exist");
      return sum / count;
    } catch (SQLException e) {
      throw new Exception(e.getMessage());
    } finally {
      try {
        con.close();
      } catch (SQLException e) {}
    }
  }
}
```

- Этот сервлет требует определения источника данных (в данном случае JNDI с именем «jdbc/DB8H»), определенного в веб-контейнере, указывающего на базу данных DB2

Дескриптор развертывания

Пример E.8. web.xml

```
<?xml version=»1.0» encoding=»UTF-8»?>
<!DOCTYPE web-app PUBLIC «-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN» «http://java.sun.com/dtd/web-app_2_3.dtd»>
<web-app id=»WebApp_ID»>
    <display-name>Salary</display-name>
    <servlet>
          <servlet-name>Salary</servlet-name>
          <servlet-class>SalaryServlet</servlet-class>
    </servlet>
    <servlet-mapping>
          <servlet-name>Salary</servlet-name>
          <url-pattern>/salary</url-pattern>
    </servlet-mapping>
</web-app>
```

Программа доступа к MQ на языке C

MQPUT записывает сообщение в очередь (вводится в TSO).

Программа запускается командой TSO CALL 'ZSCHOLAR.PROGRAM.LOAD(MQPUT)', после чего необходимо ввести сообщение.

MQGET получает сообщение обратно и выводит его на экране.

Программа запускается командой TSO CALL 'ZSCHOLAR.PROGRAM.LOAD(MQGET)', после чего необходимо ввести сообщение.

Также можно принять сообщение, используя Java-приложение из раздела «Программа доступа к MQ на языке Java».

MQPUT

Пример E.9. ZSCHOLAR.PROGRAM.SRC(MQPUT)

```
#pragma csect(code,»CSQ4BCK1»)
/* */
/* Define static CSECT name */
/* */
#pragma csect(static,»BCK1WS»)
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <cmqc.h>
/* */
```

```
/* Function prototypes */
/* */
void usageError( char* programName );
void errorMessage( char* msgStr, MQLONG CC, MQLONG RC );
int main( int argc, char** argv )
{
    /* */
    /* API variables */
    /* */
    MQHCONN HConn = MQHC_DEF_HCONN;
    MQHOBJ HObj;
    MQLONG OpenOptions;
    MQMD MsgDesc = { MQMD_DEFAULT };
    MQOD ObjDesc = { MQOD_DEFAULT };
    MQPMO PutMsgOpts = { MQPMO_DEFAULT };
    MQLONG CompCode;
    MQLONG Reason;
    /* */
    /* Parameter variables */
    /* */
    MQCHAR48 qMgr;
    MQCHAR48 qName;
    char msgBuffer[255];
    int msgLength;
    char persistent = 'N';
    long rc = 0;
    printf(«Please enter message text:\n»);
    fgets(msgBuffer, 255, stdin);
    msgLength = strlen(msgBuffer);
    strcpy( qMgr, «MQ8H\0» );
    strcpy( qName, «GMULLER\0» );
    /*
    memset( qMgr, '\0', MQ_Q_MGR_NAME_LENGTH );
    memset( qName, '\0', MQ_Q_NAME_LENGTH );
    */
    /* */
    /* Connect to Queue Manager (MQCONN) */
    /* */
    MQCONN( qMgr,
            &HConn,
            &CompCode,
            &Reason );
    /* */
    /* If connect failed then display error message and exit */
```

```
/* */
if( MQCC_OK != CompCode ) {
       errorMessage( «MQCONN», CompCode, Reason );
       return Reason;
   }
printf( «MQCONN SUCCESSFUL\n» );
/* */
/* Open Queue for output (MQOPEN). Fail the call if the queue */
/* manager is quiescing. */
/* */
OpenOptions = MQOO_OUTPUT +
              MQOO_FAIL_IF_QUIESCING;
strncpy( ObjDesc.ObjectName, qName, MQ_Q_NAME_LENGTH );
MQOPEN( HConn,
        &ObjDesc,
        OpenOptions,
        &HObj,
        &CompCode,
        &Reason );
/* */
/* If open failed then display error message, */
/* disconnect from the queue manager and exit */
/* */
if( MQCC_OK != CompCode ) {
       errorMessage( «MQOPEN», CompCode, Reason );
       rc = Reason;
       MQDISC( &HConn,
               &CompCode,
               &Reason );
       return rc;
   }
printf( «MQOPEN SUCCESSFUL\n» );
/* */
/* Set persistence depending on parameter passed */
/* */
if( 'P' == persistent )
   MsgDesc.Persistence = MQPER_PERSISTENT;
else
   MsgDesc.Persistence = MQPER_NOT_PERSISTENT;
/* */
/* Put String format messages */
strncpy( MsgDesc.Format, MQFMT_STRING, MQ_FORMAT_LENGTH );
/* */
/* Set the put message options to fail the call if the queue */
```

```
/* manager is quiescing. */
/* */
PutMsgOpts.Options = MQPMO_FAIL_IF_QUIESCING;
strncpy( MsgDesc.MsgId, MQMI_NONE, MQ_MSG_ID_LENGTH );
strncpy( MsgDesc.CorrelId, MQCI_NONE, MQ_CORREL_ID_LENGTH );
MQPUT( HConn,
        HObj,
        &MsgDesc,
        &PutMsgOpts,
        msgLength,
        msgBuffer,
        &CompCode,
        &Reason );
/*
/* If put failed then display error message */
/* and break out of loop */
/* */
if( MQCC_OK != CompCode )
  {
      errorMessage( «MQPUT», CompCode, Reason );
      rc = Reason;
  }
printf(«MESSAGE PUT TO QUEUE\n»);
free( msgBuffer );
/* */
/* Close the queue and then disconnect from the queue manager */
/* */
MQCLOSE( HConn,
         &HObj,
         MQCO_NONE,
         &CompCode,
         &Reason );
if( MQCC_OK != CompCode )
  {
      errorMessage( «MQCLOSE», CompCode, Reason );
      rc = Reason;
  }
else printf( «MQCLOSE SUCCESSFUL\n» );
MQDISC( &HConn,
        &CompCode,
        &Reason );
if( MQCC_OK != CompCode )
  {
      errorMessage( «MQDISC», CompCode, Reason );
```

```
            return Reason;
          }
        else
          {
            printf( «MQDISC SUCCESSFUL\n» );
            return rc;
          }
        return(rc);
    } /*end main*/
    /*************************************************************/
    /* Functions to display error messages */
    /*************************************************************/
    void errorMessage( char* msgStr, MQLONG CC, MQLONG RC )
    {
      printf( «*********************************************\n» );
      printf( «* %s\n», msgStr );
      printf( «* COMPLETION CODE : %09ld\n», CC );
      printf( «* REASON CODE : %09ld\n», RC );
      printf( "*********************************************\n" );
    }
```

JCL-код, выполняющий компиляцию:

Пример E.10. ZSCHOLAR.PROGRAM.CNTL(MQPUT)

```
    //GMULLERT JOB 1,GEORG,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID
    //* COMPILE MQ PROGRAM
    //STEP1    EXEC PROC=EDCCB,
    //            INFILE='ZSCHOLAR.PROGRAM.SRC(MQPUT)',
    //            OUTFILE='ZSCHOLAR.PROGRAM.LOAD(MQPUT),DISP=SHR'
    //SYSLIB   DD DSN=MQ531.SCSQC370,DISP=SHR
    //BIND.CSQBSTUB DD DSN=MQ531.SCSQLOAD(CSQBSTUB),DISP=SHR
    //BIND.SYSIN DD *
                INCLUDE CSQBSTUB
    /*
```

MQGET

Исходный код.

Пример E.11. ZSCHOLAR.PROGRAM.SRC(MQGET)

```
    #pragma csect(code,"CSQ4BCK1")
    /* */
    /* Define static CSECT name */
    /* */
```

```
#pragma csect(static,"BCK1WS")
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <cmqc.h>
#define maxMessageLength 65536
/* */
/* Function prototypes */
/* */
void usageError( char* programName );
void errorMessage( char* msgStr, MQLONG CC, MQLONG RC );
int main( int argc, char** argv )
    {
    /* */
    /* API variables */
    /* */
    MQHCONN      HConn = MQHC_DEF_HCONN;
    MQHOBJ       HObj;
    MQLONG       OpenOptions;
    MQMD         MsgDesc = { MQMD_DEFAULT };
    MQOD         ObjDesc = { MQOD_DEFAULT };
    MQGMO        GetMsgOpts = { MQGMO_DEFAULT };
    MQLONG       CompCode;
    MQLONG       Reason;
    /* */
    /* Parameter variables */
    /* */
    MQCHAR48     qMgr;
    MQCHAR48     qName;
    char   msgBuffer[maxMessageLength];
    int    msgLength = maxMessageLength;
    char   persistent = 'N';
    long rc = 0;
    long dataLength;
    char browseGet = 'D'; /* destructive get */
    char syncpoint = 'N'; /* no Syncpoint */
    memset( msgBuffer, '\0', msgLength );
    strcpy( qMgr, "MQ8H\0" );
    strcpy( qName, "GMULLER\0" );
    /* */
```

```
/* Connect to Queue Manager (MQCONN) */
/* */
MQCONN( qMgr,
        &HConn,
        &CompCode,
        &Reason );
/* */
/* If connect failed then display error message and exit */
/* */
if( MQCC_OK != CompCode ) {
       errorMessage( "MQCONN", CompCode, Reason );
       return Reason;
   }
printf( "MQCONN SUCCESSFUL\n" );
/* */
/* Open Queue for input shared and browse. Fail the call if the */
/* queue manager is quiescing. */
/* */
OpenOptions = MQOO_INPUT_SHARED +
              MQOO_BROWSE +
              MQOO_FAIL_IF_QUIESCING;
strncpy( ObjDesc.ObjectName, qName, MQ_Q_NAME_LENGTH );
MQOPEN( HConn,
        &ObjDesc,
        OpenOptions,
        &HObj,
        &CompCode,
        &Reason );
/* */
/* If open failed then display error message, */
/* disconnect from the queue manager and exit */
/* */
if( MQCC_OK != CompCode ) {
       errorMessage( "MQOPEN", CompCode, Reason );
       rc = Reason;
       MQDISC( &HConn,
               &CompCode,
               &Reason );
       return rc;
   }
printf( "MQOPEN SUCCESSFUL\n" );
```

```
    /* */
    /* Set persistence depending on parameter passed */
    /* */
    if( 'P' == persistent )
          MsgDesc.Persistence = MQPER_PERSISTENT;
    else
          MsgDesc.Persistence = MQPER_NOT_PERSISTENT;
    /* */
/* Set GetMsgOpts .. don't wait if there are no messages on the */
/* queue, truncate the message if it does not fit into our */
/* buffer, perform data conversion on the message if required */
/* and if possible, and fail the call if the queue manager is */
/* quiescing. */
/* */
    GetMsgOpts.Options = MQGMO_NO_WAIT +
                         MQGMO_ACCEPT_TRUNCATED_MSG +
                         MQGMO_CONVERT +
                         MQGMO_FAIL_IF_QUIESCING;
    strncpy( MsgDesc.MsgId, MQMI_NONE, MQ_MSG_ID_LENGTH );
    strncpy( MsgDesc.CorrelId, MQCI_NONE, MQ_CORREL_ID_LENGTH );
    /* */
/* Set additional GetMsgOpts depending on parameters passed */
/* into program. */
/* */
    if( ('S' == syncpoint) && ('B' != browseGet) )
          GetMsgOpts.Options += MQGMO_SYNCPOINT;
    else
          GetMsgOpts.Options += MQGMO_NO_SYNCPOINT;
    if( ('B' == browseGet) )
          GetMsgOpts.Options += MQGMO_BROWSE_FIRST;
    MsgDesc.Encoding = MQENC_NATIVE;
    MsgDesc.CodedCharSetId = MQCCSI_Q_MGR;
    /* GET */
    MQGET( HConn,
           HObj,
           &MsgDesc,
           &GetMsgOpts,
           msgLength,
           msgBuffer,
           &dataLength,
           &CompCode,
```

```
            &Reason );
    if( (MQCC_FAILED == CompCode) ) {
          errorMessage( "MQGET", CompCode, Reason );
          rc = Reason;
      }
    else {
/* */
/* Only character data messages are correctly displayed */
/* by this code */
/* */
          if (MQRC_TRUNCATED_MSG_ACCEPTED == Reason) {
                msgBuffer??( msgLength - 1 ??) = 0;
                printf( "Message received (truncated):\n%s\n",
                msgBuffer );
            }
          else {
                msgBuffer??( dataLength ??) = 0;
                printf( "Message received:\n%s\n",
                msgBuffer );
            }
      }
    free( msgBuffer );
/* */
/* Close the queue and then disconnect from the queue manager */
/* */
    MQCLOSE( HConn,
             &HObj,
             MQCO_NONE,
             &CompCode,
             &Reason );
    if( MQCC_OK != CompCode ) {
          errorMessage( "MQCLOSE", CompCode, Reason );
          rc = Reason;
      }
    else printf( "MQCLOSE SUCCESSFUL\n" );
    MQDISC( &HConn,
            &CompCode,
            &Reason );
    if( MQCC_OK != CompCode ) {
          errorMessage( "MQDISC", CompCode, Reason );
```

```
            return Reason;
        }
      else {
            printf( "MQDISC SUCCESSFUL\n" );
            return rc;
        }
      return(rc);
} /*end main*/
/*************************************************************/
/* Functions to display error messages */
/*************************************************************/
void errorMessage( char* msgStr, MQLONG CC, MQLONG RC )
{
   printf( "*******************************************\n" );
   printf( "* %s\n", msgStr );
   printf( "* COMPLETION CODE : %09ld\n", CC );
   printf( "* REASON CODE : %09ld\n", RC );
   printf( "*******************************************\n" );
}
```

JCL-код, выполняющий компиляцию:

Пример E.12. ZSCHOLAR.PROGRAM.CNTL(MQGET)

```
//GMULLERT JOB 1,GEORG,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//* COMPILE MQ PROGRAM
//STEP1    EXEC PROC=EDCCB,
//           INFILE='ZSCHOLAR.PROGRAM.SRC(MQGET)',
//           OUTFILE='ZSCHOLAR.PROGRAM.LOAD(MQGET),DISP=SHR'
//SYSLIB   DD DSN=MQ531.SCSQC370,DISP=SHR
//BIND.CSQBSTUB DD DSN=MQ531.SCSQLOAD(CSQBSTUB),DISP=SHR
//BIND.SYSIN DD *
 INCLUDE CSQBSTUB
/*
```

Программа доступа к MQ на языке Java

Java-программа получает сообщение из очереди. Класс MessageHandler также содержит класс для отправки сообщений.

Необходимо добавить com.ibm.mq.jar и connector.jar в CLASSPATH.

Все файлы находятся в «program sample\mq».

Программа запускается командой java -jar mqconnect.jar.

Пример E.13. MQReceiver.java

```
import com.ibm.mq.MQException;
public class MQReceiver {
  public static void main(String[] args) {
  // Connection settings
  String hostname = "wtsc04.itso.ibm.com";
  String queueName = "GMULLER";
  int port = 1598; // mq port
  String channel = "GMULLER.SERV";
  MessageHandler handler = new MessageHandler(hostname, port,
                               queueName,
                               channel);
  String message;
  try {
    System.out.println("Sending message...");
    handler.sendMessage("Hello");
    //System.out.println("Receiving message...");
    //message = handler.receiveMessage();
    //System.out.println("Message: " + message);
    System.out.println("Finished");
  } catch (MQException e) {
    if (e.reasonCode == MQException.MQRC_NO_MSG_AVAILABLE)
      System.out.println("No message in queue");
    else {
      System.out.println("Error getting message");
      e.printStackTrace();
      }
     }
  }
}
```

Пример E.14. MessageHandler.java

```
import java.io.IOException;
import com.ibm.mq.*;
public class MessageHandler {
  private String hostname;
  private String queueName;
  public MessageHandler(String hostname, int port, String
                        queueName, String channel) {
    MQEnvironment.hostname = hostname;
    MQEnvironment.port = port;
```

```
  MQEnvironment.channel = channel;
  this.queueName = queueName;
}
public String receiveMessage() throws MQException {
  try {
    MQQueueManager mqm = new MQQueueManager(hostname);
    int openOptions = MQC.MQOO_INPUT_AS_Q_DEF
                    + MQC.MQOO_OUTPUT;
    MQQueue queue = mqm.accessQueue(queueName, openOptions);
    // create new Message for receiving
    MQMessage message = new MQMessage();
    // get message from queue
    queue.get(message);
    // get the whole message string
    String messageString =
              message.readString(message.getMessageLength());
    // close queue;
    queue.close();
    // disconnect from queue manager
    mqm.disconnect();
    return messageString;
  } catch (IOException e) {
        e.printStackTrace();
        return null;
  }
}
public void sendMessage(String messageString) throws MQException {
  try {
    MQQueueManager mqm = new MQQueueManager(hostname);
    int openOptions = MQC.MQOO_INPUT_AS_Q_DEF +
                    MQC.MQOO_OUTPUT;
    MQQueue queue = mqm.accessQueue(queueName, openOptions);
    // create new Message for receiving
    MQMessage message = new MQMessage();
    // write message
    message.writeString(messageString);
    message.encoding = MQC.MQENC_NATIVE;
    message.characterSet = MQC.MQCCSI_INHERIT;
    // put message onto the queue
    queue.put(message);
    // close queue;
```

```
            queue.close();
            // disconnect from queue manager
            mqm.disconnect();
        } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

F

Справочные сведения

Это приложение содержит следующие сведения:

- Рекомендуемая литература.
- Глоссарий.

Рекомендуемая литература

В этом разделе перечислены публикации, содержащие более подробное обсуждение вопросов, рассматриваемых в этой книге.

