Министерство образования и науки Российской Федерации
Государственное бюджетное образовательное учреждение высшего профессионального образования
МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Н.Э. БАУМАНА
(МГТУ им. Н.Э. Баумана)
Факультет «Информатика и системы управления»
Кафедра «Компьютерные системы и сети»

Утверждаю
Заведующий кафедрой ИУ-6,
Д.т.н., профессор
_____Сюзев В.В.
«___»_____ 2013 г.

Смирнова Е.В., Белоус В.В.

Сборник методических указаний по выполнению лабораторных работ на большой ЭВМ (мейнфрейм)

2013 год
Москва

Содержание:

Введение

Данный сборник представляет собой набор методических указаний к лабораторным работам, выполняемым студентами кафедры в рамках нескольких курсов:

- лабораторные работы № 1, № 2 и № 3 – в курсе по выбору «Операционная система zOS большой ЭВМ» в 7-ом семестре,
- лабораторные работы № 4 и № 5 – в курсе «ЭВМ» в 11-ом семестре.

Эти лабораторные работы объединены общей тематикой – выполняя их, студенты постепенно, по принципу от простого к сложному, осваивают большую вычислительную машину.

Выполняя первую работу, студенты учатся обращаться с операционной системой большой ЭВМ класса мейнфрейм IBM. Для этого они осваивают подсистему TSO (Time Sharing Option), обеспечивающую интерактивный пользовательский терминальный интерфейс, в этой подсистеме они изучают командный язык CLIST. Затем осваивают диалоговый интерфейс ISPF (Interactive System Productivity Facility), который значительно облегчает пользователю общение с операционной системой и имеет полноэкранный текстовый редактор. Студенты осваивают команды меню интерфейса ISPF, создают наборы данных разных типов и форматов, подготавливая себе средства разработки, с помощью которых они будут выполнять следующую лабораторную работу. Поскольку эта работа первая, ее задачей является лишь познакомить студентов со способами общения с операционной системой и освоить интерфейсы для этого общения.

Вторая и третья лабораторные работы выполняются в 7-ом семестре, здесь студенты учатся системному программированию в среде большой вычислительной машины, для этого изучают языки скриптов JCL и REXX.

Четвертая и пятая лабораторные работы посвящены написанию клиент-серверных приложений, которые будут выполняться в среде большой ЭВМ в рамках самого совершенного на сегодня программного обеспечения в области электронного бизнеса – IBM WebSphere. Для разработки этих приложений студенты осваивают студию разработки WebSphere Application Developer, использующую все современные приемы технологии программирования.

После выполнения этих лабораторных работ студент имеет представление всем спектре программного обеспечения большой ЭВМ, от особенностей работы операционной системы и ее подсистем до последних достижений в области электронного бизнеса.

Лабораторная работа №1

Последовательность работы в среде MainFrame с использованием эмулятора Vista TN3270

Методические указания по выполнению лабораторной работы
по курсу «ОПЕРАЦИОННАЯ СИСТЕМА Z/OS UNIX и
ВИРТУАЛЬНАЯ СРЕДА MAINFRAME»

Целями лабораторной работы являются:

1. Освоение способов подключения к виртуальной среде большой вычислительной машины Mainframe и работа в ее операционной системе z/OS.
2. Создание контейнеров данных (Data Set) для хранения файлов (Member).

Выполнение лабораторной работы

1. Подключение к серверу z/OS

Для того чтобы получить доступ к серверу z/OS с вашего компьютера, вам необходимо установить компонент доступа, называемый Клиент, примеры таких компонентов показаны на Рисунок1.

Workstation Client	Transmission Protocol	centraler Server component
Browser	HTTP	Web Server, e.g. Apache
FTP Client	FTP	FTP Server
Telnet Client	Telnet	Telnet Server

Рисунок 1.

Операционная система z/OS поддерживает множество способов обращения к ней, в частности, вы можете использовать протокол IBM 3270, который обеспечивает связь с сервером коммуникаций z/OS. Этот протокол использует протокол TELNET, как носитель,

и имеет доступ через TCP/IP порт 23. Клиент 3270 называют также эмулятором терминала Mainframe (см. Рисунок 2).

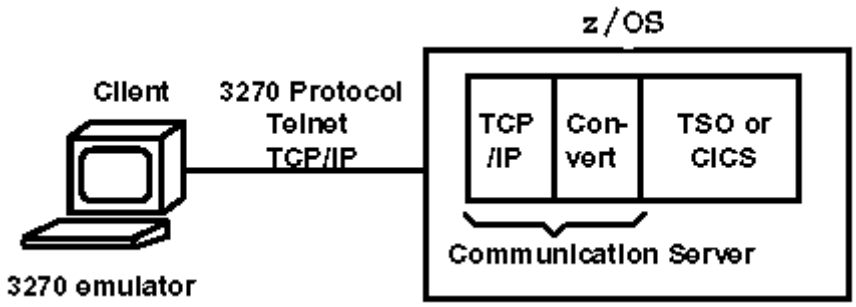


Рисунок 2. Клиент 3270 и сервер z/OS

Эмулятор 3270 включен во многие дистрибутивы Linux. Вы можете скачать дистрибутив эмулятора терминала Mainframe “Vista TN3270” с сайта кафедры.

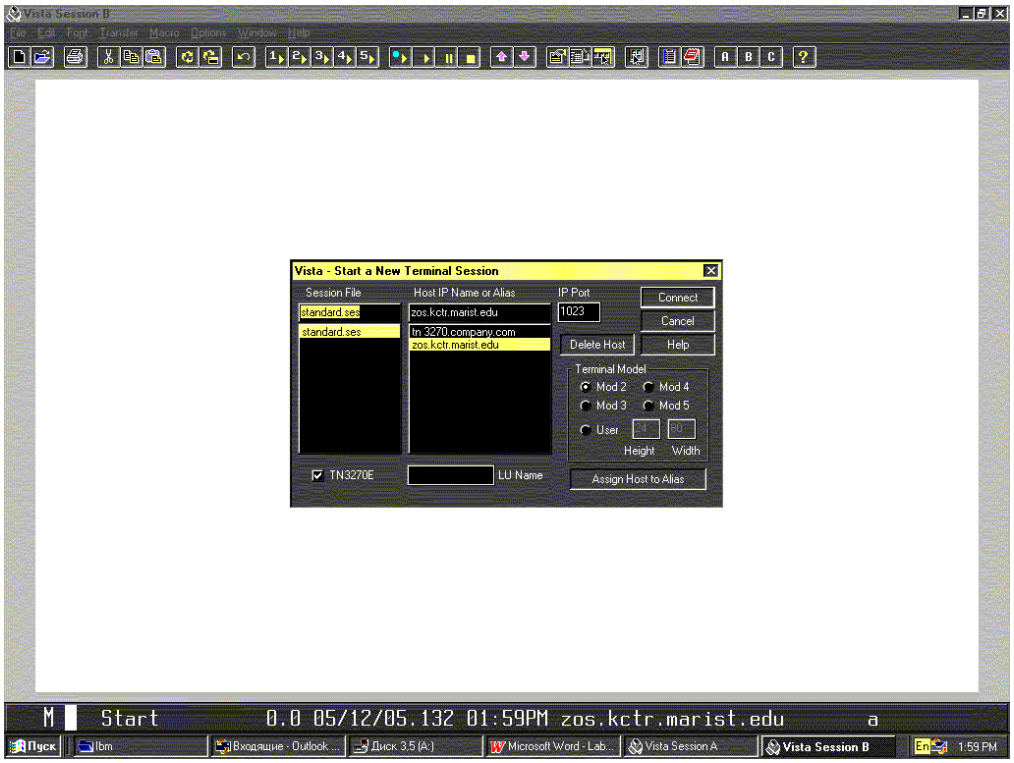


Рисунок 3. Окно интерфейса эмулятора “Vista TN3270”

Установите эмулятор терминала Mainframe “Vista TN3270” на своем компьютере. Запустите эмулятор терминала Mainframe “Vista TN3270” из рабочего окна Windows, указав номер порта и IP адрес сервера у преподавателя.

2. Вход в операционную систему z/OS

При удачном соединении на экране появится изображение, показанное на Рисунке 4.

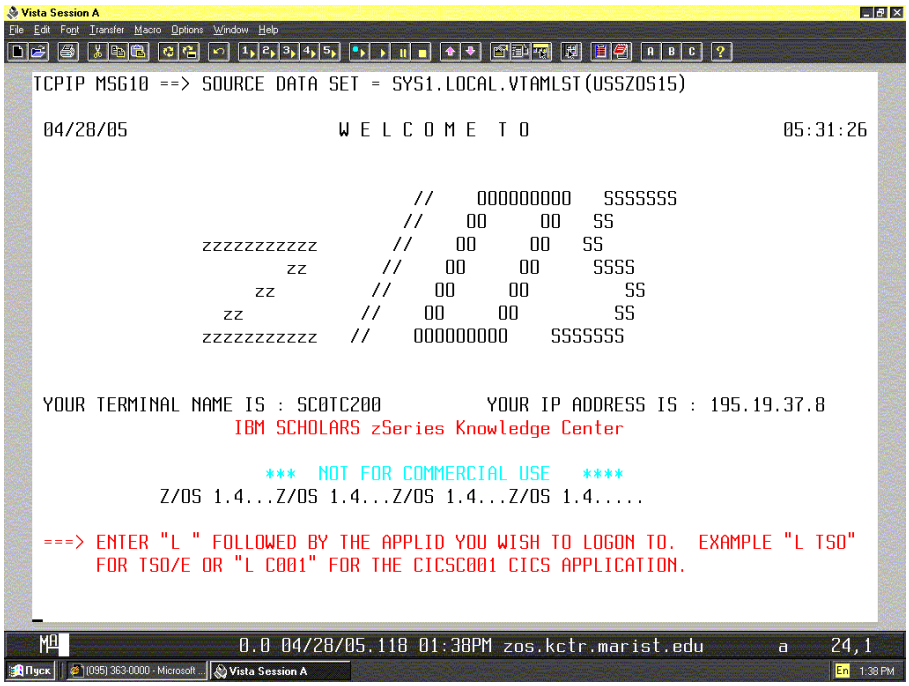


Рисунок 4 – Интерфейс операционной системы zOS большой ЭВМ класса IBM Mainframe

Нижняя строка экрана – командная строка. Вызовите подпрограмму TSO (Time Shared Operations), набрав текст «L TSO» и нажмите Enter.