Компания IBM предоставляет доступ к руководствам по z/OS через Интернет. Для просмотра, поиска и распечатки руководств z/OS, посетите библиотеку z/OS Internet Library:

```
http://www.ibm.com/servers/eserver/zseries/zos/bkserv
```

Справочники по архитектуре мэйнфреймов

- *z/Architecture Principles of Operation*, SA22-7832

Справочники по управлению данными в z/OS

- *z/OS DFSMS: Using Data Sets*, SC26-7410

Справочники по JCL и утилитам z/OS

- *z/OS MVS JCL Reference, SA22-7597*
- *z/OS MVS JCL User's Guide, SA22-7598*
- *z/OS DFSMSdfp Utilities*, SC26-7414

Системное программирование z/OS

- *z/OS MVS System Data Set Definition, SA22-7629*
- *z/OS MVS Initialization and Tuning Reference, SA22-7592*
- *z/OS MVS Initialization and Tuning Guide, SA22-7591*
- *JES2 Initialization and Tuning Guide*, SA22-7532

Справочники по z/OS UNIX

- *z/OS UNIX System Services Command Reference, SA22-7802*
- *z/OS UNIX System Services User's Guide, SA22-7801*
- *z/OS UNIX System Services Planning*, GA22-7800

Справочники по z/OS Communications Server

- *z/OS Communications Server IP Configuration Guide, SC31-8775*
- *z/OS Communications Server IP Configuration Reference*, SC31-8776

Справочники по языкам программирования

- *HLASM General Information, GC26-4943*
- *HLASM Installation and Customization Guide, SC26-3494*
- *HLASM Language Reference, SC26-4940*
- *Enterprise COBOL for z/OS and OS/390 V3R2 Language Reference, SC27-1408*
- *Enterprise COBOL for z/OS and OS/390 V3R2 Programming Guide, SC27-1412*
- *Enterprise PL/I Language Reference, SC27-1460*
- *Enterprise PL/I for z/OS V3R3 Programming Guide, SC27-1457*
- *C/C++ Language Reference, SC09-4764*
- *C/C++ Programming Guide, SC09-4765*
- *IBM SDK for z/OS V1.4 Program Directory, GI11-2822*
- *z/OS V1R5.0 Language Environment Concepts Guide, SA22-7567*

- *z/OS V1R5.0 Language Environment Programming Guide, SA22-7561*
- *The REXX Language, 2nd Ed., Cowlishaw, ZB35-5100*
- *Procedures Language Reference (Level 1), C26-4358 SAA CPI*
- *REXX on zSeries V1R4.0 User's Guide and Reference, SH19-8160*
- *Creating Java Applications Using NetRexx*, SG24-2216

Электронная информация доступна по адресу:

`http://www.ibm.com/software/awdtools/REXX/language/REXXlinks.html`

Справочники по CICS

- *CICS Application Programming Primer, SC33-0674*
- *CICS Transaction Server for z/OS - CICS Application Programming Guide, SC34-6231*
- *CICS Transaction Server for z/OS - CICS System Programming Reference*, SC34-6233

Справочники по IMS

- *An Introduction to IMS, ISBN 0-13-185671-5*
- *IMS Application Programming: Design Guide, SC18-7810*
- *IMS Application Programming: Database Manager, SC18-7809*
- *IMS Application Programming: Transaction Manager, SC18-7812*
- *IMS Java Guide and Reference*, SC18-7821

Электронная информация доступна по адресу:

`http://www.ibm.com/ims`

Справочники по DB2

- *DB2 UDB for z/OS: Administration Guide, SC18-7413*
- *DB2 UDB for z/OS: Application Programming and SQL Guide, SC18-7415*
- *DB2 UDB for z/OS: SQL Reference*, SC18-7426

Справочники по WebSphere MQ

- *WebSphere MQ Application Programming Guide, SC34-6064*
- *WebSphere MQ Bibliography and Glossary, SC34-6113*
- *WebSphere MQ System Administration Guide*, SC34-6068

Электронная информация доступна по адресу:

`http://www.ibm.com/software/integration/mqfamily/library/manualsa/`

Книги серии IBM Redbooks

Сведения о приобретении этих публикаций см. в разделе «Как приобрести книги серии IBM Redbooks». Обратите внимание на то, что некоторые приведенные документы могут быть доступны только в электронном варианте.

- ABCs of z/OS System Programming, Volume 1: Introduction to z/OS and storage concepts, TSO/E, ISPF, JCL, SDSF, MVS delivery and installation
- ABCs of z/OS System Programming, Volume 2: z/OS implementation and daily maintenance, defining subsystems, JES2 and JES3, LPA, linklist, authorized libraries, catalogs
- ABCs of z/OS System Programming, Volume 3: Introduction to DFSMS, storage management

- ABCs of z/OS System Programming, Volume 4: Communication Server, TCP/IP, and VTAM
- ABCs of z/OS System Programming, Volume 5: Base and Parallel Sysplex, system logger, global resource serialization, z/OS system operations, automatic restart management, hardware management console, performance management
- ABCs of z/OS System Programming, Volume 6: RACF, PKI, LDAP, cryptography, Kerberos, and firewall technologies
- ABCs of z/OS System Programming, Volume 7: Infoprint Server, Language Environment, and SMP/E
- ABCs of z/OS System Programming, Volume 8: z/OS problem diagnosis
- ABCs of z/OS System Programming, Volume 9: z/OS UNIX System Services
- ABCs of z/OS System Programming, Volume 10: Introduction to z/Architecture, zSeries processor design, zSeries connectivity, LPAR concepts, and HCD
- IBM WebSphere Application Server V5.1 System Management and Configuration WebSphere Handbook Series, SG24-6195 (IBM Redbook)
- z/OS WebSphere Application Server V5 and J2EE 1.3 Security Handbook, SG24-6086 (IBM Redbook)

Сетевые ресурсы

Нижеперечисленные веб-сайты IBM можно использовать в качестве дополнительных источников информации:

- Информационный центр z/OS Basic Skills:

http://publib.boulder.ibm.com/infocenter/zoslnctr/v1r7/index.jsp

- Веб-сайт z/OS:

http://www.ibm.com/servers/eserver/zseries

- Интернет-библиотека z/OS:

http://www.ibm.com/servers/eserver/zseries/bkserv

- Веб-сайт z/OS Communications Server:

http://www.software.ibm.com/network/commserver/support/

- Терминология IBM

http://www.ibm.com/ibm/terminology/

Как приобрести книги серии IBM Redbooks

Можно выполнять поиск, просматривать и копировать книги серии Redbooks, документы серии Redpapers, публикации серии Hints and Tips, проекты публикаций и дополнительные материалы, а также заказать книги серии Redbooks в бумажном варианте или на компакт-дисках через веб-сайт:

ibm.com/redbooks

Поддержка от компании IBM

Поддержка IBM Support и загружаемые данные

ibm.com/support

G

Глоссарий

A

abend. Аварийный останов.

ACB. Access Control Block (блок управления доступом).

ACCEPT. Команда SMP/E, используемая для установки SYSMOD в дистрибутивных библиотеках.

ALLOCATE, команда. В z/OS: команда TSO/E, выступающая в качестве связи между логическим именем файла (DD-именем) и физическим именем файла (именем набора данных).

AMODE. Addressing mode (режим адресации).

ANSI. American National Standards Institute (Американский национальный институт стандартов).

AOR. Application-owning region (регион-владелец приложения).

APAR. Authorized program analysis report (авторизованный отчет об анализе программы).

APAR-исправление. Временное исправление ошибки в системной управляющей программе IBM или в лицензированной программе, влияющее на определенного пользователя. APAR-исправление обычно впоследствии заменяется постоянным исправлением PTF. APAR-исправления идентифицируются в SMP/E оператором ++APAR.

APF. Authorized program facility (средство авторизации программ).

API. Application programming interface (интерфейс программирования приложений).

APPC. Advanced Program-to-Program Communications (Усовершенствованный интерфейс связи между программами).

APPLY. Команда SMP/E, используемая для установки SYSMOD в целевых библиотеках.

APPN. Advanced Peer-to-Peer Network (улучшенный протокол одноранговых сетей).

ARM. Automatic restart manager (менеджер автоматического перезапуска).

ASCII. American Standard Code for Information Interchange (американский стандартный код для обмена информацией).

ASID. Address space identifier (идентификатор адресного пространства).

ASSEM-запись. В SMP/E: запись, содержащая операторы ассемблера, которые можно ассемблировать, создав, таким образом, объектный модуль.

ATM. Automated teller machine (банкомат).

B

BAL. Basic Assembler Language.

BCP. Base control program (базовая управляющая программа).

BLK. Подпараметр параметра SPACE оператора DD. Указывает, что пространство выделяется блоками.

BLKSIZE. Block size (размер блока).

BLOB. Binary large object (двоичный большой объект).

BPAM. Basic partitioned access method (базисный библиотечный метод доступа).

BSAM. Basic sequential access method (базисный последовательный метод доступа).

C

CART. Command and response token (маркер команд и ответов).

CCW. Channel command word (управляющее слово канала).

CEMT. CICS-транзакция, позволяющая выполнить проверку состояния терминалов, подключений и других CICS-сущностей с консоли или из терминальных сеансов CICS.

CF. Coupling Facility (устройство сопряжения)

CFRM. Coupling Facility resource management (управление ресурсами устройства сопряжения).

CGI. Common Gateway Interface (общий шлюзовой интерфейс).

CHPID. Channel path identifier (идентификатор канального пути).

CI. Control interval (управляющий интервал).

CICS. См. *Customer Information Control System.*

CICSplex. Конфигурация, состоящая из связанных CICS-систем, в которых каждая система выделена для одного из основных элементов общей рабочей нагрузки. См. также *регион-владелец приложения* и *регион-владелец терминала.*

CKD. См. *Count-Key Data.*

CLIST. Command list (командный список).

CLOB. Character large object (символьный большой объект).

CLPA. Create Link Pack Area (создание области загрузки модулей).

CMOS. Complementary Metal Oxide Semiconductor (комплементарный металл-оксид-полупроводник).

CMS. См. *Conversational Monitor System.*

COBOL. Common Business-Oriented Language.

COMMAREA. Область связи, доступная для приложений, выполняющихся под управлением CICS.

Common Business-Oriented Language (COBOL). Высокоуровневый язык, сходный с английским, используемый преимущественно для создания бизнес-приложений.

Conversational Monitor System (CMS). Операционная система виртуальной машины, обеспечивающая разделение общего интерактивного времени, разрешение проблем и возможность разработки программ и работающая только под контролем программы управления VM/370.

CORBA. Common Object Request Broker Architecture.

Count-Key Data. Устройство дисковой памяти, предназначенное для хранения данных в следующем формате: поле счетчика, за которым идет поле ключа, и за которыми идут непосредственно данные записи. Поле счетчика содержит, помимо прочей информации, адрес записи в формате CCHHR (где CC – двухзначный номер цилиндра, HH – двухзначный номер головки и R – номер записи) и длину данных. Поле ключа содержит ключ записи.

Coupling Facility (устройство межсистемного сопряжения). Специальный логический раздел, обеспечивающий высокоскоростное кеширование, обработку списков и функции блокирования в сисплексе.

CP. Central processor (центральный процессор).

CPC. Central processor complex (центральный процессорный комплекс).

CPU. Central processing unit (центральное процессорное устройство).

create link pack area (CLPA). Опция, используемая при начальной загрузке, для инициализации перемещаемой области загрузки модулей.

CSA. Common service area (общая сервисная область).

CSECT. Control section (управляющий секция).

CSI. Консолидированный набор данных о составе программного обеспечения. См. *SMPCSI.*

CSS. Channel subsystem (канальная подсистема).

CTC. Канал-канал.

CTC-адаптер (CTCA). Устройство ввода-вывода, используемое для связи программы в одной системе с программой в другой системе.

CTC-подключение. Подключение канал-канал.

CTC-подключение. Подключение между двумя CHPID на одном или разных процессорах, либо прямое, либо через коммутатор. При подключении через коммутатор оба CHPID должны быть подключены через один или через последовательно подключенные коммутаторы .

Customer Information Control System (CICS). Система обработки оперативных транзакций (OLTP), обеспечивающая специальные интерфейсы к базам данных, файлам и терминалам для поддержки бизнес-приложений и коммерческих приложений. CICS позволяет одновременно обрабатывать транзакции, введенные на удаленных терминалах, пользовательскими приложениями.

D

DASD. Direct access storage device (устройство хранения с прямым доступом).

DASD-том. Пространство DASD, идентифи-

цируемое общей меткой, и доступ к которому осуществляется по набору связанных адресов. См. также *том*.

Data Facility Sort (DFSORT). Лицензированная программа IBM, представляющая собой высокоскоростную утилиту обработки данных. DFSORT осуществляет операции сортировки, объединения и копирования, а также универсальной обработки данных на уровне записей, полей и битов.

DB2. DATABASE 2; в общем смысле: одно из семейств систем управления реляционными базами данных IBM; в частности: система, работающая под управлением z/OS.

DBCS. Double-byte character set (двухбайтовый набор символов).

DCB. Data control block (блок управления данными).

DCLGEN. Declarations generator (генератор определений).

DD-имя. Имя определения данных.

DD-оператор. Оператор определения данных.

DFS. См. *Distributed File Service*.

DFSMS. См. *Data Facility Storage Management Subsystem*.

DFSMShsm. Продукт IBM, используемый для резервного копирования и восстановления данных, а также для управления пространством томов в иерархии хранения.

DFSORT. См. *Data Facility Sort*.

DISP. Диспозиция (параметр JCL DD).

Distributed File Service (DFS). Компонент DCE. DFS объединяет локальные файловые системы нескольких машин, выступающих в качестве файловых серверов, делая файлы одинаково доступными для всех клиентских машин DFS. DFS позволяет пользователям осуществлять доступ и совместно использовать файлы, хранящиеся на файловом сервере, с любого узла сети, независимо от физического расположения файла. Файлы являются частью единого глобального пространства имен, так что пользователь может работать с любого узла системы, используя одинаковое имя.

DLIB. Distribution library (дистрибутивная библиотека).

DLL. Dynamic link library (динамически подключаемая библиотека).

dsname. Имя набора данных.

DSORG. Организация набора данных (параметр DCB и DD, а также определения класса данных).

E

EBCDIC. См. *Extended Binary Coded Decimal Interchange Code*.

EC. Engineering change (техническое изменение).

ECSA. Extended common service area (расширенная общая сервисная область).

EDT. Eligible device table (таблица подходящих устройств).

Enterprise Systems Connection (ESCON). Набор продуктов и служб, обеспечивающий динамически связанную среду, использующую в качестве среды передачи оптоволоконные кабели.

EOF. End of file (конец файла).

ESCON. Enterprise Systems Connection.

ETR. External Time Reference. См. также *Sysplex Timer®*.

EXCP. Execute channel programs (выполнение канальных программ).

Extended Binary-Coded Decimal Interchange Code (EBCDIC). Схема кодирования, используемая для представления символьных данных в среде z/OS. Сравните: *ASCII* и *Unicode*.

F

Fiber Connection Environment (FICON). Метод оптоволоконной связи, использующий каналы с высокой скоростью передачи данных, высокой пропускной способностью, большими расстояниями передачи и большим количеством устройств на устройство управления для мэйнфрейм-систем. Может работать совместно с ESCON-каналами или вместо них.

FICON. См. *Fiber Connection Environment*.

FIFO. First In, First Out («первым пришел – первым обслужен»).

FILEDEF. Оператор определения файла.

First In, First Out («первым пришел –первым обслужен»). Дисциплина очереди, в которой следующий извлекаемый элемент является самым старым элементом в очереди.

FlashCopy. Функция создания мгновенной копии, которая может быстро копировать данные из исходного расположения в целевое расположение.

FMID. Function modification identifier (идентификатор функциональных изменений).

Fortran. Высокоуровневый язык, используемый преимущественно для приложений, включающих численные расчеты. Ранее имя языка писа-

лось заглавными буквами: FORTRAN.

FTP. File Transfer Protocol (протокол передачи файлов).

G

GDG. Generation data group (группа поколений данных).

H

HASP. См. *Houston Automatic Spooling Priority*.

HCD. Hardware Configuration Definition.

HFS. Hierarchical File System (иерархическая файловая система).

HLL. Высокоуровневый язык.

HMC. Hardware Management Console (консоль управления аппаратными средствами).

HOLDDATA. В SMP/E: одна или несколько MCS, используемых для того, чтобы указать, что определенные SYSMOD содержат ошибки или требуют специальной обработки перед установкой. Операторы ++HOLD и ++RELEASE используются для определения HOLDDATA. SYSMOD, затрагиваемые HOLDDATA, называются исключительными SYSMOD.

Houston Automatic Spooling Priority (HASP). Компьютерная программа, содержащая дополнительные функции управления заданиями, управления данными и управления задачами, в частности: управление потоком заданий, упорядочение задач и спулинг. См. также: *JES2*.

I

I/O. Input/output (ввод-вывод).

IBM Support Center (Центр поддержки IBM). Организация в составе IBM, отвечающая за обслуживание программного обеспечения.

ICSF. Integrated Cryptographic Service Facility.

IDCAMS. Программа IBM, используемая для обработки команд служб метода доступа. Она может быть вызвана как задание или шаг задания с терминала TSO или из пользовательского приложения.

IMS DB. Information Management System Database Manager.

IMS. См. *Information Management System*.

Information Management System (IMS). Продукт компании IBM, поддер-живающий иерархические базы данных, передачу данных, обработку транзакций, а также откат и восстановление баз данных.

Interactive Problem Control System (IPCS). Компонент z/OS, позволяющий осуществлять управление проблемами, интерактивную диагностику проблем, оперативную отладку с помощью дампов, трассировку проблем и генерирование отчетов о проблемах.

Interactive System Productivity Facility (ISPF). Диалоговый менеджер интерактивных приложений. Обеспечивает управление и функции, поддерживающие работу диалоговых окон.

IOCDS. Набор данных конфигурации ввода-вывода.

IODF. Файл определения ввода-вывода.

IPCS. См. *Interactive Problem Control System*.

IPL. Initial program load (начальная загрузка).

IPv6. Internet Protocol Version 6.

ISMF. Interactive Storage Management Facility.

ISPF. См. *Interactive System Productivity Facility*.

ISPF/PDF. Interactive System Productivity Facility/Program Development Facility.

IVP. Процедура проверки инсталляции.

J

JCL. Job Control Language (язык управления заданиями).

JES. Job Entry Subsystem (подсистема ввода заданий).

JES2. Подсистема z/OS, принимающая задания в системе, преобразующая их во внутренний формат, выбирающая задания для выполнения, обрабатывающая их выходные данные и удаляющая их из системы. В инсталляции, имеющей несколько процессоров, среди которых каждый процессор JES2 независимо управляет вводом заданий, планированием и обработкой выходных данных. Сравните: *JES3*.

JES3. Подсистема z/OS, принимающая задания в системе, преобразующая их во внутренний формат, выбирающая задания для выполнения, обрабатывающая их выходные данные и удаляющая их из системы. В комплексах, имеющих несколько слабосвязанных процессорных устройств, JES3 управляет процессорами таким образом, что глобальный процессор осуществляет централизованное управление локальными процессорами и распределяет задания между ними через общую очередь заданий. Сравните: *JES2*.

job entry subsystem 2. См. *JES2*.

job entry subsystem 3. См. *JES3*.

K

KSDS. Key-sequenced data set (набор данных, упорядоченный по ключам).

L

LAN. Local area network (локальная сеть).

Language Environment. Сокращенная форма названия z/OS Language Environment. Набор архитектурных конструкций и интерфейсов, обеспечивающий общую среду выполнения и содержащий службы времени выполнения для приложений на языках C, C++, COBOL, Fortran, PL/I, VisualAge PL/I и Java, скомпилированные компиляторами, совместимыми с Language Environment.

Last In, First Out («последним пришел – первым обслужен»). Дисциплина очереди, в которой следующий извлекаемый элемент является самым новым элементом в очереди.

LCSS. Logical channel subsystem (логическая канальная подсистема).

LCU. Logical control unit (логическое устройство управления).

LDAP. Lightweight Directory Access Protocol.

LIC. Licensed Internal Code (лицензированный внутренний код).

LIFO. Last In, First Out («первым пришел – первым обслужен»).

Lightweight Directory Access Protocol (LDAP). Стандарт Интернет-протокола, основанного на TCP/IP, позволяющий осуществлять доступ и управление данными, организованными в виде Directory Information Tree (DIT).

LMOD. В SMP/E: сокращение от «load module» (загрузочный модуль).

LP. Logical partition (логический раздел).

LPA. Link pack area (область загрузки модулей).

LPAR. Logically partitioned mode (режим логического разделения).

LRECL. Logical record length (длина логической записи).

LSQA. Local system queue area (локальная область системных очередей)

LU. Logical unit (логическое устройство).

M

MAS. Конфигурация Multi-Access Spool.

MCS. (1) Multiple Console Support (Многоконсольная поддержка). (2) Modification control statement (оператор управления модификацией; в SMP/E).

Multiple console support (MCS). Интерфейс оператора в системе z/OS.

Multiple Virtual Storage (MVS). Ранний вариант операционной системы z/OS.

MVS. Multiple Virtual Storage.

MVS/ESA™. Multiple Virtual Storage/Enterprise Systems Architecture.

N

NCP. Network Control Program.

Network File System. Компонент z/OS, обеспечивающий удаленный доступ к данным базового процессора z/OS с рабочих станций, персональных компьютеров или с любой другой системы в TCP/IP-сети, использующей клиентское программное обеспечение для протокола Network File System.

NIP. Nucleus initialization program (программа инициализации ядра).

n-процессорный. Содержащий несколько (n) центральных процессоров (CP) в CPC. Например, 6-процессорный CPC содержит шесть CP.

O

OS/390. Вариант операционной системы, предшествующей z/OS.

P

Parallel Sysplex. Сисплекс, использующий одно или несколько устройств сопряжения.

PCHID. Physical channel identifier (идентификатор физического канала).

PE. См. *PE-PTF*.

PE-PTF (program error PTF, PTF с программной ошибкой). PTF, в котором была обнаружена ошибка. PE-PTF указывается в операторе ++HOLD ERROR, вместе с APAR, который первым сообщил об ошибке.

PL/I. Универсальный высокоуровневый язык для научных и коммерческих приложений. PL/I представляет собой мощный процедурный язык, особенно хорошо подходящий для решения сложных научных задач или для выполнения длительных и сложных бизнес-транзакций, а также приложений учета.

PLPA. Pageable link pack area (перемещаемая область загрузки модулей).

Portable Operating System Interface (POSIX). Интерфейс переносимой операционной системы для вычислительных сред; стандарт интерфейса, регулируемый IEEE и основанный на UNIX. POSIX не является продуктом. Скорее это семейство развивающихся стандартов, описывающих широкий спектр компонентов операционной системы, от языка C и интерфейсов оболочки до системного администрирования.

POSIX. Portable Operating System Interface (интерфейс переносимой операционной системы).

PPRC. Peer-to-peer remote copy (одноранговое удаленное копирование).

PPT. В z/OS: таблица свойств программы (program properties table).

Processor Resource/Systems Manager™ (PR/SM). Средство, позволяющее процессору одновременно использовать несколько образов z/OS и осуществляющее создание логических разделов. См. также *LPAR*.

PSP. Preventive service planning (планирование профилактического обслуживания).

PSW. Program status word (слово состояния программы).

PTF. Program temporary fix (временное программное исправление).

Q

QSAM. Queued sequential access method (последовательный метод доступа с очередями).

R

RACF. См. *Resource Access Control Facility*.

RDW. Record descriptor word (дескриптор записи).

RECEIVE. SMP/E-команда, используемая для чтения SYSMOD и других данных из SMPPTFIN и SMPHOLD.

RECFM. Record format (формат записи).

RESTORE. SMP/E-команда, используемая для удаления примененных SYSMOD из целевых библиотек.

Restructured Extended Executor (REXX). Универсальный процедурный язык программирования для конечного пользователя, разработанный для упрощения работы обычных пользователей и компьютерных специалистов. Он также полезен для написания макросов приложений. REXX включает возможность передачи команд от этих макросов и процедур базовой операционной системе.

RIM. Related installation material (дополнительные материалы по установке).

RJE. Remote job entry (удаленный ввод заданий).

RMF. Resource Measurement Facility (средство измерения ресурсов).

RMODE. Residency mode (режим размещения).

RSA. Register save area (область хранения регистров).

S

SAF. System authorization facility (средство авторизации системы).

SAP. System Assistance Processor (вспомогательный системный процессор).

SDSF. System Display and Search Facility (средство отображения и поиска).

SIGP. Signal Processor (команда сигнала процессору).

SLA. Service level agreement (соглашение об уровне сервиса).

SMF. System management facilities (средства управления системой).

SMP/E. System Modification Program/Extended (программа модификации системы/расширенная).

SMPCSI. Набор данных SMP/E, содержащий информацию о структуре пользовательской системы, а также информацию, необходимую для установки операционной системы на пользовательской системе. Оператор SMPCSI DD обращается конкретно к CSI, содержащему глобальную зону. Также называется главным CSI.

SMS. Storage Management Subsystem (подсистема управления памятью).

SNA. Systems Network Architecture (системная сетевая архитектура).

spool. Simultaneous peripheral operations online; спул.

SPUFI. SQL Processing Using File Input (обработка SQL с файловым вводом).

SQA. System queue area (область системных очередей).

SQL. Structured Query Language (язык структурированных запросов).

SREL. System release identifier (идентификатор выпуска системы).

SRM. System resources manager (менеджер системных ресурсов).

SSID. Subsystem identifier (идентификатор подсистемы).

Storage Management Subsystem (SMS). Средство, используемое для автоматизации и централизации управления памятью. Используя SMS, администратор внешней памяти описывает свойства размещения данных, цели производительности и доступности, требования резервного копирования и сохранения, а также требования хранения, используя определения класса данных, класса памяти, класса управления, группы памяти и ACS.

SVC. Инструкция вызова супервизора.

SVC-прерывание. Прерывание, вызванное выполнением инструкции вызова супервизора, вызывающее передачу управления супервизору.

SVC-программа. Управляющая программа, выполняющая или запускающая службу управляющей программы, заданную инструкцией вызова супервизора.

SWA. Scheduler work area (рабочая область планировщика).

SYSIN. Поток входных данных системы; это имя также используется как имя определения данных набора данных во входном потоке.

SYSLIB. (1) Подзапись, используемая для идентификации целевой библиотеки, в которой установлен элемент. (2) Набор сцепленных библиотек макросов, используемых ассемблером. (3) Набор программ, используемых утилитой редактирования связей для разрешения неразрешенных внешних ссылок.

SYSLOG. System log (системный журнал).

SYSMOD. System modification (системная модификация).

SYSOUT. Поток выходных данных системы; это имя также используется в операторах определения данных, чтобы указать, что набор данных подлежит записи в устройство системного вывода.

SYSOUT-класс. Категория выходных данных с определенными свойствами, записываемая на определенное устройство вывода. Каждая система имеет собственный набор SYSOUT-классов, обозначаемый буквой от A до Z, числом от 0 до 9 или символом *.

Sysplex Timer. Устройство производства IBM, синхронизующее часы на разных процессорах или компонентах процессора.

SYSRES. System residence disk (резидентный системный диск).

System Modification Program Extended (SMP/E). Программный продукт компании IBM или элемент OS/390 или z/OS, используемый для установки программного обеспечения и программных изменений в системах z/OS. SMP/E консолидирует данные установки, обеспечивая более высокую гибкость при выборе устанавливаемых изменений, обеспечивает диалоговый интерфейс и поддерживает динамическое распределение наборов данных. SMP/E является основным средством управления изменениями в операционной системе z/OS.

Systems Network Architecture (SNA). Описание логической структуры, форматов, протоколов и операционных последовательностей передачи единиц информации, а также управления конфигурацией и эксплуатацией сетей.

T

TCB. Task control block (блок управления задачей).

TCP/IP. Transmission Control Protocol/Internet Protocol.

TGTLIB. Target library (целевая библиотека).

Time Sharing Option/Extensions (TSO/E). Средство в z/OS, позволяющее пользователям осуществлять интерактивное разделение компьютерного времени и ресурсов.

TLIB. Target library (целевая библиотека).

Transmission Control Protocol/Internet Protocol (TCP/IP). Аппаратно-независимый коммуникационный протокол, используемый для связи между физически отделенными компьютерами. Был разработан для упрощения связи между компьютерами, расположенными в разных физических сетях.

Transport Layer Security (TLS). Протокол, обеспечивающий конфиденциальность коммуникаций в Интернете.

TRK. Подпараметр параметра SPACE оператора DD. Указывает, что в качестве единицы распределения пространства следует использовать дорожки.

TSO. Time Sharing Option (система с разделением времени). См. *Time Sharing Option/Extensions (TSO/E)*.

TSO/E. Time Sharing Option/Extensions (система с разделением времени/расширенная).

U

UCB. Unit control block (блок управления устройством).

UCLIN. В SMP/E: команда, используемая для инициации изменений в наборах данных SMP/E. Непосредственно изменения вносятся последующими UCL-операторами.

UIM. Unit information module (информационный модуль устройства).

Unicode. Стандарт универсальной кодировки символов, поддерживающий обмен, обработку и вывод текста, написанного на любом из современных языков. Он также может поддерживать множество классических и исторических текстов; этот стандарт постоянно расширяется.

UNIX. См. *z/OS UNIX System Services*.

UNLOAD. SMP/E-команда, используемая для копирования данных из записей набора данных SMP/E в форме UCL-операторов.

USERMOD. User modification (пользовательская модификация).

V

VB. Variable Blocked (переменный блочный).

VIO. Virtual input/output (виртуальный ввод-вывод).

Virtual Telecommunications Access Method (VTAM). Набор программ, осуществляющих управление связью между терминалами и приложениями, работающими под управлением z/OS.

VM. Virtual Machine (виртуальная машина).

VOLSER. Volume serial number (серийный номер тома).

VPN. Virtual private network (виртуальная частная сеть).

VSAM. Virtual storage access method (виртуальный метод доступа).

VTAM. Virtual Telecommunications Access Method (виртуальный телекоммуникационный метод доступа).

VTOC. Volume table of contents (оглавление тома).

W

WAP. Wireless access point (точка беспроводного доступа).

WLM. Workload manager (менеджер рабочей нагрузки).

WTO. Write-to-operator.

WTOR. Write-to-operator-with-reply.

X

XA. Extended Architecture (расширенная архитектура).

XCF. Cross-system coupling facility (межсистемное средство сопряжения).

Z

z/Architecture. Архитектура IBM для мэйнфрейм-компьютеров и периферийных устройств. Семейство серверов zSeries использует архитектуру z/Architecture.

z/OS Language Environment. Программный продукт IBM, обеспечивающий общую среду выполнения и общие службы времени выполнения для совместимых компиляторов высокоуровневых языков.

z/OS UNIX System Services (z/OS UNIX). Службы z/OS, поддерживающие UNIX-подобную среду. Пользователи могут переключаться между традиционным интерфейсом TSO/E и интерфейсом оболочки. Пользователи с навыками работы в UNIX могут взаимодействовать с системой, используя знакомый набор стандартных команд и утилит. Пользователи с навыками работы в z/OS могут взаимодействовать с системой, используя знакомые команды и интерактивные меню TSO/E для создания и управления файлами иерархической файловой системы и для копирования данных между наборами данных z/OS и файлами. Программистам приложений и пользователям доступны оба набора интерфейсов на выбор, и при выборе соответствующих компромиссных решений, можно выбрать сочетание этих интерфейсов.

z/OS. Широко используемая операционная система для мэйнфрейм-компьютеров IBM, использующая 64-разрядную основную память.

zAAP. См. *zSeries Application Assist Processor (вспомогательный процессор zSeries для приложений)*.

zFS. См. *zSeries File System (файловая система zSeries)*.

zSeries Application Assist Processor (zAAP). Специализированное вспомогательное процессорное устройство, настроенное на выполнение Java-программ на компьютерах zSeries.

zSeries File System (zFS). Файловая система z/OS UNIX, хранящая файлы в линейных наборах данных VSAM.

А

аварийное восстановление. Восстановление после аварии (например, после пожара), повредившей или другим способом нарушившей работу системы. Аварийное восстановление обычно включает восстановление данных на другой системе (системе восстановления) и использование системы восстановления вместо поврежденной системы. См. также *восстановление*, *резервное копирование* и *система восстановления*.

аварийное завершение. (1) Завершение обработки до запланированного завершения. (2) Неуспешное завершение, вызванное системным сбоем или действиями оператора. Синонимы: *abend*, *аварийный останов*.

аварийный останов. Завершение задачи, задания или подсистемы в связи с возникновением ошибки, которую нельзя разрешить средствами восстановления при выполнении задачи. См. также *аварийное завершение*.

автоматизированные операции. Автоматизированные процедуры, призванные заменить или упростить действия операторов в области как системных, так и сетевых операций.

автоматическая библиотека вызовов. Содержит загрузочные модули или объектные модули, предназначенные для использования в качестве вторичных входных данных редактора связей при разрешении внешних символов, оставшихся неопределенными после обработки первичных входных данных.

Автоматическая библиотека вызовов может включать:

- библиотеки, содержащие объектные модули, и, возможно, управляющие операторы редактора связей;
- библиотеки, содержащие загрузочные модули;
- библиотеку, содержащую программы времени выполнения Language Environment.

автоматический вызов. Процесс, используемый редактором связи для разрешения внешних символов, оставшихся неопределенными после обработки первичных входных данных. См. также *автоматическая библиотека вызовов*.

автоматический перезапуск. Перезапуск, выполняемый во время текущего выполнения, т. е. без повторной передачи задания. Автоматический перезапуск может выполняться в ходе шага задания или в начале шага задания. Сравните: *отложенный перезапуск*. См. также *перезапуск с контрольной точки*.

автономный (offline). Относится к оборудованию или устройствам, не находящимся под управлением процессора.

авторизованный отчет об анализе программы (APAR). Запрос на исправление проблемы, вызванной дефектом текущей неизменной версии программы. Исправление называется *APAR-исправлением*.

адаптер каналов. Устройство, осуществляющее электронное группирование двух или более контроллеров канальных интерфейсов.

администратор. Сотрудник, отвечающий за административные задачи, такие как авторизация доступа и управление содержимым. Администраторы могут также назначать уровни полномочий пользователей.

администратор базы данных (DBA). Ответственный за проектирование, разработку, эксплуатацию, защиту, поддержку и использование базы данных.

администратор безопасности. Программист, управляющий, защищающий и контролирующий доступ к конфиденциальной информации.

администратор внешней памяти. Сотрудник вычислительного центра, отвечающий за определение, реализацию и поддержку политик управления памятью.

администрирование производительности. Процесс определения и настройки целей управления нагрузкой и группами ресурсов на основе бизнес-целей инсталляции.

адрес подключения канала (CCA). Адрес ввода-вывода, уникально идентифицирующий устройство ввода-вывода для канала во время операции ввода-вывода.

адрес. Уникальный код, присваиваемый каждому устройству, рабочей станции или системе, подключенной к сети.

адрес устройства. Поле фрейма уровня ESCON-устройства, выбирающее определенное устройство в составе устройства управления. Один-два крайних левых знака являются адресом канала, к которому подключено устройство. Два крайних правых знака представляют адрес устройства.

адрес устройства управления. Старшие биты адреса устройства управления памятью, используемые для определения устройства управления в базовой системе.

адресат. Сочетание имени узла и одного из следующих свойств: идентификатор пользователя, удаленный принтер или перфоратор, специальный локальный принтер или LOCAL (используется по умолчанию, если задано имя узла).

адресное пространство. Полный диапазон адресов, доступных программе. В z/OS адресное пространство может занимать до 16 экзабайт непрерывных адресов виртуальной памяти, создаваемых системой для пользователя. Адресное пространство содержит пользовательские данные и программы, а также системные данные и программы, некоторые из которых являются общими для всех адресных пространств. См. также *виртуальное адресное пространство*.

анклав. Транзакция, которая может охватывать несколько диспетчеризуемых единиц работы (SRB и задач) в одном или нескольких адресных пространствах, и которые отображаются и управляются как одно целое.

аппаратное обеспечение. Физическое оборудование, в отличие от компьютерной программы или метода использования; например, механические, магнитные, электрические или электронные устройства. Сравните: *программное обеспечение*.

аппаратное устройство. Центральный процессор, элемент памяти, канальный путь, устройство и т. д.

асинхронная обработка. Набор операций, выполняемых отдельно от задания, из которо-

го они были запрошены; например, передача пакетного задания из интерактивного задания, выполняемого на рабочей станции. См. также *синхронная обработка*.

ассемблер (язык). Символьный язык программирования, содержащий инструкции для основных компьютерных операций, структурированные в соответствии с форматами данных, структурами памяти и регистрами компьютера.

ассемблер. Компьютерная программа, преобразующие инструкции языка ассемблера в двоичный машинный язык (объектный код).

аудит. Обзор и изучение действий системы обработки данных, выполняемые, главным образом, для тестирования адекватности и эффективности процедур безопасности и обеспечения точности данных.

Б

база данных. Набор таблиц или набор табличных пространств и индексных пространств.

базовая функция. В SMP/E: SYSMOD, определяющий элементы базовой системы z/OS или других продуктов, ранее не существовавшие в целевых библиотеках. Базовые функции задаются в SMP/E оператором ++FUNCTION. Сам SMP/E представляет собой пример базовой функции z/OS.

байт. Базовая единица адресуемости памяти. Имеет длину 8 бит.

библиотека. Секционированный набор данных (PDS), содержащий связанный набор именованных разделов. См. *секционированный набор данных*.

библиотека параметров (parmlib). Все разделы PDS SYS1.PARMLIB, содержащие параметры, ограничивающие и контролирующие работу z/OS.

блок управления данными (DCB). Блок управления, используемый программами метода доступа при сохранении и извлечении данных.

блок управления задачей (TCB). Структура данных, содержащая информацию и указатели, связанные с обрабатываемой задачей.

блок управления устройством. Аппаратное устройство, управляющее чтением, записью или выводом данных в одно или несколько устройств ввода-вывода или терминалов.

блокировочная структура. Структура устройства сопряжения, позволяющая приложениям в сисплексе, использовать настраиваемые протоколы блокировки для синхронизации ресурсов приложений. Блокировочная структура поддерживает разделяемую блокировку, исключительную блокировку и блокировку, определяемую приложением, а также общее управление конфликтами и протоколы восстановления.

брандмауэр. Промежуточный сервер, отделяющий безопасную сеть от небезопасной сети.

буфер. Область памяти, используемая для временного хранения входных или выходных данных.

В

ведущая система. Система, используемая для установки программы. Сравните: *целевая система*.

версия. Отдельная лицензированная программа, основанная на существующей лицензированной программе и обычно содержащая важный новый код или новые функции. Сравните: *выпуск* и *уровень модификации*.

ветвление. Создание и запуск дочернего процесса. Ветвление подобно созданию адресного пространства и подключению подзадачи. При этом создается копия родительского процесса, включая дескрипторы открытых файлов.

виртуальная память. (1) Пространство памяти, в которой виртуальные адреса отображаются в реальные адреса и которое пользователь компьютерной системы воспринимает как адресуемую основную память. Размер виртуальной памяти ограничен схемой адресации компьютерной системы и количеством вспомогательной памяти, а не действительным количеством ячеек основной памяти. (2) Схема адресации, позволяющая использовать внешнюю дисковую память как основную память.

виртуальное адресное пространство. В системах виртуальной памяти: виртуальная память, назначаемая заданию, пользователю терминала или системной задаче. См. также *адресное пространство*.

виртуальный ввод-вывод (VIO). Распределение наборов данных, существующих только в страничной памяти.