Оболочка TSO – это подсистема операционной системы z/OS, позволяющая работать с ней с помощью набора команд. Подпрограмма ISPF – это пользовательский интерфейс, используется для упрощения общения с ОС. Функционально он похож на оболочку UNIX.

Архитектура серверов серии zSeries – это современная архитектура, имеющая ряд функций, недоступных на других платформах. Например, имеются существенные особенности у подсистемы ввода-вывода, ориентированной на параллельное обслуживание десятков тысяч пользователей, имеется виртуализация на аппаратном уровне, обеспечивается распределение функций и параллельных вычислений с учетом целей управления рабочими нагрузками, и многие другие. В то же время эта архитектура поддерживает и старые функции, в частности, пользовательский интерфейс ISPF. Именно поэтому все программы и приложения, написанные в прежние годы, могут быть выполнены на современной большой ЭВМ.

Отметим, что клавиатура клиентского приложения терминала большой ЭВМ, работающего по протоколу 3270, не полностью совместима с клавиатурой персонального компьютера. Изначально клавиша «Enter» – это клавиша «Control Key», то есть, правая

нижняя клавиша на персональном компьютере. Однако, многие эмуляторы 3270 позволяют переопределять позиции клавиш, особенно клавиши Enter.

После набора “L TSO” появится экран Рисунок 5, ожидающий ввода параметров учетной записи пользователя (логина и пароля).

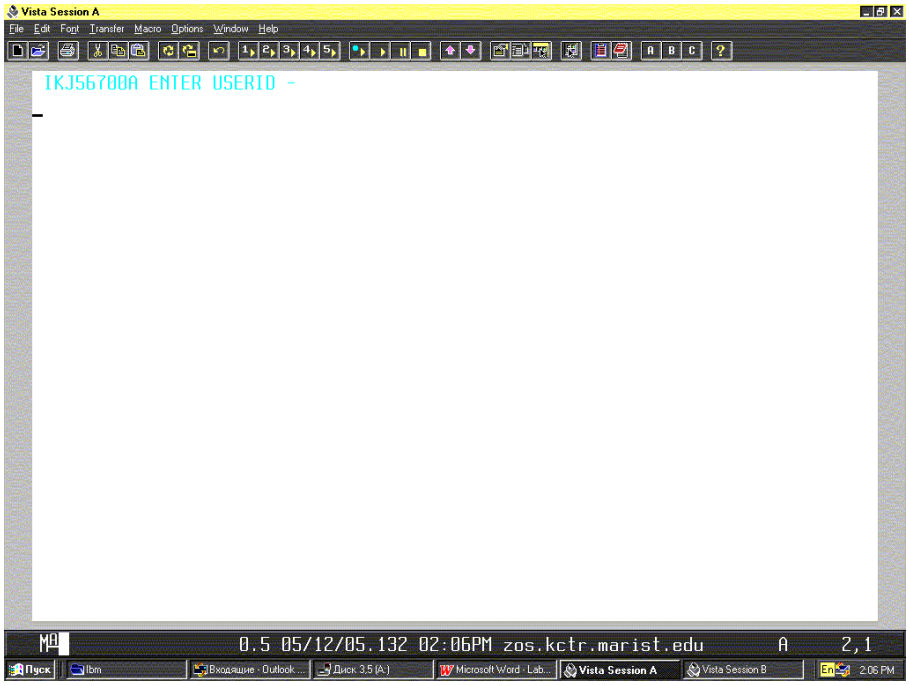


Рисунок 5 - Входной экран TSO.

Для работы в операционной системе z/OS Вы должны получить регистрационный номер (ID) и пароль (PW) у преподавателя.

После ввода логина перед вами появится приветственный экран TSO/E – последняя версия оболочки TSO, показанная на рисунке 6.

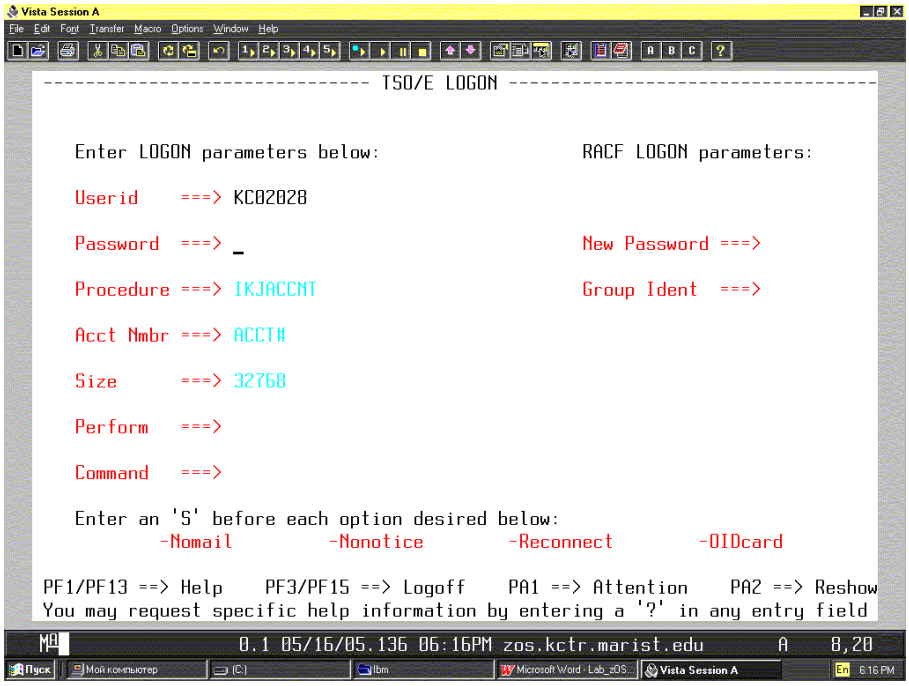


Рисунок 6. Экран входа в операционную систему TSO/E

Введите пароль и нажмите еще раз клавишу «Enter», и вы попадете в приветственный экран системы TSO, показанный на рисунке 7.

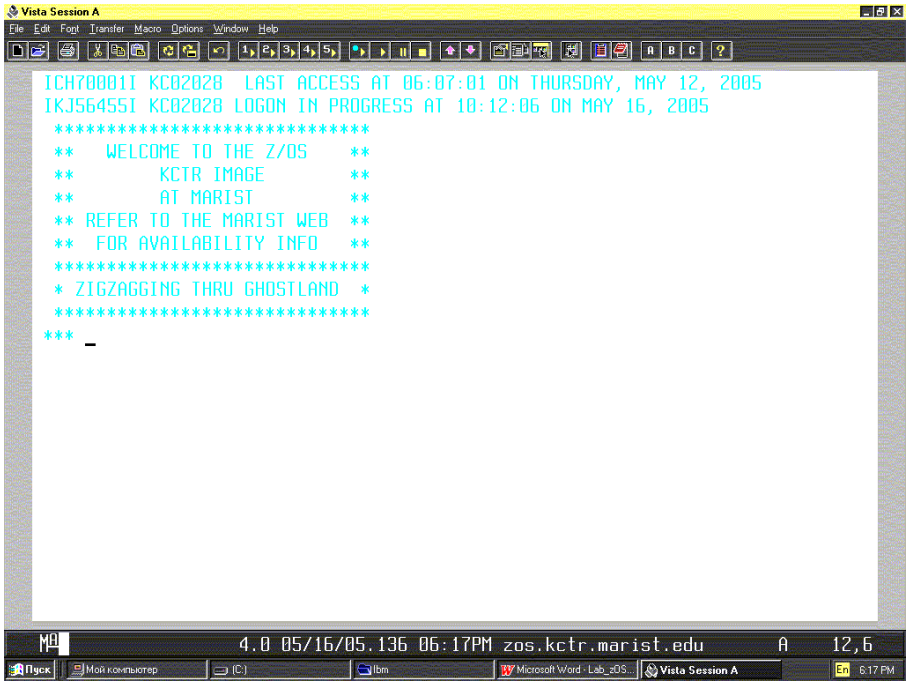


Рисунок 7. Внешний вид экрана TSO

Нажмите еще раз «Enter», и вы попадете в окно интерфейса ISPF (см. Рисунок8).

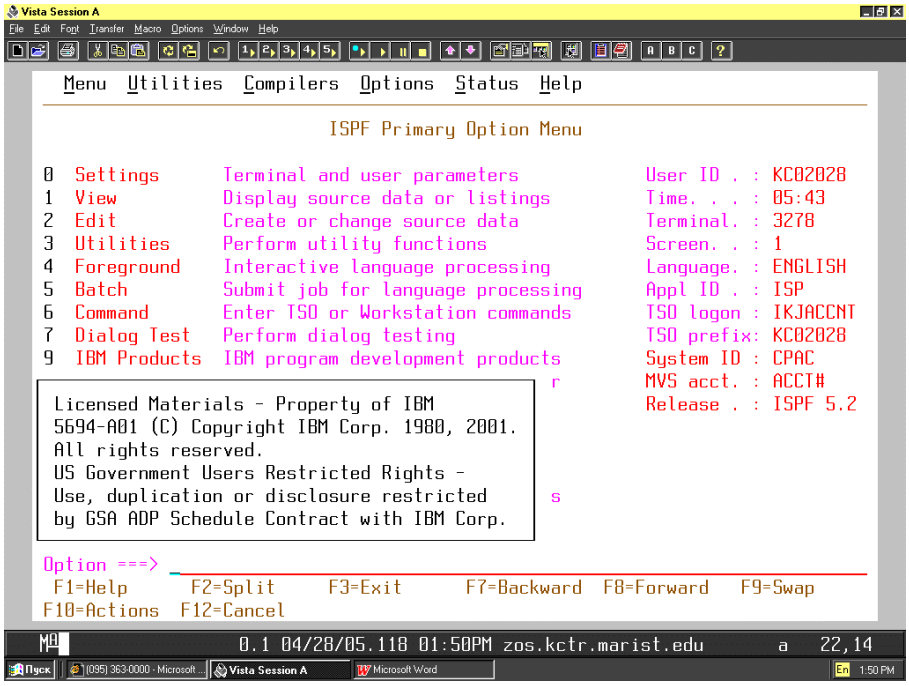


Рисунок 8. Начальное окно интерфейса ISPF

TSO предоставляет множество функций, которые можно вызвать из этого окна, введя команду в командной строке. Операционная система z/OS управляет многими подсистемами, причем каждая имеет свое собственное адресное пространство. Иерархическая взаимосвязь подсистем и функций, которые вы будете осваивать, показана на Рисунок 9.