виртуальный метод доступа (VSAM). Метод доступа, предназначенный для прямой или последовательной обработки записей фиксированной и переменной длины на устройствах с прямым доступом. Записи в наборе данных или файле VSAM могут быть организованы в логическую последовательность по ключевому полю (ключевой последовательности), в физическую последовательность, при которой они записываются в набор дан-

ных или файл (последовательности добавления), или по относительнму номеру записи.

внешний ключ. Столбец или набор столбцов в зависимой таблице ссылочного ограничения. Ключ должен иметь такое же количество столбцов, с такими же описаниями, как и первичный ключ родительской таблицы. Каждое значение внешнего ключа должно либо соответствовать значению родительского ключа в связанной родительской таблице, либо быть пустым.

внешняя ссылка. В объектном модуле: ссылка на символьное имя (например, на имя точки входа), определенное в другой программе или модуле.

внутренний считыватель. Средство, осуществляющее передачу заданий в JES.

возврат каретки (CR). (1) Клавиша, обычно указывающая конец командной строки. (2) В текстовых данных: действие, указывающее на продолжение печати с левой границы следующей строки. (3) Символ, вызывающий запуск печати с начала той физической строки, в которой произошел возврат каретки.

возврат. Запрос на удаление всех изменений ресурсов с момента последней фиксации или отката или, для первой единицы восстановления, с начала выполнения приложения. Возврат также называют *откатом* или *аварийным прекращением*.

восстановление (recovery). Процесс реконструирования данных после их повреждения, часто посредством восстановления резервной версии данных или путем повторного применения транзакций, записанных в журнале.

восстановление (restore). В SMP/E: удаление примененных SYSMOD из целевых библиотек с использованием команды RESTORE.

восходящая совместимость. Способность приложений продолжать работать в последующих версиях z/OS без перекомпиляции или перекомпоновки.

временное программное исправление (PTF). Временное решение или обход проблемы, диагностируемой IBM как результат дефекта в текущем неизмененном выпуске программы.

временный набор данных. Набор данных, создаваемый и удаляемый в одном задании.

время выполнения. Любой промежуток времени, в который происходит выполнение программы. Синоним: *время исполнения*.

время ожидания. (1) Состояние задачи, требующее возникновения одного или нескольких событий для перехода в состояние готовности. (2) Состояние единицы обработки, когда все операции приостановлены.

вспомогательная память. Вся адресуемая память, за исключением процессорной памяти.

вход в систему. (1) Процедура запуска пользователем сеанса терминала. (2) В VTAM: запрос на подключение терминала к VTAM-приложению.

выгрузка. В SMP/E: копирование данных из записей набора данных SMP/E в форме UCL-операторов с использованием команды UNLOAD.

выделение зарезервированной памяти. Объем основной и расширенной памяти, который можно динамически конфигурировать для логического раздела оперативным или автономным способом.

выделенный. Относится к выделению системного ресурса (устройства, программы или целой системы) приложению или определенной цели.

вызов автоматической библиотеки. Автоматический вызов. См. также *автоматическая библиотека вызовов*.

вызов супервизора (SVC). Инструкция, прерывающая выполняемую программу и передающую управление супервизору таким образом, чтобы он мог выполнять определенную функцию, заданную инструкцией.

вызываемая программа. Программа, вызванная другой программой.

выпуск. Дистрибутив нового продукта или новой функции и APAR-исправления для существующего продукта. Сравните: *уровень модификации* и *версия*.

высокий параллелизм. Относится к нескольким параллельно работающим системам, каждая из которых может иметь несколько процессоров. См. также *n-процессорный*.

высокоуровневый язык (HLL). Язык программирования более высокого уровня, чем ассемблер, но более низкого уровня, чем генераторы программ и языки запросов. Примерами являются C, C++, COBOL, Fortran и PL/I.

выход из системы. (1) Процедура завершения пользователем сеанса терминала. (2) В VTAM: запрос на отключение терминала от VTAM-приложения.

Г

ГБ. Гигабайт (1 073 741 824 байт).

генератор определений (DCLGEN). Подкомпонент DB2, генерирующий определения

SQL-таблицы и определения структуры данных COBOL, C или PL/I, соответствующие таблице. Определения генерируются на основании информации системного каталога DB2.

географически распределенный параллельный сисплекс (GDPS). Приложение, интегрирующее технологию Parallel Sysplex с технологией удаленного копирования, повышая доступность приложений и улучшая возможности аварийного восстановления. Топология GDPS представляет собой кластер Parallel Sysplex, размещенный на двух узлах, в котором выполняется зеркальное отображение всех критически важных данных между узлами. GDPS управляет конфигурацией удаленного копирования и подсистемами хранения, автоматизирует операторские задачи Parallel Sysplex и автоматизирует восстановление после отказов с единого пункта управления.

гигабайт. 2^{30} байт, 1 073 741 824 байт. Составляет приблизительно миллиард байтов.

главная задача. В контексте многозадачности z/OS, главная программа в многозадачной среде.

главный IODF. Централизованно хранящийся IODF, содержащий определения ввода-вывода для нескольких систем или даже для полной структуры предприятия. Главные IODF позволяют поддерживать согласованность данных ввода-вывода и обеспечивать полные отчеты.

главный каталог. Каталог, содержащий полную информацию о наборах данных и томах, требуемую VSAM для поиска наборов данных, для распределения и освобождения пространства, для проверки полномочий программы или оператора, требующего доступа к набору данных, и для накопления статистики обращений по наборам данных.

глобальная зона. Группа записей в наборе данных CSI, используемая для записи информации о SYSMOD, полученных для определенной системы. Глобальная зона также содержит информацию, которая (1) позволяет SMP/E осуществлять доступ к целевым и дистрибутивным зонам в системе, (2) позволяет настраивать аспекты обработки SMP/E.

глобальная синхронизация ресурсов (global resource serialization). Функция, обеспечивающая в z/OS механизм синхронизации доступа к ресурсам (обычно – наборов данных) между несколькими образами z/OS.

граница полного слова. Участок памяти, адрес которого кратен 4.

григорианский календарь. Календарь, используемый с пятницы, 15 октября 1582 года в большей части мира.

группа. Набор пользователей RACF, которые могут разделять полномочия доступа к защищенным ресурсам.

группа вывода. Набор выходных данных задания, имеющих общие свойства, такие как класс, пункт назначения и внешнее записывающее устройство.

группа поколений данных (GDG). Набор исторически связанных наборов данных, отличных от VSAM, организованных в хронологическом порядке; каждый набор данных называется набором данных одного поколения.

группа с разделением данных DB2. Набор из одной или нескольких параллельных подсистем DB2, осуществляющих прямой доступ и изменение одних и тех же данных, обеспечивая их целостность.

группа совместного использования данных IMS DB. Набор из одной или нескольких параллельных подсистем IMS DB, осуществляющих прямой доступ и изменение одних и тех же данных, обеспечивая их целостность.

группа памяти. Набор томов и атрибутов хранения, определяемый администратором внешней памяти. Набор может представлять группу DASD-томов или томов на магнитной ленте или группу DASD-томов, оптических томов или томов на магнитной ленте, воспринимаемый как единая иерархия хранения объектов.

Д

дамп. Отчет, представляющий содержимое памяти. Дампы обычно создаются после отказов программ в целях диагностики.

данные записей. Наборы данных со структурой, ориентированной на записи, доступ к которым осуществляется по записям. Такая структура наборов данных стандартна для z/OS и других операционных систем для мэйнфреймов. См. также *поток байтов*.

двоичные данные. (1) Любые данные, не предназначенные для прямого прочтения человеком. Двоичные данные могут содержать непечатаемые символы, не относящиеся к диапазону текстовых символов. (2) Тип данных, использующий числовые значения, хранящиеся в битовых комбинациях из нулей и единиц. Двоичные данные позволяют разместить большое число в меньшем объеме памяти.

двойное слово. Последовательность битов или символов, составляющая восемь байт (два 4-байтовых слова) и обрабатываемая как одно целое.

двухбайтовый набор символов (DBCS). Набор символов, в котором каждый символ

представлен двухбайтовым кодом. Такие языки, как японский, китайский и корейский, содержащие больше символов, чем можно представить 256 кодовыми комбинациями, требуют двухбайтовых наборов символов. Так как каждый символ требует двух байтов, для ввода, отображения и печати DBCS-символов требуется оборудование и программы, поддерживающие DBCS. Сравните: *однобайтовый набор символов*.

демон. В системах UNIX: долгосрочный процесс, работающий автоматически и выполняющий непрерывные или периодические системные функции, например, управление сетью. Некоторые демоны запускаются автоматически для выполнения определенной задачи; другие демоны запускаются периодически. Примером является демон cron, периодически выполняющий задачи, перечисленные в файле crontab. В z/OS аналогом является запускаемая задача.

диалог конфигурации оборудования (HCD). В z/OS: панельная программа, входящая в средство Hardware Configuration Definition. Программа позволяет инсталляции определять устройства для системных конфигураций z/OS.

диалог. Логическое соединение между двумя программами через сеанс LU 6.2, позволяющий им осуществлять обмен данными друг с другом при обработке транзакции.

диалоговое окно. Интерактивное всплывающее окно, содержащее опции, позволяющие просматривать или изменять информацию, предпринимать особые действия, связанные с выбранными объектами, или осуществлять доступ к другим диалоговым окнам. Например, HCM содержит набор диалоговых окон, позволяющих создавать, редактировать, удалять и подключать объекты, а также управлять схемой конфигурации.

диалоговый. Относится к программе или системе, осуществляющей диалог с пользователем терминала, принимая входные данные и отвечая на входные данные достаточно быстро, чтобы соответствовать ходу мыслей пользователя.

динамическая реконфигурация. Способность вносить изменения в канальную подсистему и операционную систему во время работы системы.

динамически подключаемая библиотека (DLL). Файл, содержащий исполняемый код и данные, связываемые с программой во время загрузки или во время выполнения. Код и данные в динамически подключаемой библиотеке могут совместно использоваться несколькими приложениями одновременно.

динамическое распределение. Назначение системных ресурсов программе во время выполнения программы, а не во время ее загрузки в основную память.

директория. (1) Тип файла, содержащий имена и управляющую информацию о других файлах или других директориях. Директории могут также содержать поддиректории, которые могут содержать свои поддиректории. (2) Файл, содержащий записи директории. Две записи директории в одной директории не могут иметь одинаковое имя (POSIX.1). (3) Файл, указывающий на файлы и другие директории. (4) Индекс, используемый управляющей программой для поиска блоков данных, хранящихся в отдельных областях набора данных в хранилище с прямым доступом.

дистрибутивная библиотека (DLIB). Библиотека, содержащая главную копию всех элементов системы. Дистрибутивную библиотеку можно использовать для создания или резервного копирования целевой библиотеки.

дистрибутивная зона. В SMP/E: группа записей в наборе данных CSI, описывающая SYSMOD и элементы дистрибутивной библиотеки.

дополнительные материалы по установке (RIM). В самостоятельно разработанных предложениях IBM: документация по задачам, задания, примеры программ «выхода», процедуры, параметры и примеры, разработанные IBM.

доступ для чтения. Разрешение на чтение информации.

E

единица восстановления (UR). Набор изменений в одном узле, фиксируемых или отменяемых в составе ACID-транзакции. Единица восстановления неявно запускается при первом обращении менеджера ресурсов к защищенному ресурсу узла. Единица восстановления завершается по завершении процесса двухфазовой фиксации изменяющей ее ACID-транзакции.

единица компиляции. Фрагмент компьютерной программы, достаточно полный для корректной компиляции.

единый образ системы. Свойство, демонстрируемое продуктом, когда для нескольких образов продукта можно осуществлять отображение и управление как для одного образа.

единица обслуживания. Объем обслуживания, потребляемый рабочим запросом, опре-

деляемый по коэффициентам определения обслуживания, а также по единицам обслуживания CPU, SRB, ввода-вывода и памяти.

единый пункт управления. Свойство дисплеев сисплекса, состоящее в возможности выполнения определенного набора задач с одной рабочей станции, даже если для этого требуется использовать несколько продуктов IBM и сторонних изготовителей.

Ж

журнал операций. В z/OS журнал операций представляет собой основное место, куда делаются записи о коммуникациях и проблемах системы для каждой системы в сисплексе.

З

забронировать. Предоставление принтеру возможности завершить выполнение текущего задания перед остановкой устройства.

загрузка (fetch). Динамическая загрузка процедуры.

загрузочный модуль. Исполняемая программа, хранящаяся в библиотеке программ в секционированном наборе данных. См. также *программный объект*.

загрузчик программы. Программа, подготавливающая программы к выполнению путем их загрузки в определенные участки памяти и перенастройки каждой перемещаемой адресной константы.

задание. Единица работы в операционной системе. Задания определяются JCL-операторами.

запись контрольной точки. Любая запись в набор данных контрольной точки. Основной термин для первичных, промежуточных и итоговых операций записи, обновляющих любой набор данных контрольной точки.

задача. В среде мультипрограммирования или мультипроцессирования: одна или несколько последовательностей инструкций, воспринимаемых управляющей программой как элемент работы, подлежащий выполнению компьютером.

запись переменной длины. Запись, длина которой не зависит от длины других записей, с которыми она логически или физически связана. Сравните: *запись фиксированной длины*.

запись фиксированной длины. Запись, имеющая такую же длину, как и все другие записи, с которыми она логически или физически связана. Сравните: *запись переменной длины*.

запись. (1) Группа связанных данных, слов или полей, обрабатываемых как одно целое, например, запись, содержащая имя, адрес и телефон. (2) Самостоятельный набор информации об одном объекте. Запись содержит несколько отдельных элементов, называемых полями. Некоторые программы оболочки (например, awk, join и sort) предназначены для обработки данных, состоящих из записей, разделенных символами новой строки, где каждая запись содержит несколько полей, отделенных пробелами или каким-то другим символом. awk может также обрабатывать записи, разделенные символами, отличными от символов новой строки. См. *запись фиксированной длины*, *запись переменной длины*.

запуск. Действие, вызывающее выполнение программы, утилиты или другой машинной функции.

запускаемая задача. В z/OS: адресное пространство, запускаемое автоматически в результате команды START. Запускаемые задачи обычно используются для критических приложений. Является аналогом демона в UNIX.

И

идентификатор канального пути. Логический аналог каналов в физическом процессоре.

идентификатор пользователя (user ID). Имя длиной от 1 до 8 символов, идентифицирующее пользователя системы.

идентификатор физического канала (PCHID). Физический адрес канального пути в оборудовании. Логические CHPID имеют соответствующие физические каналы. Реальное оборудование ввода-вывода подключается к процессору через физические каналы. Каждый канал имеет идентификатор физического канала (PCHID), определяющий физическое расположение канала в процессоре. PCHID представляет собой трехзначное шестнадцатеричное число, назначаемое процессором.

идентификатор функциональных изменений (FMID). Код, идентифицирующий уровни версий лицензированной программы z/OS.

иерархическая файловая система (HFS). Набор данных, содержащий POSIX-совместимую файловую систему, представляющую собой набор файлов и директорий, организованных в иерархическую структуру, к которым можно осуществлять доступ через z/OS UNIX System Services.

избыточный массив независимых дисков (RAID). Архитектура дисковой подсисте-

мы, объединяющая два или более физических устройств дисковой памяти в единое логическое устройство для обеспечения избыточности данных.

имя входа. В ассемблере: имя, определяемое программистом в секции, идентифицирующее точку входа и допускающее обращение из любой управляющей секции. См. также *точка входа*.

имя определения данных (DD-имя). (1) Оператор имени определения данных (DD), соответствующий блоку управления данными, содержащему такое же имя. (2) Символическое представление имени, размещенное в поле имени оператора определения данных (DD).

имя определения данных. См. *DD-имя*.

имя точки входа. Символ (или имя), представляющее точку входа. См. также *точка входа*.

инициатор. Часть операционной системы, выполняющая считывание и обработку операторов языка управления заданиями, поступающих с устройства ввода системы.

инициатор/терминатор. Функция планировщика заданий, выбирающая задания и шаги заданий для выполнения, распределяющая для них устройства ввода-вывода, передающая их системе управления задачами и по завершении задания передающая управляющую информацию для записи выходных данных задания в устройство вывода системы.

интерактивный (interactive). Относится к программе или системе, которая поочередно принимает ввод данных и реагирует на него. В интерактивной системе, между пользователем и системой поддерживается постоянный диалог. Сравните: *пакет*.

интерфейс малых компьютерных систем (SCSI). Стандартный аппаратный интерфейс, позволяющий различным периферийным устройствам взаимодействовать друг с другом.

интерфейс подсистем (SSI). Компонент, обеспечивающий связь между операционной системой z/OS и ее подсистемой ввода заданий.

исключительный SYSMOD. SYSMOD, содержащий ошибку или требующий специальной обработки перед установкой. Исключительные SYSMOD определяются операторами ++HOLD и ++RELEASE.

исполняемая программа. (1) Программа в форме, подходящей для выполнения на компьютере. Программа может представлять собой приложение или скрипт оболочки. (2) Программа, которая была скомпонована и может выполняться процессором. (3) Программа, которая может выполняться как самостоятельная процедура. Она состоит из главной программы и, возможно, одной или нескольких подпрограмм. (4) См. также *исполняемый файл*, *загрузочный модуль*.

исполняемый модуль. Загрузочный модуль или программный объект, который должен быть загружен в память для выполнения.

исправление. Исправление ошибки в программе; обычно представляет собой временное исправление или обход ошибочного кода.

исходная программа. Набор инструкций, написанных на языке программирования, которые необходимо транслировать в машинный язык, прежде чем можно будет запустить программу.

исходный код. Входные данные компилятора или ассемблера, написанные на исходном языке.

К

кабель «в наличии». Неиспользуемые кабели.

канал coupling facility. Оптоволоконный канал с высокой пропускной способностью, обеспечивающий высокоскоростную связь, необходимую для обмена данными между устройством сопряжения и центральными процессорными комплексами, напрямую подключенными к нему.

канал-канал (CTC). Связь (передача данных) между программами, находящимися по разные стороны от CTC-адаптера (CTCA).

канальная подсистема (CSS). Набор подканалов, направляющих поток информации между устройствами ввода-вывода и основной памятью. Логические разделы используют подканалы для связи с устройствами ввода-вывода. Максимальное количество CSS, поддерживаемое процессором, зависит также от типа процессора. Если процессор поддерживает более одной CSS, тогда для каждой CSS используется шестнадцатеричный идентификатор CSS (CSS ID), уникальный для данного процессора.

канальный интерфейс. Схемы в контроллере памяти, подключающие пути доступа к памяти и каналу основной системы.

каталог. (1) Директория файлов и библиотек со ссылкой на их расположение. (2) Каталогизация – ввод информации о файле или библиотеке в каталог. (3) Набор всех индексов для всех наборов данных, используемых управляющей программой для поиска тома, содержащего определенный набор данных.

каталогизированная процедура. Набор операторов JCL, помещенных в библиотеку и извлекаемых по имени.

каталогизированный набор данных. Набор данных, представленный в индексе или иерархии индексов, обеспечивающих средства его поиска.

квалификатор. Модификатор в составном имени, отличный от последнего компонента имени. Например, «TREE» и «FRUIT» являются квалификаторами в имени «TREE.FRUIT.APPLE».

кеш. Электронная память с произвольным доступом в выделенных средствах управления памятью, служащая для хранения часто используемых данных в целях более быстрого доступа из канала.

кеш-структура. Структура устройства сопряжения, позволяющая осуществлять высокопроизводительный совместный доступ к кешируемым данным из многосистемных приложений в сисплексе. Приложения могут использовать кеш-структуру для реализации различных типов систем кеширования, включая сквозное кеширование (store-through) и кеширование с обратным копированием (store-in).

класс данных. Набор атрибутов распределения и пространства, определяемых администратором памяти, используемых при распределении нового набора данных, управляемого SMS.

класс задания. Любая из категорий заданий, которая может быть определена. Классификация заданий и использование инициаторов/терминаторов для инициирования определенных классов заданий позволяют управлять набором параллельно выполняющихся заданий.

класс памяти. Набор атрибутов хранения, идентифицирующий цели производительности и требования доступности, определяемый администратором внешней памяти и используемый для выбора устройства, которое может соответствовать этим целям и требованиям.

кластер ввода-вывода. Сисплекс, владеющий управляемым канальным путем для процессорной конфигурации с логическими разделами.

клиент. Функциональный модуль, получающий разделяемое обслуживание сервером. См. также *клиент-сервер*.

клиент-сервер. В TCP/IP: модель взаимодействия при распределенной обработке данных, при которой программа, запущенная на одном узле, отправляет запрос программе, запущенной на другом узле, и ожидает ответа. Запрашивающая программа называется клиентом; отвечающая программа называется сервером.

ключевое слово. Часть операнда команды, содержащая определенную символьную строку (например, DSNAME=).

код возврата. Код, генерируемый программой для обозначения ее успешного или неуспешного выполнения. Может использоваться для воздействия на выполнение последующих инструкций или программ.

код компоненты. Четырехзначный код, используемый IBM для обработки заказов аппаратных и программных средств.

код перенаправления. Код, назначаемый сообщению оператора и используемый для доставки сообщения на требуемую консоль.

код причины. Код завершения, описывающий причину отказа или неполного успеха выполнявшейся операции.

код серьезности. Часть сообщений для операторов, указывающая серьезность ошибки (I, E или S).

код ситуации. Код, отражающий результат предыдущей операции ввода-вывода, арифметической или логической операции.

кодовая комбинация. Однобайтовый код, представляющий один из 256 возможных символов.

кодовая страница. (1) Назначение графических символов и значений контрольных функций всем кодовым комбинациям; например, назначение символов и значений 256 кодовым комбинациям в случае 8-разрядного кода, назначение символов и значений 128 кодовым комбинациям в случае 7-разрядного кода. (2) Определенное назначение шестнадцатеричных идентификаторов графическим символам.

команда. Запрос на выполнение операции или запуск программы. Если с командой заданы параметры, аргументы, флаги или другие операнды, результирующая текстовая строка представляет одну команду.

команды оператора. Команды, которые операторы системы могут использовать для получения информации, изменения операций, инициации новых операций или завершения операций.

коммутатор. Устройство, обеспечивающее возможности связи и управление соединением двух ESCON- или FICON-подключений.

компилятор. Программа, транслирующая исходную программу в исполняемую программу (объектный модуль).

комплекс глобальной синхронизации ресурсов. Одна или несколько систем z/OS,

использующих глобальную синхронизацию ресурсов для синхронизации доступа к разделяемым ресурсам (например, к наборам данных на разделяемых DASD-томах).

комплементарный металл-оксид-полупроводник (CMOS). Технология, совмещающая электрические свойства положительных и отрицательных потенциалов, что позволяет значительно уменьшить потребление электроэнергии в сравнении с другими типами полупроводников.

компонент. Функциональная часть операционной системы; например, планировщик или супервизор.

компонента. Часть продукта IBM, которую клиент может заказать отдельно.

компоновка. (1) Объединение одной или нескольких управляющих секций разделов или программных модулей в один программный модуль с разрешением связей между ними. (2) В SNA: запрос активации сеанса между двумя логическими устройствами (LUs).

компоновщик. Программа z/OS, обрабатывающая выходные данные языковых трансляторов и компилирующая их в исполняемую программу (загрузочный модуль или программный объект). Она заменяет редактор связей и пакетный загрузчик, использовавшиеся в более ранних вариантах операционной системы z/OS, таких как MVS и OS/390.

консоль EMCS. В z/OS: консоль, отличная от консоли MCS, в которой операторы или программы могут вводить системные команды и получать сообщения. Консоль EMCS определяется в сегменте OPERPARM.

консоль MCS. Не относящееся к SNA устройство, определенное в z/OS, локально подключенное к системе z/OS и используемое для ввода команд и получения сообщений.

консоль вывода состояния. В z/OS: консоль MCS, в которой можно управлять функциями ввода-вывода.

консоль управления аппаратными средствами (HMC). Консоль, используемая для мониторинга и управления оборудованием, например, микропроцессорами мэйнфрейма.

консоль. Любое устройство, с которого операторы могут вводить команды или получать сообщения.

консольная группа. В z/OS: группа консолей, определенная в CNGRPxx, каждая из которых может использоваться в качестве альтернативной консоли при восстановлении с консоли или с твердой копии или в качестве консоли для вывода синхронных сообщений.

контроллер. Устройство, транслирующее высокоуровневые запросы от процессоров в низкоуровневые запросы для устройств ввода-вывода, и наоборот. Каждый физический контроллер содержит одно или несколько логических устройств управления, канальных интерфейсов и интерфейсов устройств и блок питания. Контроллеры можно разделить на сегменты или сгруппировать в подсистемы.

контрольная точка. (1) Место программы, в котором выполняется проверка или запись данных для перезапуска. (2) Точка, в которой можно записать информацию о состоянии задания и системы, чтобы этот шаг задания можно было перезапустить позже.

конфигурация Multi-Access Spool. Несколько систем, совместно использующих очереди входных данных, заданий и выходных данных JES2 (посредством набора данных контрольной точки или устройства сопряжения).

конфигурация. Компоновка компьютерной системы или сети в соответствии с сущностью, количеством и главными свойствами ее функциональных модулей.

корректирующее средство. Любая системная модификация (SYSMOD), используемая для избирательного исправления проблемы в системе. В целом этот термин относится к APAR-исправлениям.

криптографический ключ. Параметр, определяющий криптографические преобразования между открытым текстом и зашифрованным текстом.

криптография. Преобразование данных, предназначенное для того, чтобы скрыть их смысл.

Л

лента кумулятивного обслуживания. Магнитная лента, поставляемая с новой функцией, содержащая все текущие PTF для этой функции.

ленточный том. Пространство памяти на ленте, обозначенное меткой тома, содержащее наборы данных или объекты и доступное свободное пространство. Ленточный том представляет пространство записи на одном картридже или бобине с лентой. См. также *том*.

лицензированная программа. Программный пакет, который можно заказывать из программных библиотек, например, IBM Soft-

ware Distribution (ISMD). Примерами лицензированных программ являются IMS и CICS.

лицензированный внутренний код (LIC). Микрокод, который компания IBM не продает как часть машины, а лицензирует для клиента. LIC реализован как часть памяти, не адресуемой пользовательскими программами. Некоторые продукты IBM используют его для реализации функций в качестве альтернативы жестко закодированным схемам.

логическая подсистема. Логические функции контроллера внешней памяти, позволяющие одному или нескольким базовым интерфейсам ввода-вывода осуществлять доступ к набору устройств. Контроллер объединяет устройства в соответствии с механизмами адресации соответствующих интерфейсов ввода-вывода. Контроллер внешней памяти содержит одну или несколько логических подсистем. В целом, контроллер связывает определенный набор устройств только с одной логической подсистемой.

логический раздел (LP). Подмножество процессорного оборудования, определенное для поддержки операционной системы. См. также *режим логического раздела (LPAR)*.

логическое устройство (LU). В SNA: порт, через который конечный пользователь осуществляет доступ к сети SNA для связи с другим конечным пользователем и через который конечный пользователь осуществляет доступ к функциям, предоставляемым пунктами управления системными службами (SSCP).

логическое устройство типа 6.2. Тип логического устройства SNA, поддерживающий связь между программами в среде коллективной обработки данных.

логическое устройство управления (LCU). Одиночное устройство управления (CU) с подключенными устройствами или без них, или группа из одного или нескольких CU, совместно использующих устройства. В канальной подсистеме (CSS), LCU представляет набор CU, физически или логически подключающих устройства вводавывода.

локальная область системных очередей (LSQA). В z/OS: один или несколько сегментов, связанных с каждым регионом виртуальной памяти, содержащих блоки управления системой, связанные с заданием.

локальная сеть (LAN). Сеть, в которой связь ограничена географической областью среднего размера (от 1 до 10 км), например, одним офисным зданием, складом или университетским городком, и которая обычно не распространяется на территории других организаций. Локальная сеть использует коммуникационную среду, способную передавать данные со средней или высокой скоростью (выше 1 Мбит/с) и обычно работающую с низким показателем ошибок.

M

магистральный кабель. Кабель, используемый для создания постоянных подключений между шкафами, и остающийся в подключенном состоянии, даже когда он не используется.

макрос. Инструкция исходного языка, заменяемая определенной последовательностью инструкций того же исходного языка.

маркер команд и ответов (CART). Параметр WTO, WTOR, MGCRE и некоторых команд TSO/E, и исполняемых программ REXX, позволяющий сопоставлять команды и связанные с ними ответы-сообщения.

маршрутизация. Назначение пути связи, при использовании которого сообщение достигает адресата.

матричный коммутатор. Статический коммутатор, который может подключать контроллеры к процессорам с параллельными интерфейсами (интерфейсами шины и тега). Матричный коммутатор содержит несколько канальных интерфейсов, которые могут подключаться к объектам, расположенным выше по иерархии, в частности, к процессорам и другим матричным коммутаторам. Матричный коммутатор также со своей стороны содержит множество интерфейсов устройств управления, которые могут подключаться к объектам, расположенным ниже по иерархии, в частности, к контроллерам и другим матричным коммутаторам.

машиночитаемый. Относится к данным, которая машина может получить или интерпретировать (считать) с устройства хранения, носителя данных или из другого источника.

МБ. Мегабайт.

мегабайт (МБ). 2^{20} байт, 1 048 576 байт.

межпространственная связь. Метод вызова программы, расположенной в другом адресном пространстве. Вызов является синхронным относительно вызывающей стороны.

межсистемное средство сопряжения (XCF). Компонент z/OS, выполняющий функции поддержки взаимодействия между авторизованными программами, запущенными в сисплексе.

межсистемные расширенные службы (XES). Набор служб z/OS, позволяющих нескольким экземплярам приложения или под-

системы, запущенных на других системах в среде сисплекса, осуществлять высокоскоростной, высокодоступный обмен данными с использованием устройства сопряжения.

межсистемный перезапуск. При отказе системы менеджер автоматического перезапуска перезапускает элементы на другой подходящей системе в сисплексе.

менеджер точки синхронизации. Функция, координирующая процесс двухфазовой фиксации для защищенных ресурсов таким образом, что все изменения в данных либо фиксируются, либо отменяются. В z/OS в качестве менеджера точки синхронизации на уровне системы может выступать RRS. Менеджер точки синхронизации также называется менеджером транзакций, координатором точки синхронизации или координатором фиксации.

метка набора данных. (1) Набор информации, описывающей атрибуты набора данных и обычно хранящийся на одном томе с набором данных. (2) Общий термин для блоков управления набором данных и меток наборов данных на магнитной ленте.

метод доступа. Способ перемещения данных между основной памятью и устройствами ввода-вывода.

миграция. Действия, часто выполняемые системным программистом, связанные с установкой новой версии программы для замены предыдущей версии. Выполнение этих действий обеспечивает корректное функционирование приложений и ресурсов системы с новой версией.

микрокод. Хранимые микроинструкции, недоступные пользователям, выполняющие определенные функции.

микропроцессор. Процессор, реализованный в одной или небольшом количестве микросхем.

многозадачность. Режим работы, обеспечивающий одновременное или попеременное выполнение двух или больше задач или потоков. Синоним: *многопоточность*.

многократное использование. Атрибут модуля или раздела, определяющий степень его использования или разделения между несколькими задачами в адресном пространстве. См. *обновляемость*, *реентерабельность* и *последовательное многократное использование*.

многосистемная поддержка консолей. Поддержка нескольких консолей в нескольких системах сисплекса. Многосистемная поддержка консолей позволяет консолям в разных системах сисплекса обмениваться данными друг с другом (отправлять сообщения и принимать команды).

многосистемная среда. Среда, в которой два или больше образов z/OS размещаются на одном или нескольких процессорах, и программы одного образа могут обмениваться данными с программами других образов.

многосистемное приложение. Приложение, имеющее различные функции, распределенные по образам z/OS в многосистемной среде.

многосистемный сисплекс. Сисплекс, в котором два или больше образов z/OS можно инициализировать в составе сисплекса.

модуль. Объект, представляющий собой результат компиляции исходного кода. Модуль нельзя запускать. Для запуска модуль необходимо скомпоновать в программу.

моноплекс. Система, содержащая одну систему, использующую набор данных сопряжения сисплекса.

мультипроцессирование. Одновременное выполнение двух и больше компьютерных программ или последовательностей инструкций. См. также *параллельная обработка*.

мультипроцессор (MP). CPC, который можно физически разделить для создания двух операционных процессорных комплексов.

Н

набор данных HFS. Набор данных, содержащий POSIX-совместимую файловую систему, представляющую собой набор файлов и директорий, организованных в иерархическую структуру, к которым можно осуществлять доступ через z/OS UNIX System Services.

набор данных в спуле. Набор данных, записанный на устройство вспомогательной памяти и управляемый JES.

набор данных контрольной точки. Набор данных, в котором можно записать информацию о состоянии задания и системы, чтобы позже можно было перезапустить шаг задания.

набор данных конфигурации ввода-вывода (IOCDS). Файл, содержащий различные определения конфигурации для определенного процессора. Одновременно используется только один IOCDS. IOCDS содержит данные конфигурации ввода-вывода для файлов, связанных с контроллером процессора на базовом процессоре, используемые канальной подсистемой. Канальная подсистема (CSS) использует данные конфигурации для управления запросами ввода-вывода. IOCDS строится из рабочего IODF.

набор данных свопинга. Набор данных, выделенный для операции свопинга.

набор данных сопряжения сисплекса. Набор данных сопряжения, содержащий данные сисплекса о системах, группах и компонентах, использующих службы XCF. Все системы z/OS в сисплексе должны иметь связь с набором данных сопряжения сисплекса. См. также *набор данных сопряжения.*

набор данных сопряжения. Набор данных, созданный утилитой форматирования наборов данных XCF и, в зависимости от присвоенного ему типа, разделяемый некоторыми или всеми системами z/OS в сисплексе. См. также *набор данных сопряжения сисплекса.*

набор данных, упорядоченный по ключам (KSDS). Файл или набор данных VSAM, записи которого загружаются в порядке возрастания ключей и управляются индексом. Извлечение и сохранение записей осуществляется по ключам или по адресам; вставка новых записей в ключевую последовательность осуществляется посредством использования распределенного свободного пространства. Относительные адреса байтов могут изменяться из-за расщеплений управляющих интервалов или областей управления.

набор данных, управляемый системой. Набор данных, которому был назначен класс памяти.

набор данных. В z/OS: именованный набор связанных записей данных, хранящийся и извлекаемый с использованием назначенного имени. Является аналогом файла.

набор подканалов. Структура, заданная инсталляцией, определяющая размещение устройств относительно канальной подсистемы или операционной системы.

начало последовательности. Первый модуль в последовательности устройств. Содержит интерфейсы последовательности, соединяющиеся с интерфейсами контроллерного устройства.

начальная загрузка (IPL). Процедура инициализации, после которой операционная система z/OS начинает работу. Во время начальной загрузки системные программы загружаются в память, после чего z/OS готова к работе. Синонимы: *загрузка*, *инициализация.*

начальное распределение памяти. Объем основной и расширенной памяти, выделяемый логическому разделу.

неиспользуемый кабель. Физический кабель, уже отключенный, но еще не переведенный в запас.

неперемещаемый регион. В MVS: раздел неперемещаемой динамической области, выделенный для шага задания или системной задачи, для которого не выполняется вытеснение при выполнении. В неперемещаемом регионе каждый виртуальный адрес идентичен его реальному адресу. Синоним: *регион V=R.*

нереентерабельная программа. Тип программы, которая не может совместно использоваться несколькими пользователями.

нестандартные метки. Метки, не соответствующие стандарту American National Standard или соглашениям о стандартных метках IBM System/370.

нисходящая совместимость. Способность приложений работать на предыдущих версиях z/OS.

номер программы. Семизначный код (в формате xxxx-xxx), используемый IBM для идентификации каждой лицензированной программы.

номер устройства. Идентификатор, состоящий из четырех шестнадцатеричных знаков, например 13A0, связанный с устройством и предназначенный для того, чтобы облегчить связь между программой и оператором узла. Номер устройства, связанный с подканалом.

нуль (null). Пустой объект; не имеет значения.

О

область ввода. В z/OS: часть экрана консоли, в которой операторы могут вводить команды или ответы на команды.

область загрузки модулей (LPA). Область виртуальной памяти, содержащая реентерабельные программы, загружаемые при начальной загрузке (IPL) и одновременно используемые всеми задачами в системе.

область системных очередей (SQA). В z/OS: область виртуальной памяти, зарезервированная для блоков управления, связанных с системой.

область хранения регистров (RSA). Область основной памяти, в которой хранится содержимое регистров.

область хранения. Область основной памяти, в которой хранится содержимое регистров.

обновляемость (refreshable). Атрибут многократного использования, позволяющий заменить (обновить) программу новой копией без прерывания ее выполнения. Обновляемый модуль не может изменять себя или быть из-

менен другим модулем во время выполнения. См. *многократное использование*.

обработка RECEIVE. SMP/E-процесс, необходимый для установки новых библиотек продуктов. В ходе этого процесса, код, организованный в виде выгруженных секционированных наборов данных, загружается во временные наборы данных SMPTLIB. Обработка SMP/E RECEIVE автоматически распределяет временные секционированные наборы данных, соответствующие файлам на ленте, и загружает их с ленты.

образ. Одиночный экземпляр операционной системы z/OS.

обратный порядок байтов (big endian). Формат хранения двоичных данных, при котором старший байт располагается первым. Обратный порядок байтов используется в большинстве аппаратных архитектур, включая z/Architecture. Сравните: *прямой порядок байтов (little endian)*.

обслуживание. PTF- и APAR-исправления.

обход. В SMP/E: обход ошибок, которые иначе привели бы к отказу в обработке SYSMOD. Выполняется с использованием операнда BYPASS команд SMP/E.

общая сервисная область (CSA). В z/OS: часть общей области, содержащая области данных, адресуемые всеми адресными пространствами.

общая утилита трассировки (GTF). Подобно системной трассировке, собирает информацию, используемую для определения и диагностики проблем, возникающих во время работы системы. В отличие от системной трассировки, GTF можно настроить на запись определенных системных и пользовательских программных событий.

объектная колода. Набор из одной или нескольких управляющих секций, генерируемый ассемблером или компилятором, и используемый как входные данные редактора связей или компоновщика. Также называется объектным кодом (OBJ).

объектный модуль. Модуль, представляющий собой выходные данные языкового транслятора (например, компилятора или ассемблера). Объектный модуль имеет перемещаемый формат с неисполняемым машинным кодом. Прежде чем объектный модуль можно будет выполнять, его необходимо обработать утилитой редактирования связей.