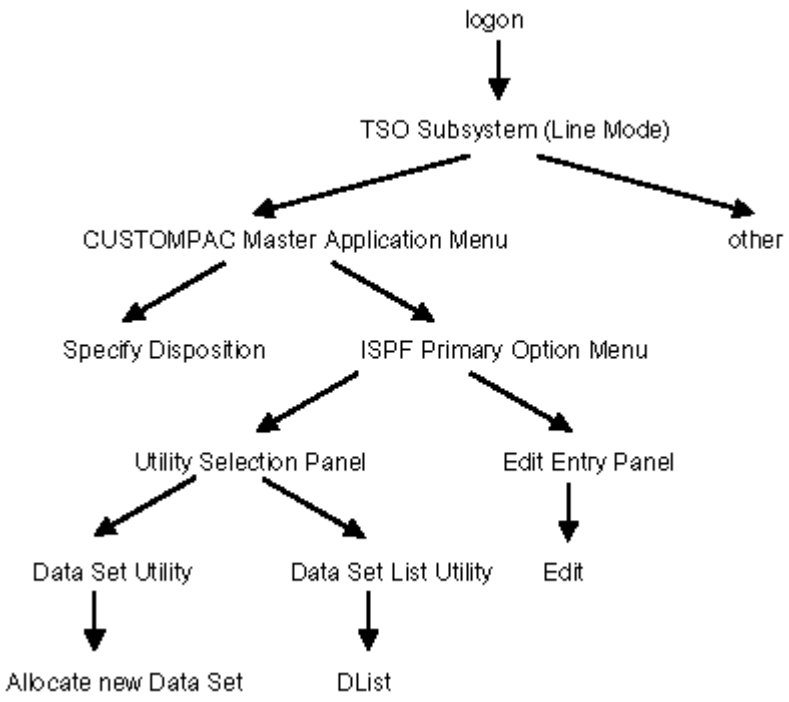


Рисунок 9.

Иерархическая структура подсистем и функций ISPF

3. Создание наборов данных (Data Set)

Создадим собственный набор данных, для этого из основного окна интерфейса ISPF (Рисунок 8) войдем в меню Utility Selection Panel, нажав «3» в командной строке. В оболочке TSO это называется вызвать окно «Панели». Появится панель “Utility Selection Panel” (Рисунок 10).

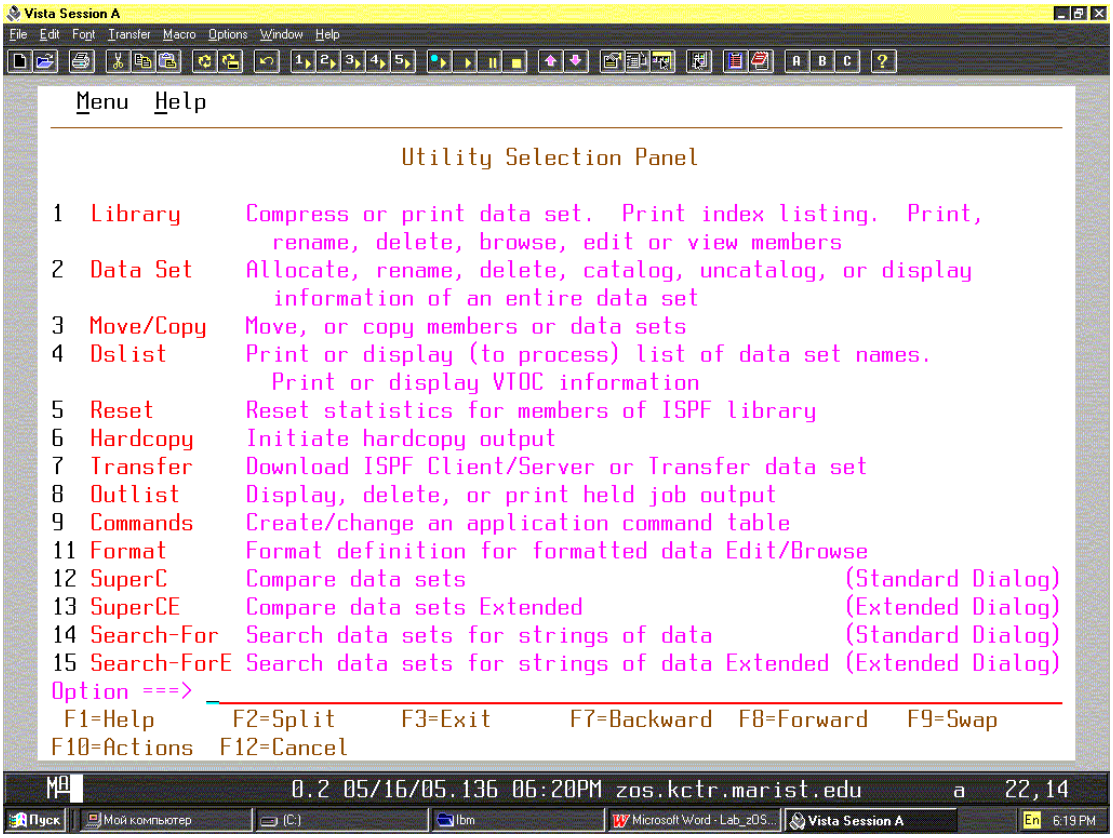


Рисунок 10. Окно подсистемы Utility Selection Panel

Для того, чтобы писать и запускать на выполнение программы, нам надо создать три контейнера:

1. для текстовых файлов – исходных кодов, текстов программ и др.;
2. для хранения JCL скриптов (аналог UNIX make file), которые инструктируют z/OS о компилировании и связях исходных кодов;
3. для хранения выполняемых файлов (машинных кодов).

Мы будем работать с типом файлов PDS (Partitioned Data Set), хотя имейте в виду, что типов файлов существует много. Структура контейнера распределенного типа (Partitioned Data Set) показана на Рисунок 11.

Directory	Member 1	Member 2	Member 3	……

Рисунок 11. Структура контейнера распределенного типа (Partitioned Data Set)

Контейнеры распределенного типа (Partitioned Data Set) – это тип мини файловой системы. Они имеют простую директорию и пространство для нескольких файлов, которые

называются Members. Контейнеры распределенного типа используют для хранения исходных кодов программ и для выполняемых файлов.

Нажимаем «2» – ПОПАДАЕМ В ОКНО “DATA SET UTILITY”

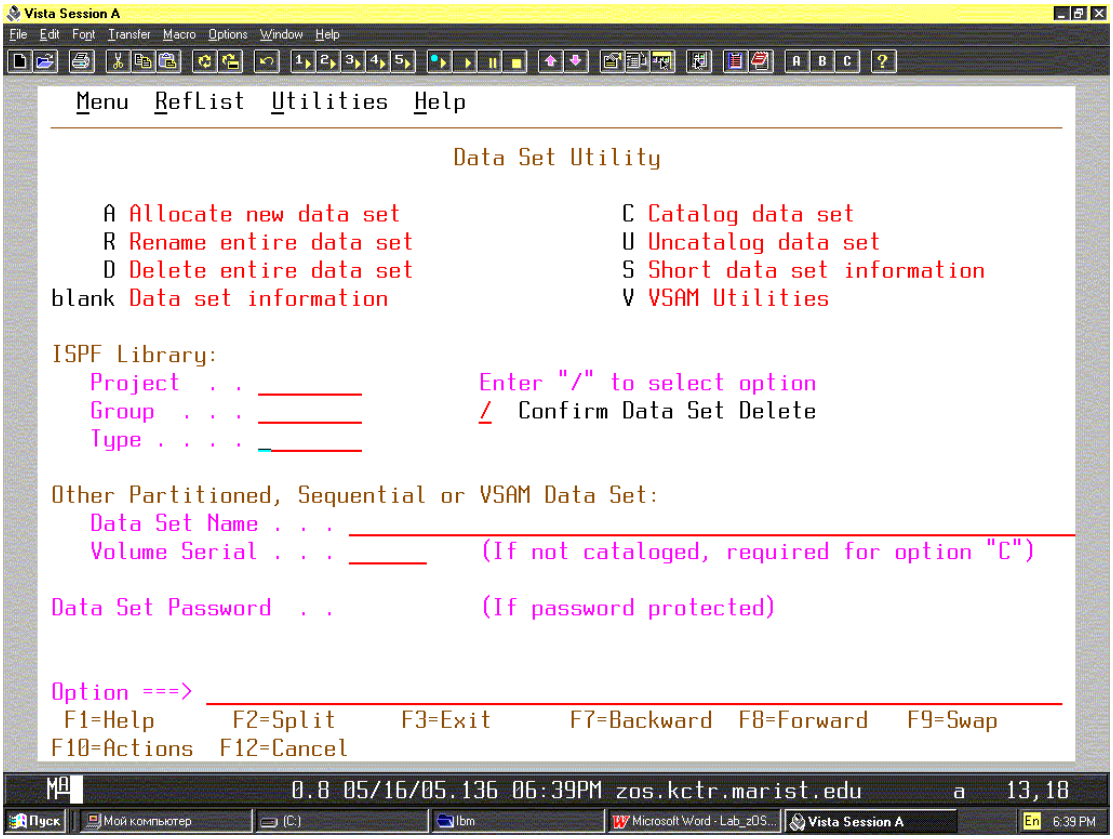


Рисунок 12. Окно «Data Set Utility»

Интерфейс ISPF предполагает, что мы введем имя контейнера, который должен быть определен. Имя контейнера в операционной системе z/OS состоит из трех полей, разделенных точкой, и имеет формат xxx.yyy.zzz. xxx, yyy, zzz – это слова из букв и цифр, имеющие максимум 8 символов каждый.