оглавление тома (VTOC). Таблица на томе устройства хранения с прямым доступом (DASD), описывающая расположение, размер и другие свойства каждого набора данных на томе.

однопроцессорный (UP). Процессорный комплекс, содержащий один центральный процессор.

однопроцессорный комплекс. Вычислительная среда, в которой только один процессор (компьютер) осуществляет доступ к спулу и составляет целый узел.

одноранговое удаленное копирование (PPRC). Прямое подключение между подсистемами DASD-контроллера, используемое преимущественно для обеспечения горячего резерва. Эти подключения могут представлять собой соединения «точка-точка» между двумя DASD-контроллерами или могут связываться через коммутаторы, как и подключения между CHPID и устройствами управления.

односистемный сисплекс. Сисплекс, в котором только одной системе z/OS разрешена инициализация в составе сисплекса. В односистемном сисплексе, XCF выполняет в системе свои функции, но не обеспечивает функции сигнализации между системами z/OS. См. также *многосистемный сисплекс*.

оперативное задание (foreground job). (1) Задание с высоким приоритетом, обычно представляющее собой задание реального времени. (2) В TSO: любое задание, выполняющееся в свопируемом регионе основной памяти, например, командный процессор или программа пользователя терминала. Сравните: *фоновое задание*.

оперативный (online). Относится к способности пользователя взаимодействовать с компьютером**.**

оперативный режим (foreground). (1) В мультипрограммировании: среда, в которой выполняются программы с высоким приоритетом. (2) В TSO: среда, в которой выполняется свопинг программ в основную память и из основной памяти, позволяя осуществлять разделение процессорного времени между терминальными пользователями. Все программы командного процессора выполняются в оперативном режиме. Сравните: *фоновый режим*.

оператор определения данных (DD). Управляющий оператор задания, описывающий набор данных, связанный с определенным шагом задания.

оператор определения данных. Управляющий оператор JCL, выступающий в качестве связи между логическим именем файла (DD-именем) и физическим именем файла (именем набора данных).

оператор управления модификацией (MCS). Управляющий оператор SMP/E, используемый для упаковки SYSMOD. MCS описывают элементы программы и связи этой программы с другими программами, установленными в той же системе.

операторы JCL. Операторы, помещенные во входной поток для определения выполняемой работы, используемых методов и требуемых ресурсов.

операционная система. Программное обеспечение, управляющее выполнением программ; кроме того, операционная система может обеспечивать такие функции, как распределение ресурсов, планирование, управление вводом-выводом и управлением данными. Хотя операционные системы состоят преимущественно из программного обеспечения, возможны частичные аппаратные реализации.

оптоволоконный канал. Физические оптоволоконные подключения и передающая среда между оптоволоконными передатчиками и приемниками. Оптоволоконный канал может содержать один или несколько оптоволоконных кабелей и портов связи в шкафах управления оптоволоконными каналами. Каждое подключение в оптоволоконном канале является либо постоянным, либо переключаемым.

опции компилятора. Ключевые слова, которые можно задавать для управления некоторыми аспектами компиляции. Опции компилятора могут определять свойства загрузочного модуля, генерируемого компилятором, типы выходных данных, выводимых на печать, эффективное использование компилятора и объект вывода сообщений об ошибках. Также называются опциями времени компиляции.

опция совместного использования DASD. Опция, позволяющая независимым вычислительным системам совместно использовать общие данные, находящиеся на совместно используемых устройствах хранения с прямым доступом.

освобождение. Освобождение ресурса, выделенного определенной задаче.

основная память. (1) В z/OS: память вычислительной системы, из которой центральное процессорное устройство может напрямую извлекать инструкции и данные, и в которую оно может напрямую возвращать результаты (раньше использовалось название «реальная память»). (2) Синоним: *процессорная память.*

основная трасса. Средство централизованной трассировки данных главного планировщика, используемое при обслуживании компонентов z/OS, занимающихся обработкой сообщений.

откат. Процесс восстановления данных, измененных приложением, в состояние на момент их последней фиксации.

отложенный перезапуск. Перезапуск, выполняемый системой в тех случаях, когда пользователь повторно передает задание на выполнение. Оператор передает перезапускаемый модуль в систему через системное устройство считывания. См. также *перезапуск с контрольной точки*. Сравните: *автоматический перезапуск.*

отсутствие страницы. В системах виртуальной памяти z/OS или System/390: программное прерывание, возникающее при обращении к странице, помеченной как не находящейся в основной памяти, из активной страницы.

очередь сообщений. Очередь сообщений, ожидающих обработки или отправления на терминал.

очередь. Строка или список, состоящий из элементов системы, ожидающих обработки.

пакет. Группа записей или заданий обработки данных, собранных для обработки или передачи. Относится к операциям, требующим минимального вмешательства пользователя. Сравните: *интерактивный.*

П

пакетная обработка. Метод выполнения программы или набора программ, в которой одна или несколько записей (пакет) обрабатываются в минимальным вмешательством пользователя или оператора. Сравните: *интерактивная обработка.*

пакетное задание. Предопределенный набор операций обработки, переданных в систему для выполнения с минимальным взаимодействием между пользователем и системой. Сравните: *интерактивное задание.*

память подпула. Все блоки памяти, выделенные для определенной задачи под заданным номером подпула.

память, управляемая системой. Память, управляемая подсистемой Storage Management Subsystem (SMS) операционной системы z/OS.

параллельная обработка. Одновременная обработка единиц работы несколькими серверами. Единицами работы могут быть транзакции или подразделы больших единиц работы (пакетов). См. также *высокий параллелизм.*

параметр. Элемент данных, передаваемый программе.

пароль. Уникальная символьная строка, известная компьютерной системе и пользователю, который должен ввести эту символьную строку для получения доступа к системе и к хранящейся в ней информации.

первичный ключ. Один или несколько символов в записи данных, используемый для идентификации записи данных или для контроля ее использования. Первичный ключ должен быть уникальным.

передаваемые данные. Данные обновления на DASD-томах прикладной системы, отправляемые на систему восстановления для записи на DASD-тома системы восстановления.

перезапись. Запись поверх существующих данных в памяти.

перезапуск шага. Перезапуск, выполняющийся с начала шага задания. Перезапуск может быть автоматическим или отложенным, где отсрочка включает повторную передачу задания. Сравните: *перезапуск с контрольной точки*.

переключаемое соединение. Соединение с использованием оптоволоконных соединительных кабелей между портами связи в шкафу или между шкафами и активными объектами, такими как CHPID, коммутаторы, конвертеры и контроллеры с ESCON- или FICON-интерфейсами. Переключаемые соединения разрываются при прекращении использования соединяемых ими портов связи.

перемещаемый регион. В MVS: раздел перемещаемой динамической области, выделяемый для шага задания или системной задачи, который может вытесняться во время выполнения. Синоним: *регион V=V*.

переносимость. Возможность переноса приложения с одной платформы на другую с небольшим количеством изменений исходного кода.

ПК. персональный компьютер.

планирование профилактического обслуживания (PSP). Установочные рекомендации и HOLDDATA для продукта или уровня обслуживания. Информацию о PSP можно получить в Центре поддержки IBM.

платформа. Среда операционной системы, в которой выполняется программа.

по умолчанию. Используемое значение или действие, выполняемое в тех случаях, когда пользователь не задал альтернативное значение (действие).

поддержка PFK. В дисплейной консоли указывает на то, что функциональные клавиши программы поддерживаются и были заданы при генерации системы.

подзадача. В контексте многозадачности z/OS: задача, инициируемая и завершаемая задачей более высокого порядка (главной задачей). Подзадачи выполняют параллельные функции – те части программы, которые могут выполняться независимо от программы основной задачи и друг от друга.

подключение. В TCP/IP: путь между двумя приложениями протокола, обеспечивающий надежную передачу потока данных. В Интернете: подключение осуществляется между TCP-приложением на одной системе и TCP-приложением на другой системе.

подсистема. Вторичная или подчиненная система или система программной поддержки, обычно могущая работать независимо от управляющей системы или асинхронно с ней. Примеры: CICS и IMS.

подсистема ввода заданиями (JES). Системное средство спулинга, организации очереди заданий и управления вводом-выводом.

подстановочные символы. Использование звездочки (*) для обозначения нескольких символов в правилах классификации.

полное слово. Последовательность битов или символов, составляющая четыре байта (одно слово) и обрабатываемая как одно целое.

полномочие доступа. Полномочие, связанное с запросом типа доступа к защищенным ресурсам. В RACF используются следующие полномочия доступа: NONE, READ, UPDATE, ALTER и EXECUTE.

полномочие. Право доступа к объектам, ресурсам или функциям.

полномочия суперпользователя. Неограниченные возможности доступа и изменения любой части операционной системы, обычно связанные с пользователем, управляющим системой.

получение. В SMP/E: чтение SYSMOD и других данных из SMPPTFIN и SMPHOLD и их сохранение в глобальной зоне для последующей обработки в SMP/E. Выполняется командой RECEIVE.

пользовательская модификация (USERMOD). Изменение, разрабатываемое пользователем для модификации существующей функции, дополнения к существующей функции или добавления пользовательской функции. USERMOD идентифицируются в SMP/E оператором ++USERMOD.

пользовательский «выход». Программа, получающая управление в определенной точке приложения. Пользовательские «выходы» часто используются для обеспечения дополнительных функций инициализации и завершения.

пользовательский аварийный останов. Запрос аварийного завершения программы, выполняемый пользовательским кодом к операционной системе. Сравните: *системный аварийный останов*.

пользовательский каталог. Дополнительный каталог, на который указывает главный каталог, и который используется так же, как и главный каталог. Позволяет уменьшить конкуренцию за доступ к главному каталогу и облегчить переносимость тома.

порт связи. Пара оптоволоконных адаптеров или разветвителей. Оптоволоконный канал может содержать любое количество портов связи. Для определения общего количества портов связи в шкафу необходимо сложить количество портов связи каждой определенной панели шкафа.

порядок байтов (endian). Атрибут представления данных, определяющий способ хранения некоторых многобайтовых данных в памяти. См. *прямой порядок байтов (little endian)* и *обратный порядок байтов (big endian)*.

последовательное многократное использование (serially reusable). Атрибут многократного использования, позволяющий запускать программу несколькими задачами последовательно. Новая задача не может использовать модуль последовательного многократного использования до тех пор, пока его не освободит предыдущая задача. См. *многократное использование*.

последовательность. Набор из одного или нескольких устройств ввода-вывода. Этот термин обычно относится к физической последовательности устройств, но может означать и набор устройств ввода-вывода, интегрированных в устройство управления.

последовательный метод доступа с очередями (QSAM). Расширенная версия базисного последовательного метода доступа (BSAM). Блоки входных данных, ожидающие обработки, или блоки выходных данных, ожидающие передачи во вспомогательную память, организуются в очереди, чтобы сократить задержки операций ввода-вывода.

последовательный набор данных. (1) Набор данных, записи которого организованы в виде последовательного физического расположения, например, на магнитной ленте. Сравните: *набор данных с прямым доступом*. (2) Набор данных, в котором содержимое организовано в последовательном физическом порядке и хранится как одно целое. Набор данных может содержать данные, текст, программу или часть программы. Сравните: *секционированный набор данных (PDS)*.

поставщик. Человек или компания, предоставляющая обслуживание или продукт другому человеку или компании.

постоянное подключение. Постоянные подключения обычно создаются между шкафами с использованием оптоволоконных магистральных кабелей. Порты связи постоянно подключены, даже если они не используются.

постоянный набор данных. Набор данных, именованный пользователем, обычно сохраняемый дольше продолжительности задания или интерактивного сеанса. Сравните: *временный набор данных*.

поток байтов. Простая последовательность байтов, хранящихся в поточном файле. См. также *данные записей*.

поток данных. (1) Вся информация (данные и управляющие команды), передаваемая через канал данных, обычно одной операцией чтения или записи. (2) Непрерывный поток элементов данных, передаваемых или предназначенных для передачи, в символьной или двоичной форме с использованием определенного формата.

предприятие. Набор всех операционных сущностей, функций и ресурсов, составляющих коммерческую организацию.

препроцессор. Программа, просматривающая исходный код приложения на наличие операторов препроцессора, которые затем ею исполняются, вызывая изменение исходного кода.

прерывание. Приостановка процесса, например, выполнения компьютерной программы, вызванная событием, внешним по отношению к этому процессу, и выполняемая таким образом, чтобы процесс можно было продолжить.

прерывание от схем машинного контроля. Прерывание, возникающее в результате сбоя оборудования или ошибки.

префикс команды. Идентификатор команды длиной от одного до восьми символов. Префикс команды определяет команду как относящуюся к приложению или подсистеме, а не к z/OS.

прикладная программа. Набор программных компонентов, используемых для выполнения определенных типов задач на компьютере, например программы управления инвентаризацией или ведением платежных ведомостей.

приложение. Программа или набор программ, выполняющий задачу; например, приложения ведения платежных ведомостей, управления инвентаризацией и текстовой обработки.

применение. В SMP/E: установка SYSMOD в целевых библиотеках. Выполняется командой APPLY.

принтер. Устройство, записывающее выходные данные с системы на бумагу или на другой носитель.

принятие. В SMP/E: установка SYSMOD в дистрибутивных библиотеках. Выполняется командой ACCEPT.

принятый SYSMOD. SYSMOD, успешно установленный SMP/E-командой ACCEPT. В принятых SYSMOD не установлен флаг ERROR, и они представлены записями SYSMOD в дистрибутивной зоне.

приоритет задания. Значение, присваемое заданию, используемое как мера относительной важности задания при соревновании задания с другими заданиями за системные ресурсы.

проверка авторизации. Действие, состоящее в определении того, разрешен ли пользователю доступ к RACF-защищенному ресурсу.

проверка глобального доступа. Способность создания инсталляцией в памяти таблицы заданных по умолчанию значений уровня авторизации для определенных ресурсов.

Программа Device Support Facilities (ICKDSF). Программа, используемая для инициализации DASD-томов при установке и обслуживании носителей.

программа «выхода» инсталляции. Средство, позволяющее системным программистам клиента модифицировать программный продукт IBM, изменяя или расширяя функции продукта.

программа инициализации ядра (NIP). Компонент z/OS, инициализирующий управляющую программу; позволяет оператору запрашивать последние изменения в некоторых опциях, задаваемых при инициализации.

программа пакетной обработки сообщений (BMP). IMS-программа пакетной обработки, имеющая доступ к оперативным базам данных и очередям сообщений. BMP работают в оперативном режиме, однако, подобно программам в пакетной среде, они запускаются с использованием JCL.

программа сортировки/объединения. Программа обработки, которую можно использовать для сортировки или объединения записей в предопределенной последовательности.

программная библиотека. Секционированный набор данных или PDSE, который всгда содержит именованные разделы.

программная маска. Четырехбитная структура в битах 20–23 слова состояния программы (PSW), определяющая, какие исключения должны вызывать программное прерывание: переполнение с фиксированной точкой, десятичное переполнение, переполнение разрядов порядка и исключение потери значимости. Возможно управление битами программной маски таким образом, чтобы включить или отключить возникновение программного прерывания.

программная функциональная клавиша (PFK). Клавиша на клавиатуре дисплейного устройства, передающая в программу сигнал вызова определенной программной операции.

программное обеспечение. (1) Все программы, процедуры, правила и соответствующая документация системы обработки данных или их часть. (2) Набор программ, процедур и, возможно, соответствующей документации, связанный с работой системы обработки данных. Например, компиляторы, библиотечные программы, руководства, схемы соединений. Сравните: *аппаратное обеспечение.*

программное прерывание. Прерывание выполнения программы в связи с возникновением некоторого события, например, исключения операции, исключения переполнения разрядов порядка или исключения адресации.

программный модуль. Выходные данные компоновщика. Собирательное понятие для программного объекта и загрузочного модуля.

программный объект. Компьютерная программа или ее часть в форме, подходящей для загрузки в виртуальную память для выполнения. Программные объекты хранятся в программных библиотеках PDSE и имеют меньше ограничений, чем загрузочные модули. Программные объекты генерируются компоновщиком.

проникновение. Действие, предпринимаемое менеджером ситуаций, когда значение, возвращаемое обработчиком ситуаций, ука-

зывает, что обработчик не смог обработать ситуацию, и что ситуация будет передана следующему обработчику.

профилактическое обслуживание. (1) Массовая установка PTF в целях недопущения повторного обнаружения проблем, исправленных этими PTF. (2) SYSMOD, поставляемые на магнитной ленте с программным обновлением.

профиль. Данные, описывающие важные свойства пользователя, группы пользователей либо одного или нескольких компьютерных ресурсов.

процедура. Набор самостоятельных операторов высокоуровневого языка (HLL), выполняющий определенную задачу и возвращающий управление вызывающему блоку. В разных языках используются различные названия для понятия процедуры. В C процедура называется функцией. В COBOL процедура называется параграфом или разделом, которые могут выполняться только из программы. В PL/I процедура представляет собой именованный блок кода, который можно инициировать извне, обычно посредством вызова.

процедура (routine). (1) Программа или последовательность инструкций, вызываемая программой. Обычно программа имеет общее назначение и часто используется. Процедуры используются в CICS и языках программирования. (2) Объект базы данных, инкапсулирующий процедурную логику и SQL-операторы; хранится на сервере баз данных и может быть вызван из SQL-оператора или оператором CALL. Тремя основными классами процедур являются собственно процедуры, функции и методы. (3) В REXX: набор инструкций, вызываемых инструкцией CALL или как функция. С точки зрения программы пользователя, процедура может быть внутренней или внешней. (4) Набор операторов в программе, при использовании которого система выполняет операцию или набор связанных операций.

процессор. Физический процессор или машина; имеет серийный номер, набор каналов и связанный с ним логический процессор. Логический процессор содержит несколько идентификаторов канальных путей (CHPID), являющихся логическим аналогом каналов. Логический процессор может быть разделен на несколько логических разделов.

процессорная память. См. *основная память.*

процессорный контроллер. Аппаратное средство, обеспечивающее функции поддержки и диагностики для центральных процессоров.

прямой порядок байтов (little endian). Формат хранения двоичных данных, при котором младший байт располагается первым. Прямой порядок байтов используется в аппаратных архитектурах Intel. Сравните: *обратный порядок байтов (big endian).*

P

рабочая нагрузка. Набор работ, для которого отслеживание, управление и создание отчетов осуществляется как для одного целого.

рабочий запрос. Элемент работы, например, запрос обслуживания, пакетное задание, APPC-, CICS- или IMS-транзакция, TSO LOGON или команда TSO.

раздел библиотеки параметров. Один из разделов PDS SYS1.PARMLIB, содержащий параметры, ограничивающие и контролирующие работу z/OS.

раздел данных. В COBOL: часть программы, описывающая файлы, используемые в программе, и записи, находящиеся в файлах. Здесь также описываются необходимые элементы данных разделов WORKING-STORAGE, LINKAGE SECTION и LOCAL-STORAGE.

раздел. Секция секционированного набора данных (PDS) или расширенного секционированного набора данных (PDSE).

разделение данных. Способность параллельных подсистем (таких как DB2 или IMS DB) или приложений осуществлять прямой доступ и изменение одних и тех же данных, обеспечивая их целостность.