ВНИМАНИЕ!

МЫ В ЛАБОРАТОРНОЙ РАБОТЕ ВВОДИМ ИМЕНА ПОЛЕЙ, отличные от тех, которые указаны на Рисунке 13:

PROJECT - ZSCHOL
GROUP TEST
TYPE DATASET

ВАЖНО: Идентификатор первого уровня вашего имени должен совпадать с вашим ID!

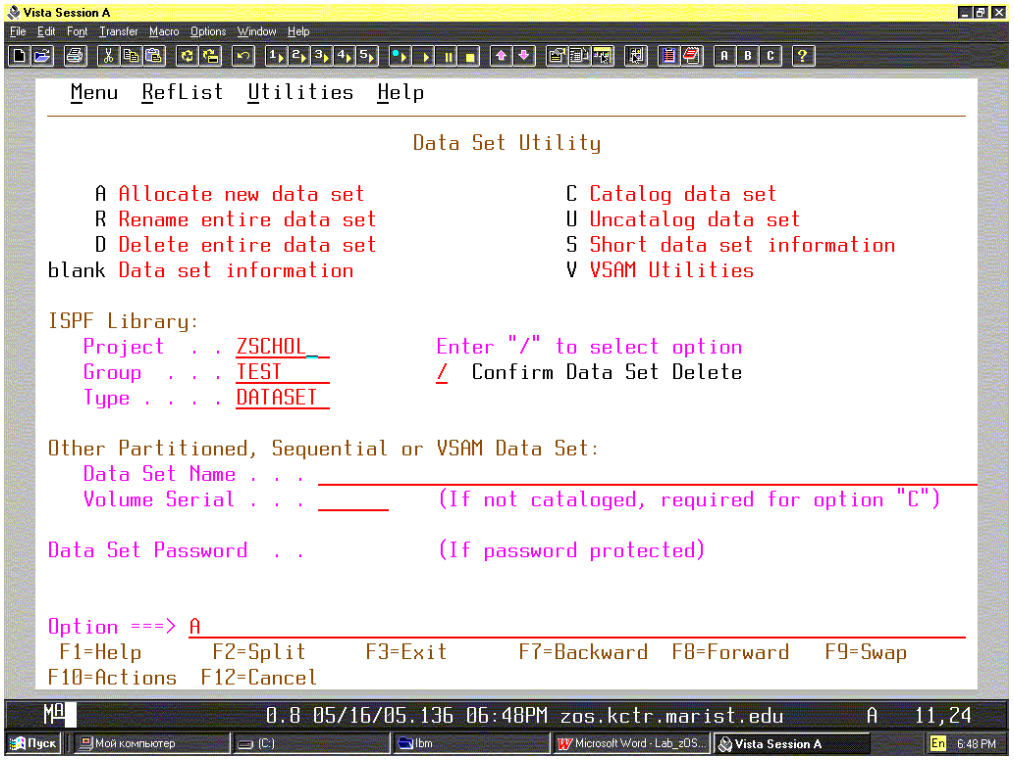


Рисунок 13. Окно «Data Set Utility» с введенными именами полей контейнера

В командной строке «Option --→» пишем «a» и нажимаем Enter. В результате открывается следующее окно подсистемы распределения пространства New Data Set Allocation (Рисунок 14).

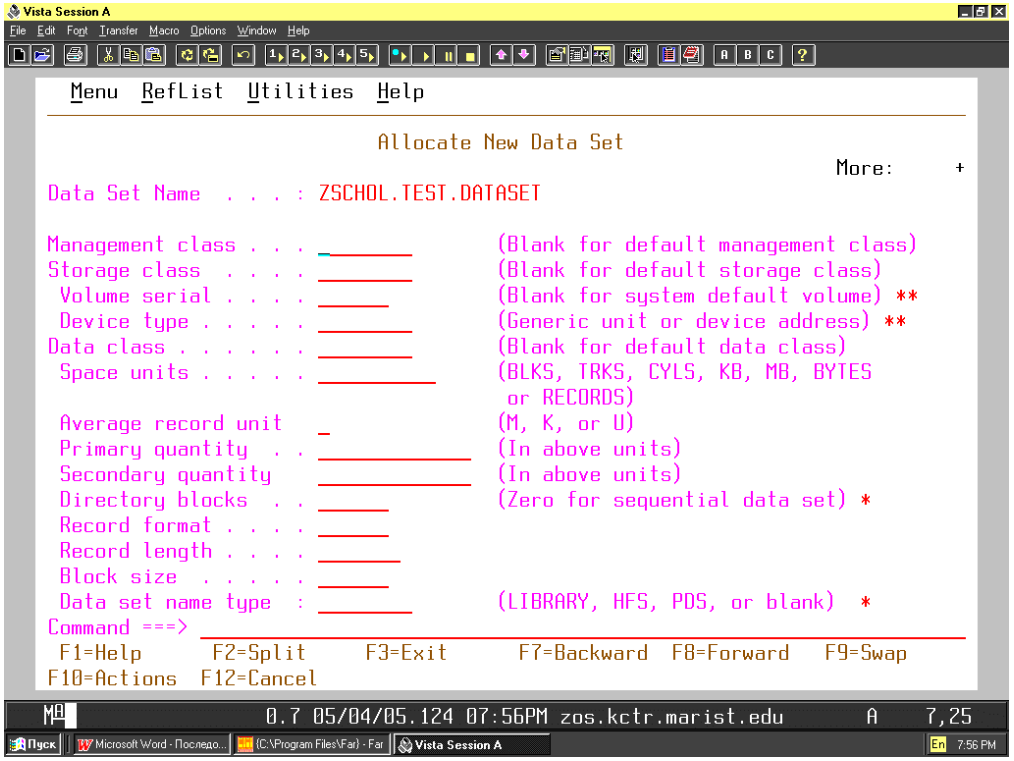


Рисунок 14. Окно определения нового набора данных

Мы хотим создать контейнер, в который будем помещать данные, используя редактор. Мы выбрали имя для контейнера “ZSCHOL.TEST.DATASET” и ввели его в три поля, как показано на Рисунок 13. Бывают случаи, когда TSO и ISPF требуют, чтобы параметры были введены заглавными буквами. Поэтому хорошо, если вы будете всегда использовать заглавные буквы, когда работаете с TSO, ISPF и CICS. Даже если это не необходимо, трудно запомнить случаи, когда именно требуются заглавные буквы. Поэтому будем использовать их всегда.

Операционная система z/OS управляет мощной директорией функций для всех имен контейнеров и называет ее «каталог». Так же как в Windows или UNIX, имена контейнеров вводятся в каталог автоматически.

Операционная система z/OS очень гибкая система. Цена, которую мы за это платим, такая: пользователь должен принимать много больше решений, чем это требуется в других операционных системах. Понятно, что для наших упражнений TSO и ISPF – безнадежно сложны. Имейте в виду, что мы работаем с мощной большой вычислительной машиной, разработанной для работы в очень сложном системном окружении.

Определим размеры нашего контейнера. Мы решили измерять их в единицах MEGABYTE. Альтернативные единицы размеров могут быть также – tracks, cylinders, и другие. Мы определили максимальный размер в 2 MBYTE и разрешили увеличить этот размер еще на 1 MBYTE. В общем, имеем всего 3 MBYTE дискового пространства.

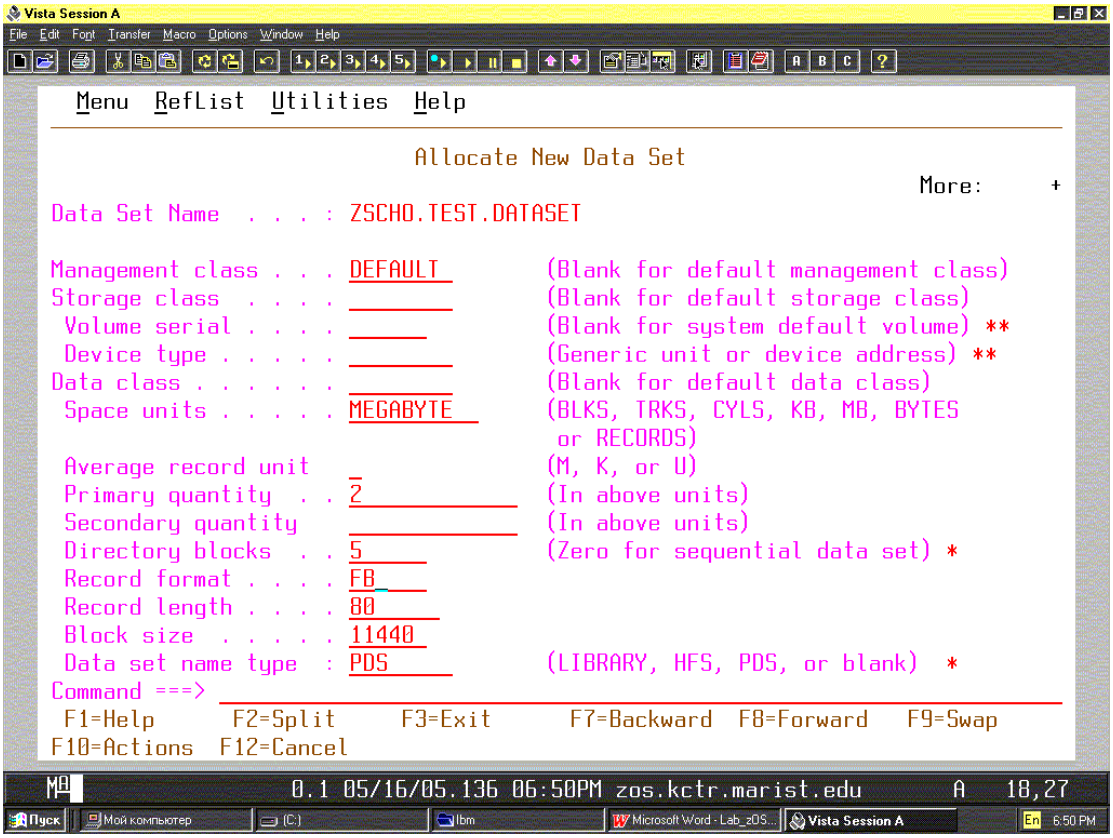


Рисунок 15. Окно определения нового набора данных с введенными параметрами

Существует также много типов контейнеров. Мы определили “Partitioned Data Set”. Для этого поставили цифру 5 в строке «Directory Blocks». Почему 5? – не спрашивайте! – просто для разных типов есть разные цифры. Например, «0» будет обозначать тип «Sequential Data Set».