разделяемая память. Область памяти, одинаковая для всех виртуальных адресных пространств. Так как разделяемая память представляет собой одинаковое пространство для всех пользователей, хранящаяся в ней информация может использоваться совместно, и ее необязательно загружать в пользовательский регион.

разделяемый CPC. CPC, который можно разделить на два отдельных CPC. См. также *физический раздел*, *режим одного образа*, *сторона.*

размер блока. (1) Количество элементов данных в блоке. (2) Показатель размера блока, обычно задаваемый в таких единицах, как записи, слова, компьютерные слова или символы. (3) Синоним: *длина блока.* (4) Синоним: *размер физической записи.*

распределение. Назначение ресурса для использования при выполнении определенной задачи.

распределенные вычисления. Вычисления, включающие взаимодействие двух или

больше машин через сеть. Данные и ресурсы совместно используются отдельными компьютерами.

распределенные данные. Данные, находящиеся в СУБД не в локальной системе.

расширенное удаленное копирование (XRC). Аппаратно-программная служба удаленного копирования, осуществляющая асинхронное копирование томов в подсистемах хранения для аварийного восстановления, миграции устройств и миграции нагрузки.

расширенный секционированный набор данных (PDSE). Управляемый системой набор данных, содержащий индексированное оглавление и разделы, подобные оглавлению и разделам секционированных наборов данных. PDSE можно использовать вместо секционированного набора данных.

реальный адрес. В системах виртуальной памяти: адрес участка основной памяти.

регион V=R. Синоним: *неперемещаемый регион*.

регион V=V. Синоним: *перемещаемый регион*.

регион-владелец приложения (AOR). В конфигурации CICSPlex®: CICS-регион, выделенный для выполнения приложений.

регион-владелец терминала (TOR). CICS-регион, выделенный для управления сетью терминалов.

регистр. Внутренний компонент компьютера, способный хранить определенное количество данных, мгновенно принимая и передавая эти данные.

редактирование связей. Создание загружаемой компьютерной программы средствами редактора связей или компоновщика.

редактор связей. Компонент операционной системы, разрешающий перекрестные ссылки между отдельно скомпилированными или ассемблированными модулями и присваивающий итоговые адреса для создания единого перемещаемого загрузочного модуля. Затем редактор связей сохраняет загрузочный модуль в загрузочной библиотеке на диске.

реентерабельность (reenterable). Атрибут многократного использования, позволяющий осуществлять одновременное использование программы несколькими задачами. Реентерабельный модуль может изменять собственные данные или другие общие ресурсы при обеспечении соответствующей синхронизации, препятствующей возникновению помех между задачами. См. *многократное использование*, *реентерабельный*.

реентерабельный (reentrant). Атрибут программы или приложения, позволяющий нескольким пользователям совместно использовать одну копию загрузочного модуля.

режим адресации (AMODE). Атрибут программы, описывающий длину адреса, использование которой ожидается при вызове программы. В z/OS адреса могут иметь длину 24, 31 или 64 бита.

режим логического разделения (LPAR). Режим сброса по питанию (power-on reset) центрального процессорного комплекса (CPC), позволяющий использовать средство PR/SM, а также позволяющий оператору распределять аппаратные ресурсы CPC (включая центральные процессоры, основную память, расширенную память и канальные пути) между логическими разделами.

режим прокрутки. Режим вывода на консоли, при котором строка разделителя между старыми и новыми сообщениями перемещается вниз при добавлении новых сообщений. Когда экран заполнен, тогда при добавлении новых сообщений строка разделителя замещает самое старое сообщение, и новое сообщение выводится непосредственно перед строкой разделителя.

режим размещения (RMODE). Атрибут программного модуля, определяющий, должен ли модуль после загрузки находиться ниже 16-мегабайтной «линии» виртуальной памяти, или же он может находиться в любом участке виртуальной памяти.

режим с одним образом (SI). Режим работы мультипроцессора (MP), позволяющий ему функционировать как один CPC. Однопроцессорная система по определению работает в режиме с одним образом. Сравните: *режим физического разделения (PP)*.

режим сквозной передачи 3270. Режим, при котором программа, выполняющаяся в оболочке z/OS, может отправлять и принимать поток данных 3270 или выдавать команды TSO/E.

режим физического разделения (PP). Состояние процессорного комплекса, при котором его аппаратные модули разделены между двумя отдельными операционными конфигурациями или сторонами. Сторона A контроллера процессора контролирует сторону 0; Сторона B контроллера процессора контролирует сторону 1. Сравните: *конфигурация с одним образом (SI)*.

резервное копирование набора данных. Резервное копирование с целью защиты от потери отдельных наборов данных.

резервное копирование тома. Резервное копирование целого тома с целью защиты от потери данных тома.

резервное копирование. Процесс создания копии набора данных, чтобы предотвратить его неумышленную потерю.

рекурсивная программа. Программа, которая может вызывать себя или быть вызванной из другой программы, которую она вызвала до этого.

ресинхронизация. Копирование образов дорожек с первичного тома на вторичный том, только тех дорожек, которые изменились со времени последнего пребывания тома в дуплексном режиме.

родовое имя. Определенная в z/OS группа устройств с похожими свойствами. Например, типы устройств 3270-X, 3277-2, 3278-2, -2A, -3, -4 и 3279-2a, -2b, -2c, -3a, -3b относятся к одному роду. Каждый род имеет родовое имя, используемое для распределения устройств в операторе JCL DD. z/OS воспринимает это имя как указание «взять любое устройство из этой группы». В любой конфигурации z/OS каждая таблица подходящих устройств (EDT) имеет одинаковый набор родовых имен.

C

свойства ACID. Свойства транзакции: атомарность, согласованность, изоляция, долговечность (Atomicity, Consistency, Isolation, Durability). В CICS, свойства относятся к единицам работы (unit of work, UoW).

свопинг. Операция страничного обмена в z/OS, записывающая активные страницы задания во вспомогательную память и считывающая страницы другого задания из вспомогательной памяти в основную память.

связанный список. Список, в котором элементы данных могут быть рассредоточены, но при этом каждый элемент данных содержит информацию, необходимую для нахождения следующего элемента. Синоним: *цепной список*.

сеанс. (1) Период времени, в течение которого пользователь терминала может взаимодействовать с интерактивной системой; обычно представляет собой время от подключения терминала к системе до отключения терминала от системы. (2) Период времени, в течение которого программы или устройства могут взаимодействовать друг с другом. (3) В VTAM: период времени, в течение которого узел подключен к приложению.

секционированный набор данных (PDS). Набор данных на устройстве с прямым доступом, разделенный на секции, называемые разделами, каждый из которых может содержать программу, часть программы или данные. Синоним: *программная библиотека*. Сравните: *последовательный набор данных*.

сервер. (1) В сети: компьютер, содержащий программы и данные, или обеспечивающий средства, доступные для других компьютеров сети. (2) Сторона, принимающая удаленные вызовы процедур. Сравните: *клиент*.

серверное адресное пространство. Любое адресное пространство, работающее от имени менеджера транзакций или менеджера ресурсов. Например, серверным адресным пространством может быть CICS AOR или управляющий регион IMS.

сервисный процессор. Часть процессорного комплекса, осуществляющая обслуживание комплекса.

серийный номер тома. Номер на метке тома, присваиваемый при подготовке тома к использованию в системе.

сетевой ввод заданий (NJE). Средство JES2, обеспечивающее передачу определенных заданий, выходных данных системы, команд оператора и сообщений между подсистемами ввода заданий, соединенными двоично-синхронными линиями связи, CTC-адаптерами и общими очередями.

сетевой оператор. (1) Сотрудник, отвечающий за управление работой телекоммуникационной сети. (2) VTAM-приложение, авторизованное для ввода команд сетевого оператора.

сеть. Набор продуктов обработки данных, соединенных линиями связи для обмена информацией между станциями.

сильносвязанная структура. Несколько центральных процессоров, совместно использующих память и управляемых одной копией z/OS. См. также *слабосвязанная структура*, *сильносвязанный мультипроцессор*.

сильносвязанное мультипроцессирование. Две вычислительные системы, параллельно работающие под управлением одной управляющей программы, совместно используя ресурсы.

сильносвязанный мультипроцессор. Любое центральное процессорное устройство с несколькими центральными процессорами.

символ управления кареткой. Произвольный символ в записи входных данных, определяющий операцию записи, пробела или пропуска.

символ. Буква, цифра или другой знак. Буква, цифра или другой знак, используемый при формировании, управлении или представлении данных. Внешне символ часто представлен смежными или соединенными штрихами.

синтаксис. Правила, регулирующие структуру языка программирования и построение операторов языка программирования.

синхронные сообщения. Сообщения WTO или WTOR, выдаваемые системой z/OS при определенных ситуациях восстановления.

сисплекс. Набор систем z/OS, взаимодействующих друг с другом через определенные многосистемные аппаратные компоненты и программные службы и обрабатывающих клиентскую нагрузку. См. также *Parallel Sysplex*.

система базового уровня. В SMP/E: уровень модулей, макросов, исходного кода и дистрибутивных библиотек целевой системы, созданных при генерации системы, для которых применимы модификации функций и служб.

система восстановления. Система, используемая вместо первичной прикладной системы, более недоступной для использования. Данные прикладной системы должны быть доступны для использования в системе восстановления. Это обычно обеспечивается с помощью методов резервного копирования и восстановления либо посредством различных методов копирования DASD, таких как удаленное копирование.

система управления базами данных (СУБД). Программная система, управляющая созданием, организацией и изменением базы данных и доступом к хранящимся в ней данным.

система. Сочетание конфигурации (аппаратного обеспечения) и операционной системы (программного обеспечения). Часто называется системой z/OS.

системная библиотека. Собрание наборов данных или файлов, в которых хранятся части операционной системы.

системная консоль. В z/OS: консоль, подключенная к процессорному контроллеру, используемому для инициализации системы z/OS.

системная модификация (SYSMOD). Входные данные SMP/E, определяющие внедрение, замену или обновление элементов операционной системы и связанных дистрибутивных библиотек, подлежащих установке. Системная модификация определяется набором MCS.

системные данные. Наборы данных, требуемые операционной системе z/OS или ее подсистемам для инициализации.

системный аварийный останов. Аварийный останов, вызванный неспособностью операционной системы обработать программу; может быть вызван ошибками в логике исходной программы.

системный инженер (SE). Представитель IBM, выполняющий обслуживание программного обеспечения IBM на месте.

скомпонованная библиотека. Набор данных, содержащий объектные модули с отредактированными связями.

слабосвязанная структура. Многосистемная структура, требующая ограниченного уровня взаимодействия между несколькими образами z/OS для обработки рабочей нагрузки. См. также *сильносвязанная структура*.

следующая последовательная ин-струкция. Следующая инструкция, выполняемая при отсутствии ветвления или передачи управления.

слово состояния программы (PSW). 64-битная структура в основной памяти, используемая для управления порядком выполнения инструкций и для хранения и указания состояния вычислительной системы по отношению к определенной программе. См. также *программная маска*.

службы сопряжения. В сисплексе: функции XCF, передающие данные и информацию о состоянии между участниками группы, находящимися на одной или нескольких системах z/OS в сисплексе.

смешанный комплекс. Комплекс глобальной синхронизации ресурсов, в котором одна или несколько систем не являются частью многосистемного сисплекса.

смещение. Количество единиц измерения от произвольной начальной точки записи, области или управляющего блока до какой-то другой точки.

совместимость. Способность работать в системе или способность работать с другими устройствами или программами.

совместное существование. Две или больше систем разных уровней (например, программного, сервисного или операционного уровня), осуществляющих совместное использование ресурсов. Совместное существование включает способность системы реагировать следующим образом на новую функцию, внедряемую на другой системе, с которой она совместно использует ресурсы: игнорирова-

ние новой функции; постепенная остановка; поддержка новой функции.

согласованная копия. Копия сущности данных (например, логического тома), включающая содержимое всей сущности данных на определенный момент времени.

соглашение об уровне сервиса (SLA). Письменное соглашение об обслуживании информационных систем, предоставляемое пользователям вычислительной системы.

соединительный кабель. Оптоволоконный кабель, используемый для переключаемых соединений между портами связи.

создание логических разделов. Функция операционной системы, позволяющая создавать логические разделы.

создание разделов. Процесс формирования нескольких конфигураций на основе одной конфигурации.

сообщение оператора. Сообщение операционной системы, направляющее оператора на выполнение определенной функции, например, подключение катушки с лентой; или информирующее оператора об определенных ситуациях в системе, например, об ошибках.

сообщение-уведомление оператора (WTO). Сообщение, отправляемое на консоль оператора, информирующее оператора об ошибках и состояниях системы, которые могут требовать коррекции.

сообщение-уведомление оператора с ответом (WTOR). Сообщение, отправляемое на консоль оператора, информирующее оператора об ошибках и состояниях системы, которые могут требовать коррекции. От оператора требуется ответ.

сопутствующие условия. Системные модификации (SYSMOD), каждая из которых может быть корректно установлена, только если установлена другая системная модификация. Сопутствующие условия определяются операндом REQ оператора ++VER.

состояние ожидания. Синоним: *время ожидания.*

состояние приостановки. Состояние, при котором выполняется обновление только одного из устройств пары томов двойного копирования или удаленного копирования по причине постоянной ошибки или команды авторизованного пользователя. При этом регистрируются все операции записи на приостановленное функциональное устройство. Это позволяет осуществлять автоматическую ресинхронизацию обоих томов, когда пара томов будет переведена в активное дуплексное состояние.

спин-набор данных. Набор данных, освобождаемый (доступный для печати) при закрытии. Поддержка спин-наборов данных обеспечивается для выходных наборов данных непосредственно перед завершением задания, создавшего набор данных.

списковая структура. Структура устройства сопряжения, позволяющая многосистемным приложениям в сисплексе совместно использовать информацию, организованную в набор списков или очередей. Списковая структура содержит набор списков и дополнительную таблицу блокировок, используемую для синхронизации доступа к ресурсам в списковой структуре. Каждый список содержит очередь записей списка.

список доступа. Список в профиле, содержащий всех авторизованных пользователей и их полномочий доступа.

спул (spool, simultaneous peripheral operations online). Чтение и запись входных и выходных потоков на устройства вспомогательной памяти, одновременно с выполнением задания, в формате, удобном для дальнейшей обработки или операций вывода.

спулинг. Чтение и запись входных и выходных потоков на устройствах вспомогательной памяти, параллельно с выполнением задания в формате, удобном для дальнейшей обработки или применения операций вывода.

среда времени выполнения. Набор ресурсов, используемых для поддержки выполнения программы. Синоним: *среда выполнения.*

среда распределенных вычислений (DCE). Полный интегрированный набор служб, поддерживающий разработку, использование и обслуживание распределенных приложений. DCE не зависит от операционной системы и сети; она обеспечивает совместимость и переносимость на различных платформах.

средства управления системой (SMF). Компонент z/OS, содержащий средства сбора и записи информации для определения использования системы.

средство авторизации программ (APF). Средство, осуществляющее идентификацию программ, авторизованных для использования ограниченных функций.

Средство восстановления ресурсов (RRS). Компонент системы z/OS, содержащий функции, вызываемые менеджером ресурсов для защиты ресурсов. RRS является

менеджером точки синхронизации системного уровня z/OS.

средство обработки сообщений (MPF). Средство, используемое для управления сохранением, изъятием и представлением сообщений.

Средства управления доступом к ресурсам (RACF). Менеджер безопасности производства IBM, обеспечивающий управление доступом путем идентификации и проверки пользователей в системе, осуществляющий авторизацию доступа к защищенным ресурсам, регистрирующий обнаруженные попытки несанкционированного входа в систему и регистрирующий обнаруженные факты доступа к защищенным ресурсам.

сторона. Одна из конфигураций, создаваемых при создании физических разделов.

страница. (1) В системах виртуальной памяти: блок инструкций и/или данных фиксированной длины, который можно передавать между основной памятью и внешней страничной памятью. (2) Подкачка – передача инструкций и/или данных между основной памятью и внешней страничной памятью.

страницы разделителя заданий. Страницы печатных выходных данных, отделяющие задания.

страничный обмен. В z/OS: процесс передачи страниц между основной памятью и внешней страничной памятью.

строка инструкций. В z/OS: часть экрана консоли, содержащая сообщения об управлении консолью и ошибках ввода.

структура. Конструкция, используемая в z/OS для отображения и управления памятью устройства сопряжения. См. *кеш-структура*, *списковая структура* и *блокировочная структура*.

СУБД. Система управления базами данных.

супервизор. Часть z/OS, координирующая использование ресурсов и обслуживающая поток операций единиц обработки.

суперпользователь. (1) Пользователь системы, работающий без ограничений. Суперпользователь имеет особые права и привилегии, необходимые для выполнения административных задач. В z/OS является аналогом пользователя в привилегированном или супервизорном режиме. (2) Пользователь системы, который может пройти все проверки безопасности в z/OS UNIX. Суперпользователь имеет особые права и привилегии, необходимые для управления процессами и файлами.

считыватель. Программа, считывающая задания с устройства ввода или из файла базы данных и помещающая их в очередь заданий.

Т

таблица подходящих устройств (EDT). Определенное инсталляцией представление устройств, подходящих для распределения ресурсов. EDT определяет групповые и родовые связи этих устройств. Во время начальной загрузки инсталляция определяет EDT, используемую в z/OS. После начальной загрузки задания могут запрашивать распределение устройств из любой группы устройств, назначенных выбранной EDT. EDT идентифицируется по уникальному (двузначному) идентификатору и содержит одно или несколько определений групповых и родовых связей.

таймаут. Время в секундах, в течение которого управление памятью остается в состоянии занятости, прежде чем завершатся физические сеансы.

твердая копия журнала. В системах с поддержкой нескольких консолей или графической консолью представляет постоянную запись об активности системы.

текст сообщения. Часть сообщения, содержащая информацию, направляемую пользователю на терминал или в программу.

терминал. Устройство, обычно оборудованное клавиатурой и дисплеем, способное передавать и принимать информацию по каналу связи.

тип данных. Свойства и внутреннее представление, характеризующее данные.

тип устройства. Общее название типа устройства; например, 3390.

том. (1) Пространство хранения на DASD, магнитной ленте или оптических устройствах, идентифицируемое по метке тома. (2) Часть единицы хранения, доступная для отдельного механизма чтения/записи, например, барабан, дисковый пакет или часть модуля дисковой памяти. (3) Записываемый носитель, подключаемый и отключаемый как одно целое, например, катушка магнитной ленты или дисковый пакет.

точка входа. Адрес или метка первой инструкции, выполняемой после ввода программы для выполнения. В загрузочном модуле: участок, в который передается управление после вызова загрузочного модуля.

транзакция. Единица работы, выполняемая одной или несколькими программами обработки транзакций, включающая определен-

ный набор входных данных и инициирующая определенный процесс или задание.

тупиковая ситуация (deadlock). (1) Состояние ошибки, при котором нельзя продолжить обработку, так как два элемента процесса ожидают друг от друга действия или ответа. (2) Неразрешимый конфликт в связи с использованием ресурсов. (3) Безвыходное положение, возникающее, когда несколько процессов ожидают доступности ресурса, который недоступен в связи с тем, что он занят другим процессом, находящимся в подобном состоянии ожидания.