Формат записи (Record) определили как «FB» (Fixed Blocks – фиксированные блоки), длина записи (Record Length) – “80 Bytes” и размер блока – 11440 записей.

Важно: размер блока должен быть получен умножением длины записи на определенное число, например: 80 x 143 = 11440. Почему 11440? Так подобрали, чтобы было удобно разместить контейнеры на физических дорожках диска (или по крайней мере, так это объясняют).

Остальные поля ISPF заполняются автоматически по умолчанию.

MANAGEMENT CLASS – MEGABYTE

Maximum size – 2

Directory blocks – 5

Record format – FB (for Fixed Blocks)

Record length – 80 Bytes

Block size – 11440 records

Введите параметры нового набора данных (контейнера) и нажмите Enter.

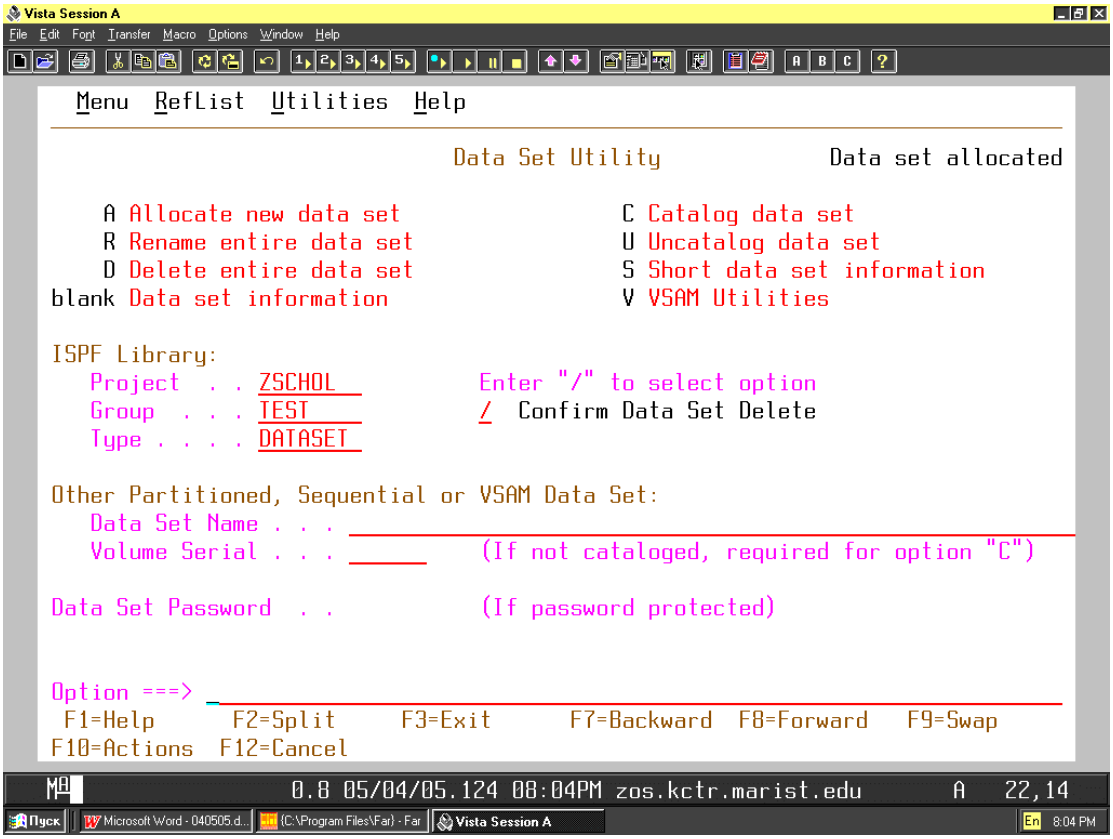


Рисунок 16. Новый набор данных создан (allocated)

В правом верхнем углу (Рисунок 16) вы увидите сообщение о том, что новый набор данных был определен (allocated). Наш контейнер способен теперь вобрать в себя несколько файлов. Нажмите F3 и вернитесь в панель “Utility Selection Panel” (см. Рисунок 17).

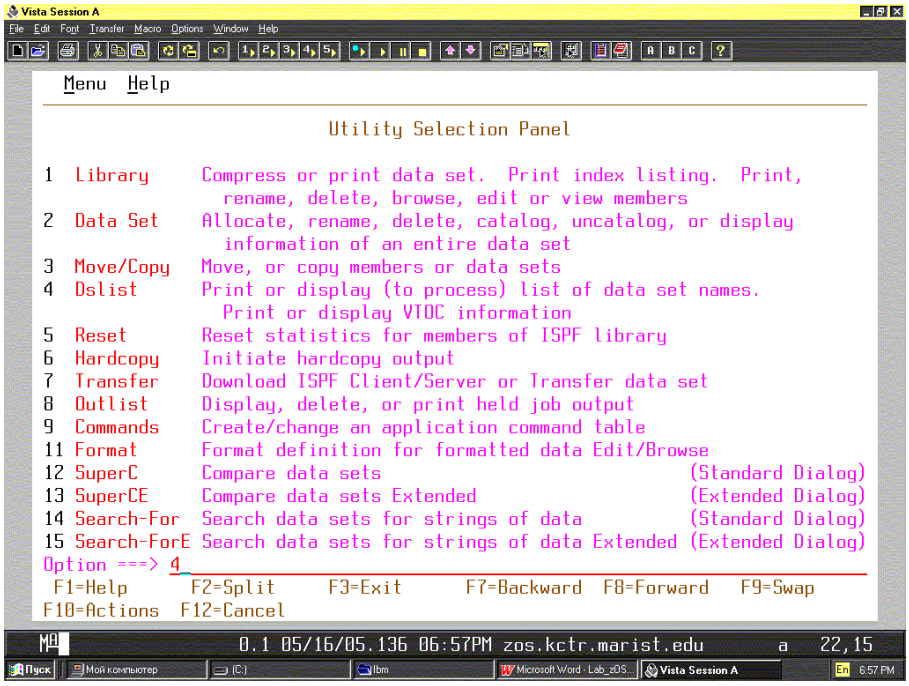


Рисунок 17. Панель “Utility Selection Panel”

Давайте проверим результат нашей работы. Это можно сделать с помощью функции Dslist (Data Set List), введя цифру 4 в командной строке и нажав Enter (см. Рисунок 18).

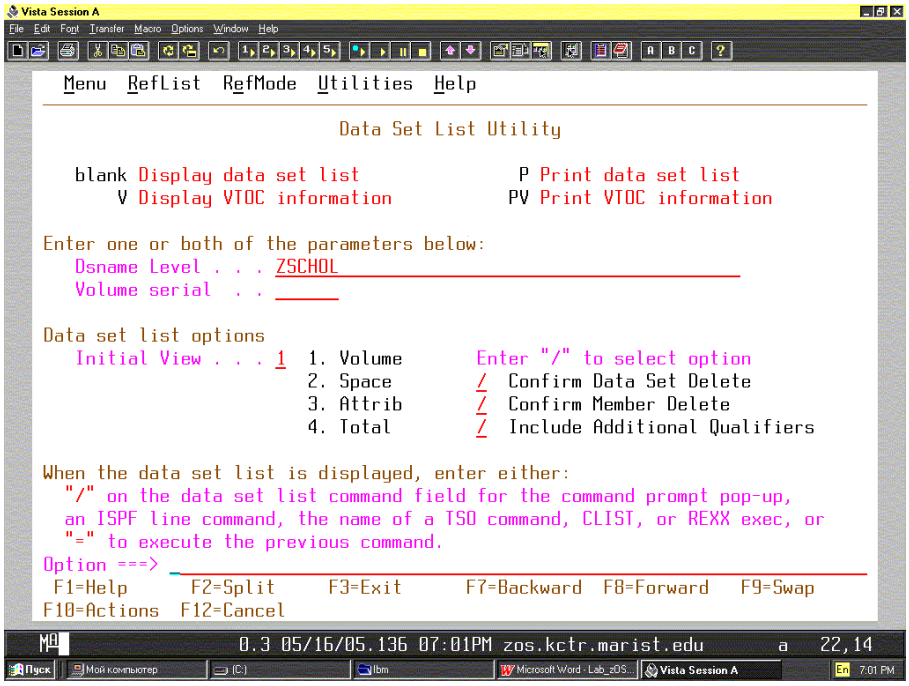


Рисунок 18. Окно “Data Set List Utility”

Окно “Data Set List Utility” предоставляет много возможностей, вначале просто нажмем Enter. Мы увидим созданный нами контейнер (см. Рисунок 19).

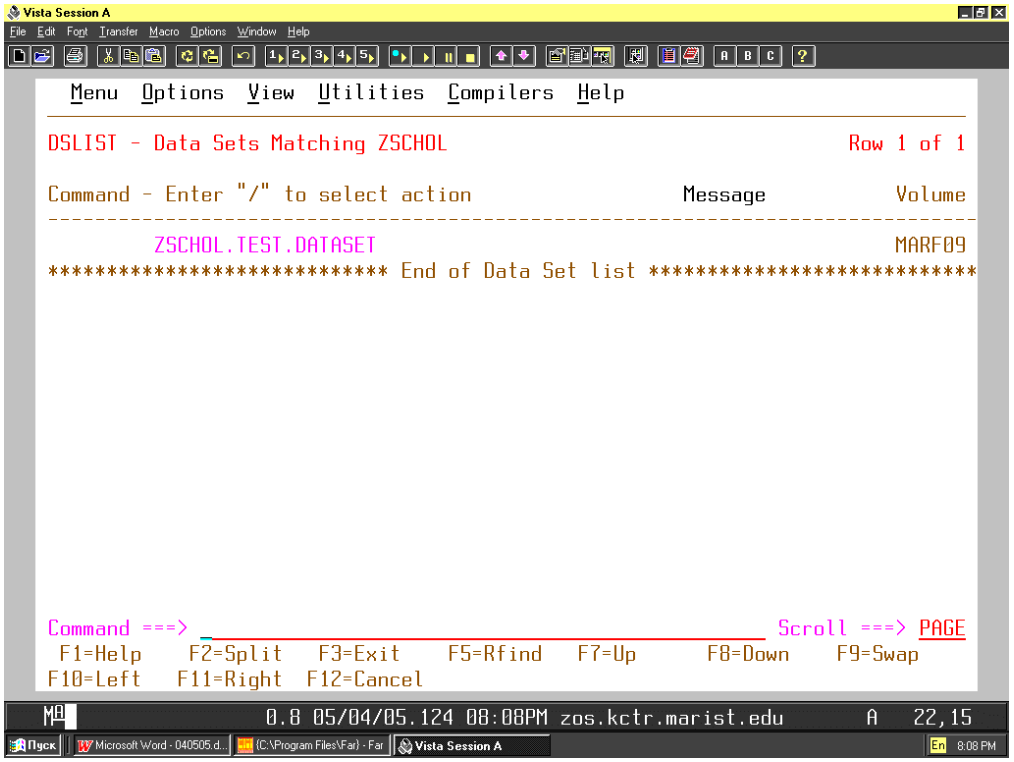


Рисунок 19. Результат работы – созданный контейнер для файлов

Повторите шаги и создайте еще два набора данных: ZSCHOL.TEST.CNTL и ZSCHOL.TEST.LOAD.

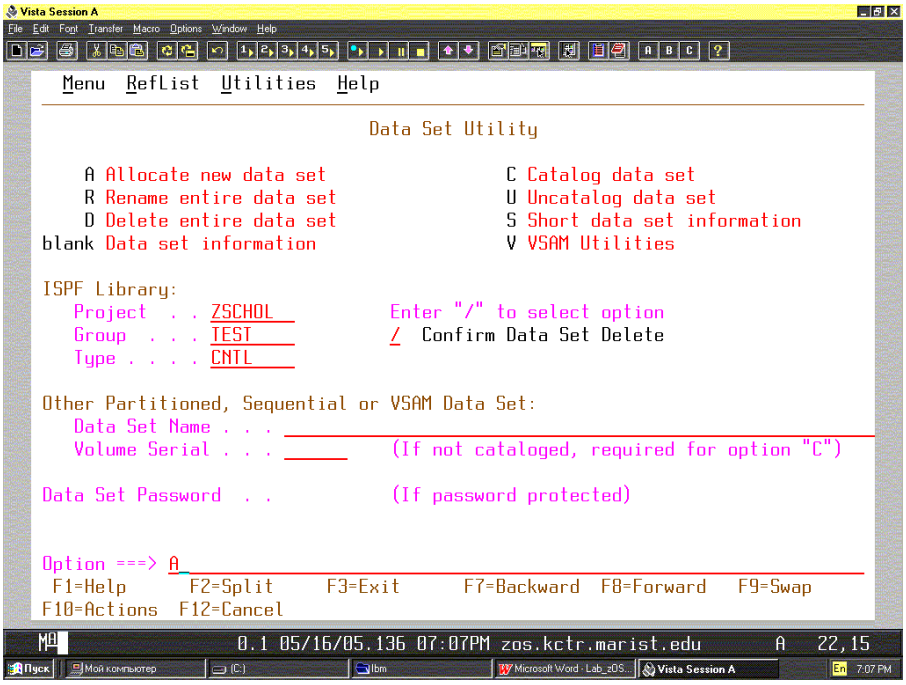


Рисунок 20. Создание контейнера ZSCHOL.TEST.CNTL

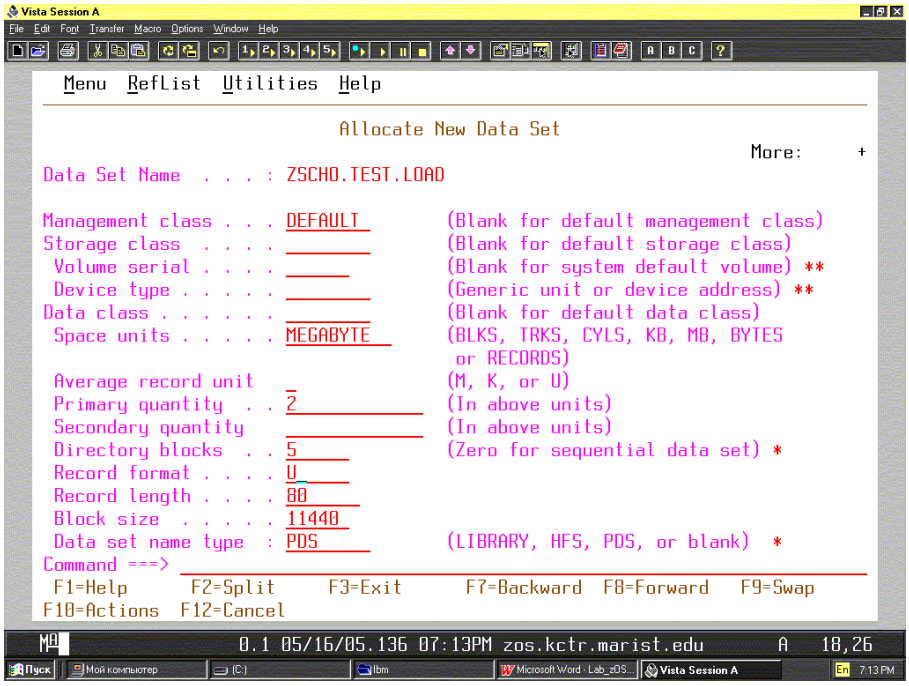


Рисунок 21. Окно определения нового контейнера (набора данных) ZSCHO.TEST.CNTL с введенными параметрами

Обратите внимание (см. Рисунок 21) на формат записи для контейнера, в котором будут храниться выполняемые программы в машинных кодах, ZSCHOL.TEST.LOAD! ОН ДОЛЖЕН БЫТЬ “U” – Undefined – неопределен.

После создания еще двух контейнеров ZSCHOL.TEST.LOAD и ZSCHOL.TEST.CNTL результат вашей работы будет выглядеть как показано на Рисунок 22.

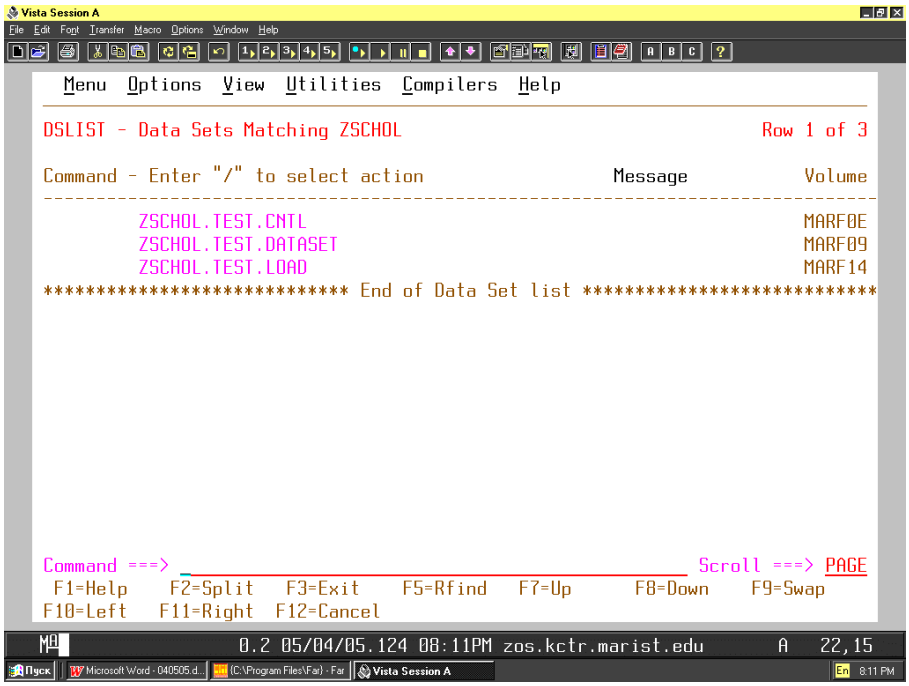


Рисунок 22. Результат запроса DSLIST

4. Выход из операционной системы z/OS (z/OS Logoff)

Нажмите несколько раз клавишу F3, чтобы вернуться в основное окно ISPF. Очередное нажатие клавишы F3 приведет вас в окно «Specify Disposition of Data Set». Система хочет знать, что вы собираетесь сделать с теми наборами данных, которые вы создали, и предоставляет вам на выбор список действий (Рисунок23). Выберите «3».