У

удаленная функция. В SMP/E: функция, удаленная из системы вследствие установки другой функции. Обозначается подзаписью DELBY в записи SYSMOD для удаленной функции.

удаленное копирование. Функция аварийного восстановления и переноса рабочей нагрузки, которая может копировать данные на удаленный узел в режиме реального времени. Доступно два варианта удаленного копирования. См. *одноранговое удаленное копирование* и *расширенное удаленное копирование*.

удаленные операции. Эксплуатация удаленных узлов с базовой системы.

удаленный ввод заданий (RJE). Передача операторов управления заданиями и данных с удаленного терминала, вызывающая планирование и выполнение описываемых задач так, как если бы они поступали из входного потока.

узел назначения. Узел, осуществляющий выполнение приложений для авторизованного внешнего пользователя.

указатель. Адрес или другое указание расположения.

управление автоматическим перезапуском. Функция восстановления z/OS, повышающая доступность пакетных заданий и запускаемых задач. При неожиданном отказе задания или системы, в которой оно выполняется, z/OS может перезапустить задание без вмешательства оператора.

управление памятью. Операции распределения, размещения, мониторинга, миграции, резервного копирования, возврата, восстановления и удаления наборов данных. Эти операции могут выполняться вручную или с использованием автоматизированных процессов. Storage Management Subsystem позволяет автоматизировать эти процессы с оптимизацией ресурсов хранения. См. также *Storage Management Subsystem*.

управление программами. Задача подготовки программ к выполнению, хранения программ, загрузки модулей или программных объектов в программных библиотеках и их выполнение в операционной системе.

управление программой. Функции системы, осуществляющие активацию и вызов программы для выполнения в соответствующей среде выполнения.

управляющая секция (CSECT). Часть программы, определенная программистом как перемещаемый модуль, все элементы которого предназначены для загрузки в соседние участки основной памяти.

управляющий блок. Область памяти, используемая компьютерной программой для хранения управляющей информации.

управляющий интервал (CI). Область фиксированной длины или диск, на котором VSAM хранит записи и создает распределяемое свободное пространство. Кроме того, в наборе данных, упорядоченном по ключам (key-sequenced data set) или в файле, управляющий интервал представляет набор записей, на которые указывает индексная запись из последовательного списка. Управляющий интервал представляет единицу информации, которую VSAM передает на диск или с диска. Управляющий интервал всегда включает целое число физических записей.

управляющий оператор. В языках программирования: оператор, используемый для изменения непрерывного последовательного выполнения операторов; управляющий оператор может быть условным оператором (например, IF) или предписывающим оператором (например, STOP). В JCL: оператор задания, используемый при идентификации задания или описании его требований в операционной системе.

управляющий регион. Область основной памяти, содержащая подсистему менеджер работ подсистемы или менеджер ресурсов подсистемы.

уровень модификации. Дистрибутив, содержащий все временные исправления, выпущенные со времени последнего уровня модификации. Изменение уровня модификации не добавляет новые функции и не изменяет категорию программной поддержки версии, к которой он относится. Сравните: *выпуск* и *версия*. При выходе нового выпуска программы устанавливается уровень модификации 0. При следующем выходе выпуска с накопленными изменениями, уровень модификации увеличивается на 1.

уровень программы. Уровень модификации, выпуск, версия и уровень исправлений.

уровень сервиса. Значения FMID, RMID и UMID для элемента. Уровень сервиса идентифицирует владельца элемента, последний SYSMOD для замены элемента и все SYSMOD, обновившие элемент с момента его последней замены.

установка. В SMP/E: применение SYSMOD для целевых библиотек или принятие SYSMOD в дистрибутивных библиотеках.

устройство ввода-вывода. Принтер, привод для магнитных лент, жесткий диск и т. д. Устройства логически группируются в модули, которые, в свою очередь, группируются в последовательности. Первый модуль, называемый началом последовательности, содержит интерфейсы последовательности, соединяющие интерфейсы контроллера с CHPID процессора. Устройства представлены строками текста в соответствующем объекте модуля в схеме конфигурации.

устройство записи выходных данных. Компонент планировщика заданий, передающий заданные выходные наборы данных на системное устройство вывода независимо от программы, сгенерировавшей наборы данных.

устройство страничного обмена. В z/OS: устройство прямого доступа, на котором хранятся страницы (и, возможно, другие данные).

устройство управления (CU). Каждый физический контроллер сдержит одно или несколько устройств управления, транслирующих высокоуровневые запросы в низкоуровневые запросы между процессором и устройствами. Синоним: *блок управления устройством.*

устройство хранения с прямым доступом (DASD). Устройство, в котором время доступа независимо от размещения данных.

устройство. Компьютерное периферийное устройство или объект, представляемый приложению таким образом.

уточненное имя. Имя набора данных, содержащее последовательность имен, разделенных точками; например, «TREE.FRUIT.APPLE».

Ф

файл определения ввода-вывода (IODF). Линейный набор данных VSAM, содержащий информацию об определениях ввода-вывода, включая определения ввода-вывода процессоров и определения ввода-вывода операционной системы, включающие все логические объекты и их связь в аппаратной конфигурации.

файл. Именованный набор связанных записей данных, хранящийся и извлекаемый по заданному имени. Аналог набора данных в z/OS.

файловая система UNIX. Раздел дерева файлов UNIX, физически размещенный на одном устройстве или дисковом разделе, который можно отдельно подключать, отключать и администрировать. См. также *иерархическая файловая система.*

физический раздел. Часть CPC, работающая как отдельный CPC и содержащая собственную копию операционной системы.

физическое устройство (PU). (1) Устройство управления или контроллер кластера SNA-терминала. (2) Часть устройства управления или контроллера кластера, исполняющая роль физического устройства, как ее описывает SNA.

фиксация. Запрос на внесение всех изменений в ресурсах с момента последней фиксации или отката или, для первой единицы восстановления, с начала выполнения приложения.

фоновое задание. (1) Задание с низким приоритетом, обычно представляющее собой пакетное или неинтерактивное задание. (2) В TSO: задание, вводимое командой SUBMIT или через SYSIN. Сравните: *оперативное задание.*

фоновый режим. (1) В мультипрограммировании: среда, в которой выполняются программы с низким приоритетом. (2) В TSO/E: среда, в которой выполняются задания, переданные командой SUBMIT или через SYSIN. Региону основной памяти одновременно назначается одно задание, которое остается в основной памяти до своего завершения. Сравните: *оперативный режим.*

формат записи. Для тома на магнитной ленте: формат данных на ленте, например, с плотностью 18, 36, 128 или 256 дорожек.

фрейм. В кластере микропроцессоров мэйнфрейма фрейм содержит один или два центральных процессорных комплекса (CPC), элементы поддержки и устройства распределения электроэнергии переменного тока.

функция. В SMP/E: продукт (например, компонент системы или лицензированная программа), который при необходимости можно установить в системе пользователя. В SMP/E функции определяются оператором ++FUNCTION. Каждая функция должна иметь уникальный FMID.

Х

хранилище данных. Система, содержащая критические бизнес-данные организации. Система хранилища данных поддерживает точность и актуальность данных, представляя данные ответственным за принятие решений таким образом, чтобы они могли их эффективно использовать.

Ц

целевая библиотека. В SMP/E: собрание наборов данных, в которых хранятся разные части операционной системы. Эти наборы данных иногда называют системными библиотеками.

целевая зона. В SMP/E: собрание VSAM-записей, описывающих системные макросы, модули, сборки, загрузочные модули, исходные модули и библиотеки, копируемые из DLIB во время генерации системы, а также SYSMOD, применяемые в целевой системе.

целостность данных. Состояние, имеющее место при отсутствии случайного или преднамеренного повреждения, изменения или потери данных.

центральная память. См. *основная память.*

центральное процессорное устройство (CPU). Синоним: *процессор.*

центральный процессор (CP). Часть компьютера, содержащая средства упорядочения и обработки для выполнения инструкций, начальной загрузки и других машинных операций.

центральный процессорный комплекс (CPC). Физический набор аппаратного обеспечения, включающий основную память, один или несколько центральных процессоров, таймеры и каналы.

цикл. Ситуация, при которой инструкция или набор инструкций повторно выполняются несколько раз.

Ч

частично заданное имя набора данных. Имя набора данных с пропущенными квалификаторами. Вместо неопределенных квалификаторов используются звездочки и знаки процента.

чувствительность к регистру. Относится к способность различать буквы верхнего и нижнего регистра.

Ш

шаг задания. Операторы JCL, запрашивающие и управляющие выполнением программы и определяющие ресурсы, нужные для выполнения программы. К JCL-операторам шага задания относятся один оператор EXEC, определяющий вызываемую программу или процедуру, за которым следует один или несколько операторов DD, определяющих наборы данных или устройства ввода-вывода, которые могут быть нужны программе.

шестнадцатеричная система. Система счисления с основой 16. Шестнадцатеричные цифры включают знаки 0–9 (десятичные 0–9) и A–F (десятичные 10–15), представляющие значения от 0 до 15.

шифрование. Систематическое кодирование данных таким образом, чтобы их нельзя было просмотреть, не зная ключа кодирования.

шкаф. Корпус для панелей, организованный в группы портов связи, представляющих собой пары оптоволоконных адаптеров или разветвителей. Шкафы используются для прокладки длинных, сложных кабелей между процессорами и контроллерами, которые могут находиться на расстоянии другого вычислительного центра. Также называются *шкаф управления оптоволоконными подключениями.*

шлюзовой узел. Узел, выступающий в качестве интерфейса между сетями.

Э

ЭБ. См. *экзабайт.*

эзотерическое имя (esoteric). Эзотерическое (групповое) имя задает определенный инсталляцией и именованный набор устройств ввода-вывода, относящихся обычно к одной группе устройств. Таблицы подходящих устройств (EDT) определяют групповые и родовые связи этих устройств. Имя, назначаемое эзотерической группе устройств, используется в операторе JCL DD. Затем задание назначает устройство из этой группы, а не с определенным номером устройства или из родовой группы устройств.

экзабайт. Для процессора: размер основной и виртуальной памяти и объем канала: 1 152 921 504 606 846 976 байт или $2^{(60)}$.

электронная коммерция. (1) Бизнес-транзакция, выполняемая через электронную среду, например, через Интернет. (2) Преобразование основных бизнес-процессов посредством использования Интернет-технологий.

элемент поддержки. Аппаратное устройство, обеспечивающее функции связи, мониторинга и диагностики центрального процессорного комплекса (CPC).

элемент управления системой (SCE). Аппаратное средство, осуществляющее передачу данных и управляющей информации, связанных с запросами памяти, между элементами процессора.

элемент. В SMP/E: часть продукта, такая как макрос, модуль, диалоговая панель или пример кода.

Ю

юлианское представление даты. Формат даты, содержащий год в позициях 1 и 2 и день в позициях 3–5. День представлен цифрами от 1 до 366, выровненными по правому краю, с заполнением неиспользуемых позиций нулями.

Я

ядро (kernel). Часть операционной системы, выполняющая основные функции, в частности, распределение аппаратных ресурсов.

ядро (nucleus). Часть управляющей программы, которая всегда остается в основной памяти.

язык C. Высокоуровневый язык, используемый для разработки программных приложений в компактном эффективном коде, который можно выполнять на разных типах компьютеров с минимальными изменениями.

язык управления заданиями (JCL). Последовательность команд, используемая для идентификации задания в операционной системе и описания требований задания.

Примечания

Информация в этом руководстве охватывает продукцию и услуги, предлагаемые в США.

Предложения IBM по услугам, товарам и их возможностям, описанным в руководстве, могут не действовать в других странах. За информацией о текущем ассортименте доступных продуктов и услуг обращайтесь в местные представительства IBM. Явные и неявные упоминания услуг, продуктов и их возможностей не означают необходимости их применения. Допускается их замена любыми функционально эквивалентными продуктами и службами сторонних производителей, не нарушающими прав на интеллектуальную собственность IBM. При этом ответственность за проверку совместимости и продуктивности решений сторонних производителей принимает на себя пользователь.

IBM может обладать патентами или патентными заявками на технологии, описанные в настоящем руководстве, предоставление которого не означает наличия лицензии на технологии. Письменные запросы лицензий следует направлять по адресу: *IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 USA.*

Приведенный ниже абзац не относится к Соединенному Королевству Великобритании и Северной Ирландии, а также иным странам, законодательству которых противоречит: КОРПОРАЦИЯ IBM ПРЕДОСТАВЛЯЕТ ДАННОЕ РУКОВОДСТВО «КАК ЕСТЬ» И НЕ ДАЕТ НИКАКИХ ЯВНЫХ ИЛИ ПОДРАЗУМЕВАЕМЫХ ГАРАНТИЙ, ВКЛЮЧАЯ (НО НЕ ОГРАНИЧИВАЯСЬ) ПОДРАЗУМЕВАЕМЫЕ ГАРАНТИИ ПАТЕНТНОЙ ЧИСТОТЫ, КОММЕРЧЕСКОЙ ПРИГОДНОСТИ И СООТВЕТСТВИЯ КОНКРЕТНОМУ НАЗНАЧЕНИЮ. В отдельных государствах отказ от явных и подразумеваемых гарантий по ряду сделок запрещен, так что указанное ограничение может вас не коснуться.

В руководстве возможны опечатки и технические неточности. Приводимые в нем сведения регулярно обновляются, соответствующие изменения будут внесены в новую редакцию руководства. Корпорация IBM оставляет за собой право в любое время без уведомления модифицировать описанные в этом руководстве продукты и программные средства.

Любые ссылки на сайты сторонних компаний в этом руководстве носят исключительно информационный характер и не свидетельствуют об их поддержке корпорацией IBM; риск, связанный с применением ресурсов этих сайтов, принимает на себя пользователь.

По своему усмотрению и без каких-либо обязательств IBM может использовать и распространять любые предоставленные сведения.

За информацией о сторонних продуктах обращайтесь к производителям, их публикациям и другим открытым источникам. IBM не тестировала эти продукты и не может подтвердить точность оценок производительности, совместимости и прочих параметров. Соответствующие вопросы следует направлять поставщикам этих продуктов.

В качестве примеров настоящее руководство содержит данные и отчеты, используемые в повседневной практике предприятий. Для наиболее полной иллюстрации примеров в руководстве встречаются имена лиц, названия компаний, торговых марок, товаров. Все они вымышлены, и любые совпадения с именами и данными реально существующих компаний случайны.

Авторское право

Данное руководство содержит примеры исходного кода прикладных программ, которые иллюстрируют приемы программирования на различных платформах. Вы можете копировать, изменять и распространять их в любом виде без отчислений в пользу IBM, руководствуясь целями разработки прикладных программ, включая коммерческие, в соответствии с интерфейсом прикладного программирования платформы, для которой предназначены эти программы. Примеры не подвергались всеобъемлющему тестированию, поэтому IBM не может явно или неявно гарантировать надежность, удобство в обслуживании и работоспособность приведенных примеров.

Товарные знаки

Ниже приведены товарные знаки корпорации IBM в США и (или) других странах:

Advanced Peer-to-Peer Networking®	Geographically Dispersed Parallel Sysplex™	RACF®
AD/Cycle®	GDDM®	RMF™
AIX®	GDPS®	S/360™
C/370™	HiperSockets™	S/370™
CICS®	IBM®	S/390®
CICSPlex®	IMS™	Sysplex Timer®
Domino®	Language Environment®	System z9™
DB2®	Lotus®	System/360™
DFS™	Multiprise®	System/370™
DFSMSdfp™	MVS™	System/390®
DFSMSdss™	MVS/ESA™	SAA®
DFSMShsm™	MVS/XA™	Tivoli®
DFSORT™	NetRexx™	VisualAge®
DRDA®	NetView®	VSE/ESA™
Encina®	Open Class®	VTAM®
Enterprise Storage Server®	OS/390®	WebSphere®
Enterprise Systems Architecture/390®	Parallel Sysplex®	z/Architecture™
ECKD™	Processor Resource/Systems Manager™	z/OS®
ESCON®	PR/SM™	z/VM®
FlashCopy®	QMF™	z/VSE™
FICON®	Redbooks™	zSeries®
		z9™

Другим компаниям принадлежат следующие товарные знаки:

EJB, Java, JDBC, JMX, JSP, JVM, J2EE, RSM, Sun, Sun Java, Sun Microsystems, VSM и все товарные знаки, связанные с Java-приложениями, являются товарными знаками Sun Microsystems, Inc. в США и (или) других странах.

Microsoft, Visual Basic, Windows и эмблема Windows – товарные знаки Microsoft Corporation в США и (или) других странах.

Intel, эмблема Intel, эмблема Intel Inside и эмблема Intel Centrino – товарные знаки или зарегистрированные товарные знаки Intel Corporation или ее подразделений в США и (или) других странах.

UNIX – зарегистрированный товарный знак Open Group в США и (или) других странах.

Linux – товарный знак, принадлежащий Линусу Торвальдсу (Linus Torvalds) в США и (или) других странах.

Названия других компаний, товаров и услуг могут быть товарными знаками соответствующих владельцев.

Введение в современные мэйнфреймы: основы z/OS

Основные понятия в сфере мэйнфреймов, включая использование и архитектуру

Основы z/OS для студентов и начинающих

Оборудование и периферийные устройства мэйнфреймов

Эта книга из серии IBM® Redbooks содержит практическую информацию, позволяющую приступить к использованию основных средств мэйнфрейм-компьютеров. Она открывает собой задуманную серию учебников, предназначенных для обучения студентов основным понятиям, связанным с мэйнфреймами, и подготовки студентов к деятельности в сфере вычислений с использованием больших систем. Оптимальный процесс обучения предполагает, что студенты успешно прошли вводный курс по основам компьютерных систем, в частности, по устройству и архитектуре компьютеров, операционным системам, управлению данными или системам передачи данных. Кроме того, предполагается, что студенты успешно прошли курсы по одному или нескольким языкам программирования и умеют работать с компьютером на уровне пользователя.

Этот учебник можно использовать для подготовки к курсам по расширенным задачам, а также для стажировки и специальных исследований. Он не претендует на полноту охвата всех аспектов функционирования мэйнфреймов, равно как и не является справочником, описывающим все функции и опции мэйнфреймов. Кроме того, этот курс может быть полезен для опытных специалистов в обработке данных, работавших с платформами, отличными от мэйнфреймов, или знакомых с некоторыми аспектами мэйнфреймов, но желающих познакомиться с другими функциями и преимуществами среды мэйнфреймов.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION (ITSO)

Разработка технических руководств на основе практического опыта

Книги IBM Redbook выходят в свет благодаря работе действующей при корпорации IBM Международной организации технической поддержки (IBM International Technical Support Organization). Актуальную техническую информацию, в основу которой положены реальные сценарии и примеры, сообщают эксперты IBM, ее партнеры и компании-клиенты со всего мира. Их точные и конкретные рекомендации помогут вам эффективнее осуществлять внедрение IT-решений в различных средах.

Подробнее см. на сайте
ibm.com/redbooks

Полезный ресурс
ibm.com/developerworks/ru

Учебный центр IBM в России: 123370, Москва, Краснопресненская наб., 18
Москва-Сити, бизнес-центр «Башня на Набережной»
Тел.: +7 (495) 775-8800, +7 (495) 940-2000
e-mail: ibmtraining@ru.ibm.com
http://www.ibm.com/ru/educ