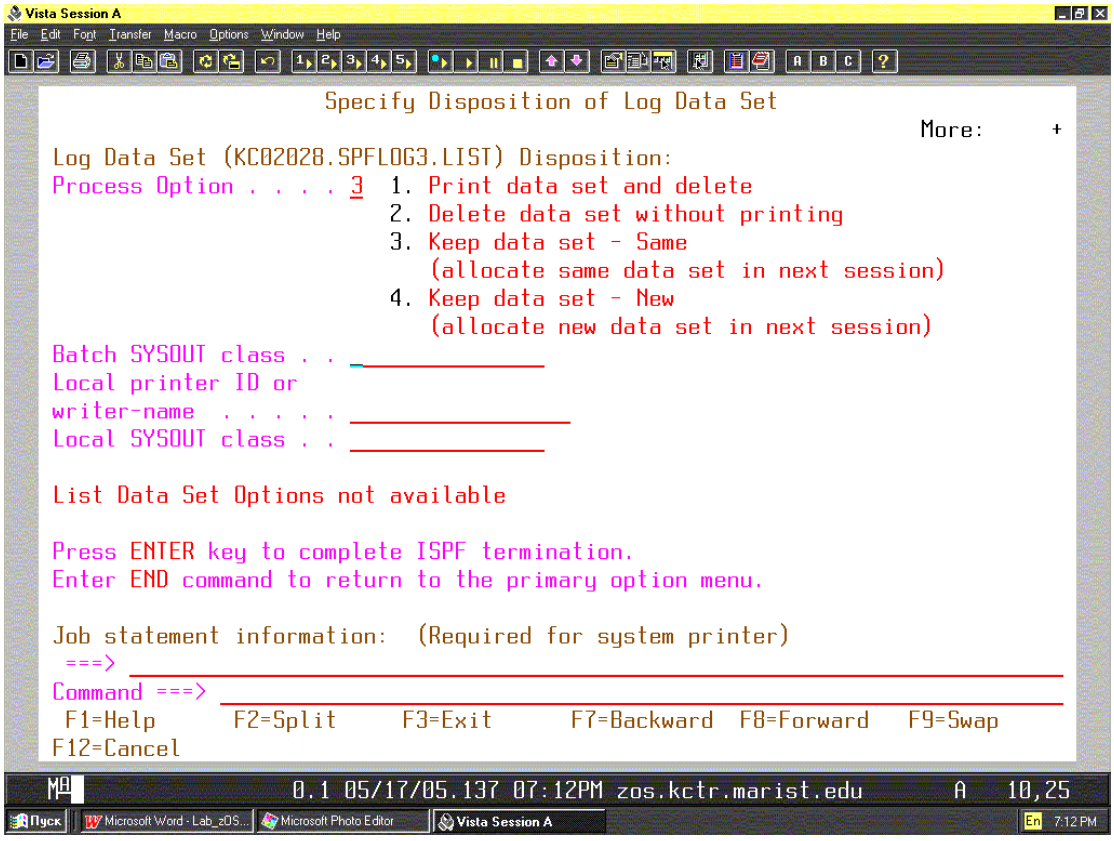


Рисунок 23. Окно «Specify Disposition of Data Set»

Сохраните данные. Появится сообщение TSO о том, что созданный набор данных будет доступен для работы при следующем вашем обращении к системе (см. рисунок 24). При этом набор данных KC02028.SPFLOG1.LIST, на который ссылается система, был создан ею автоматически.

Здесь слово «READY» означает ожидание команды от пользователя. Введите слово «LOGOFF» и нажмите Enter.

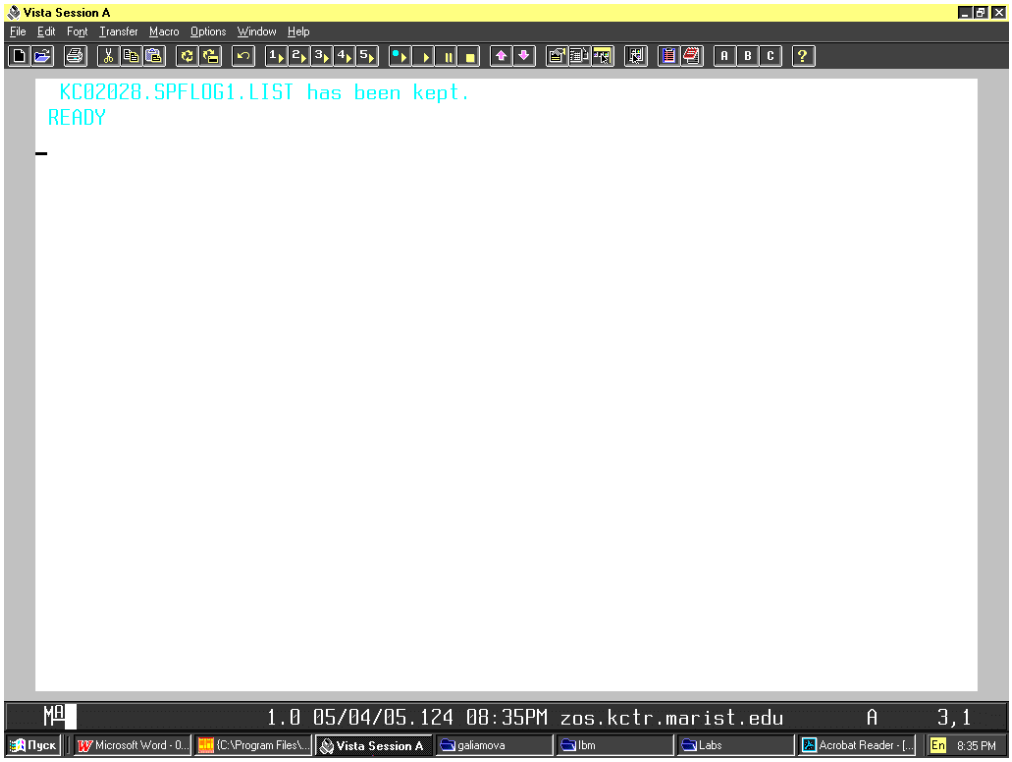


Рисунок 24. Сообщение TSO

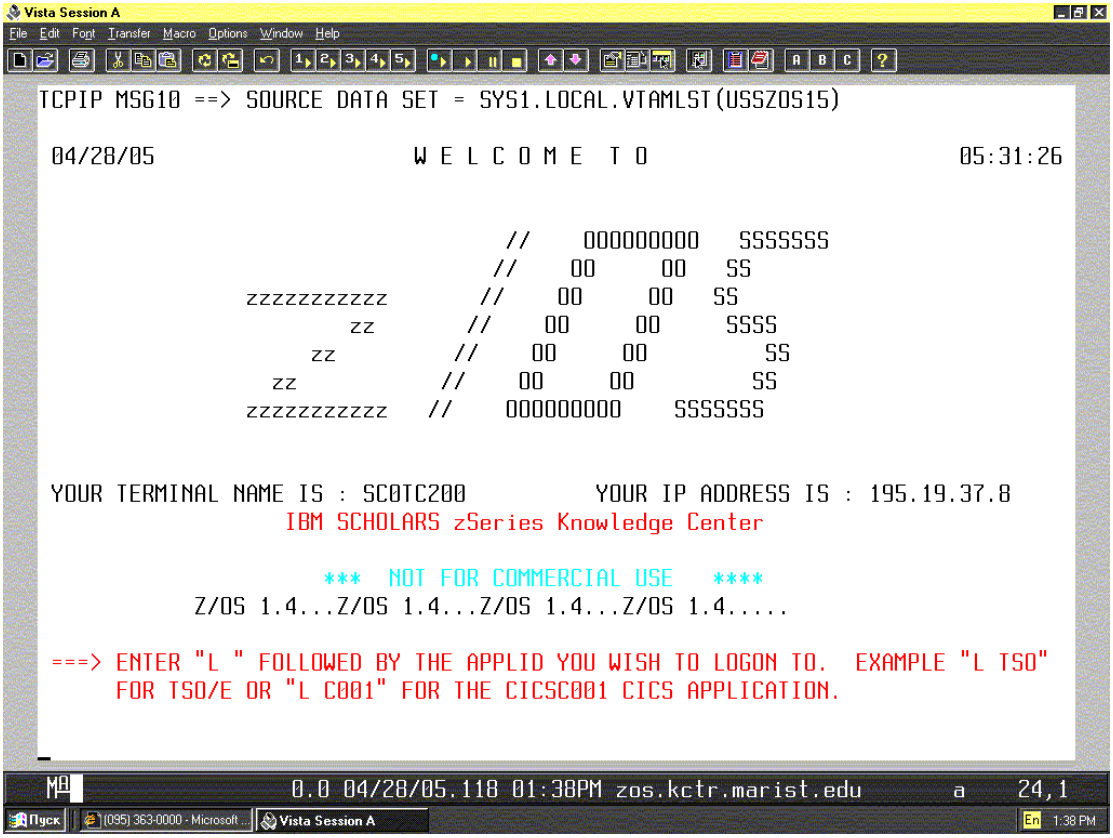


Рисунок 25. Завершение работы в TSO

Эмулятор 3270 вывел вас опять во входное окно операционной системы z/OS, ожидающее от вас команд.

На этом вы можете закончить сессию связи с виртуальной средой большой вычислительной машины MAINFRAME.

ПОЗДРАВЛЯЮ ВАС! ВЫ СДЕЛАЛИ ПЕРВЫЙ ШАГ В НОВУЮ СРЕДУ!

5. Контрольные вопросы

1. Какие способы обращения к операционной среде z/OS вы знаете?
2. Что такое эмулятор 3270?
3. Что такое TSO?
4. Что такое ISPF?
5. Перечислите основные функции ISPF.
6. Какую структуру имеют подсистемы и функции ISPF?
7. Перечислите последовательность шагов по созданию набора данных.
8. Какие типы наборов данных вы знаете?
9. Что такое «каталог»?
10. Какие поля характеристик набора данных заполняются операционной системой автоматически, и в каких случаях,
11. Какая функция ISPF отвечает за просмотр наборов данных?
12. Как выйти из режима ISPF?

Лабораторная работа № 2

Редактирование программ в среде Mainframe с использованием редактора ISPF

Методические рекомендации по курсу
«ОПЕРАЦИОННАЯ СИСТЕМА Z/OS и ВИРТУАЛЬНАЯ СРЕДА MAINFRAME»

Целями лабораторной работы являются:

- освоение навыков редактирование набора данных с использованием интерфейса редактирования ISPF;
- создание простой исходной программы на C/C++ в редакторе ISPF;
- компиляция ее с помощью скрипта JCL,
- выполнение созданной программы, вызвав программу из редактора ISPF или используя команды TSO.

Выполнение лабораторной работы

Внимание! В этом материале для создания файлов используется пользовательский ID KC02028. Вы должны будете использовать ваш собственный ID, выданный вам преподавателем.

1. Вход в операционную систему z/OS

Войдите в режим TSO и вызовите окно интерфейса ISPF (см. рисунок 1).

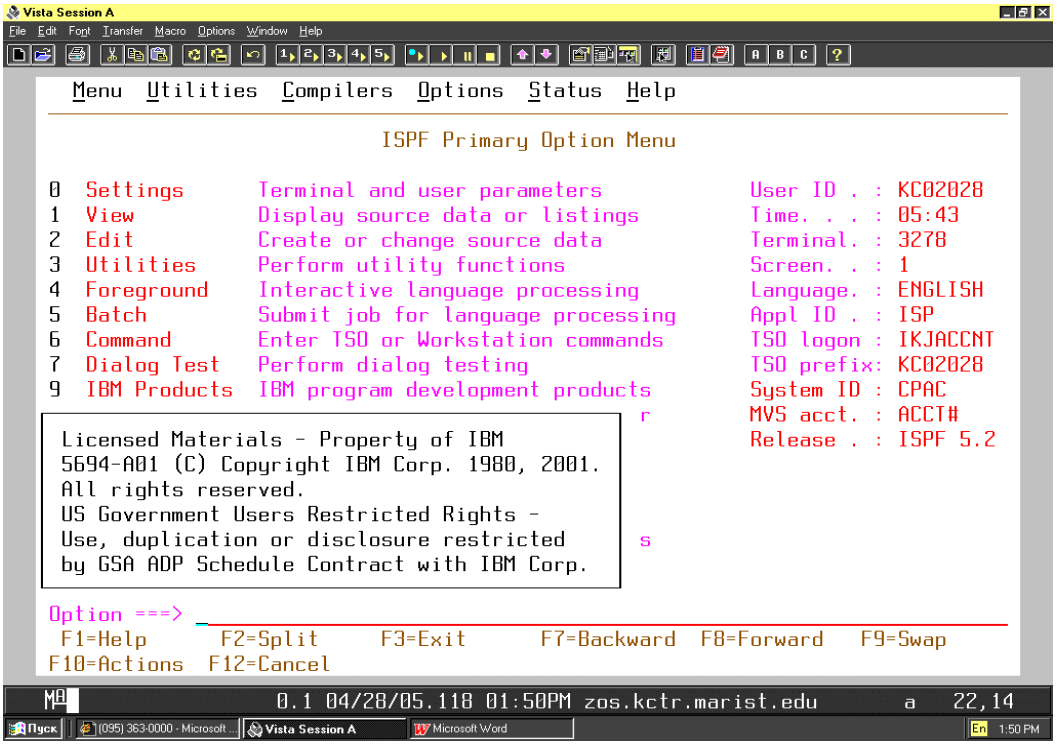


Рисунок 1. Начальное окно интерфейса ISPF

2. Создание среды разработки

Вам необходимо создать дополнительные контейнеры для хранения исходных кодов вашей программы на языке C. Мы могли бы использовать тот же набор данных KC02028.TEST.DATASET, который создали в лабораторной работе №1. Однако давайте оставим его про запас для экспериментального использования.

Поэтому, используя те же процедуры, которые вы освоили в лабораторной работе №1, создайте и поместите новый набор данных KC02028.TEST.C. Среда разработки теперь будет состоять из 3-х наборов данных:

- KC02028.TEST.C – для хранения программ в исходных кодах;
- KC02028.TEST.CNTL – для хранения компилированных программ;
- KC02028.TEST.LOAD – для хранения программ в машинных кодах (выполняемых).

3. Редактирование данных в режиме ISPF Editor (кодирование на языке C)

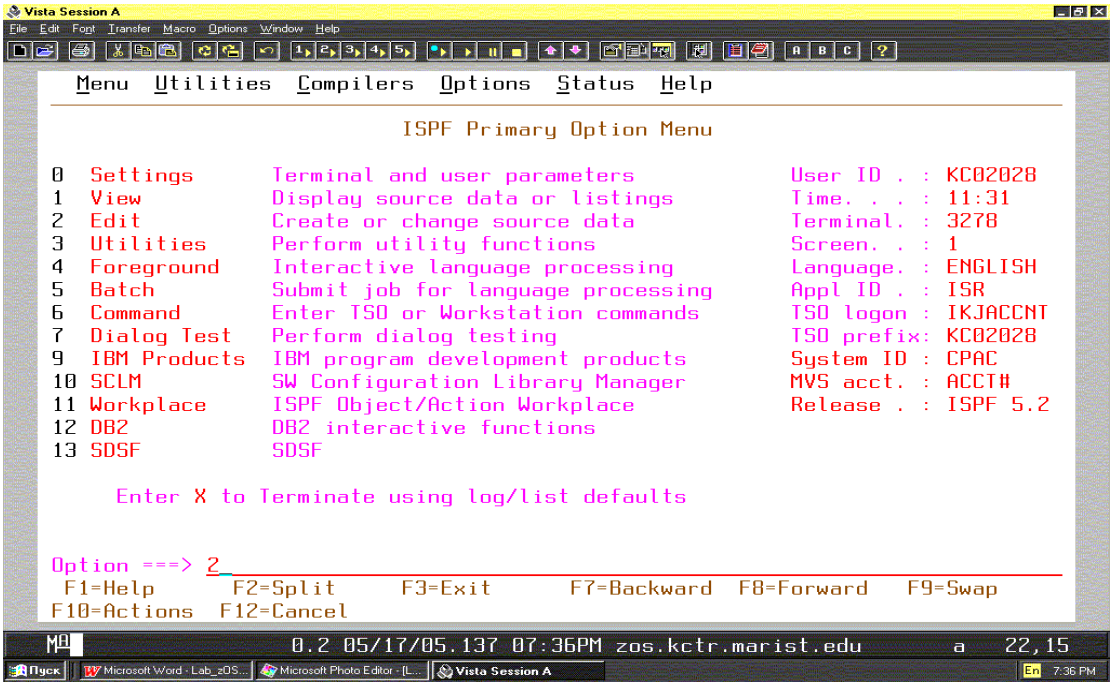


Рисунок 2. Вызов редактора ISPF

Вызвать режим редактирования вы можете из основного экрана **ISPF**, введя цифру «2».

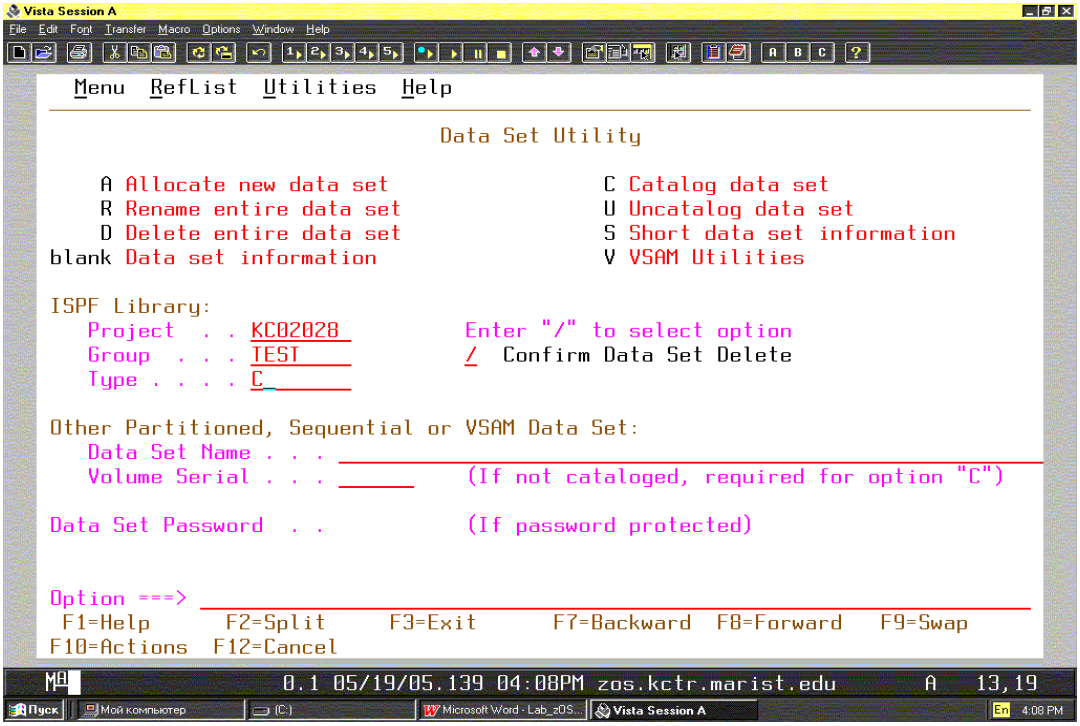


Рисунок 3.

Давайте создадим программу на C, используя редактор ISPF. В окне подпрограммы редактирования ISPF вы должны ввести имя файла (member), который будет хранить исходный код программы. Исходный код программы будет храниться в контейнере

KC02028.TEST.C, которому мы присвоили формат Partitioned Dataset. Этот файл должен иметь имя. Назовем его V1 (Version 1). Полное имя файла KC02028.TEST.C(V1). Введите эти величины в соответствующих полях, как показано на рисунке 4 и нажмите Enter.

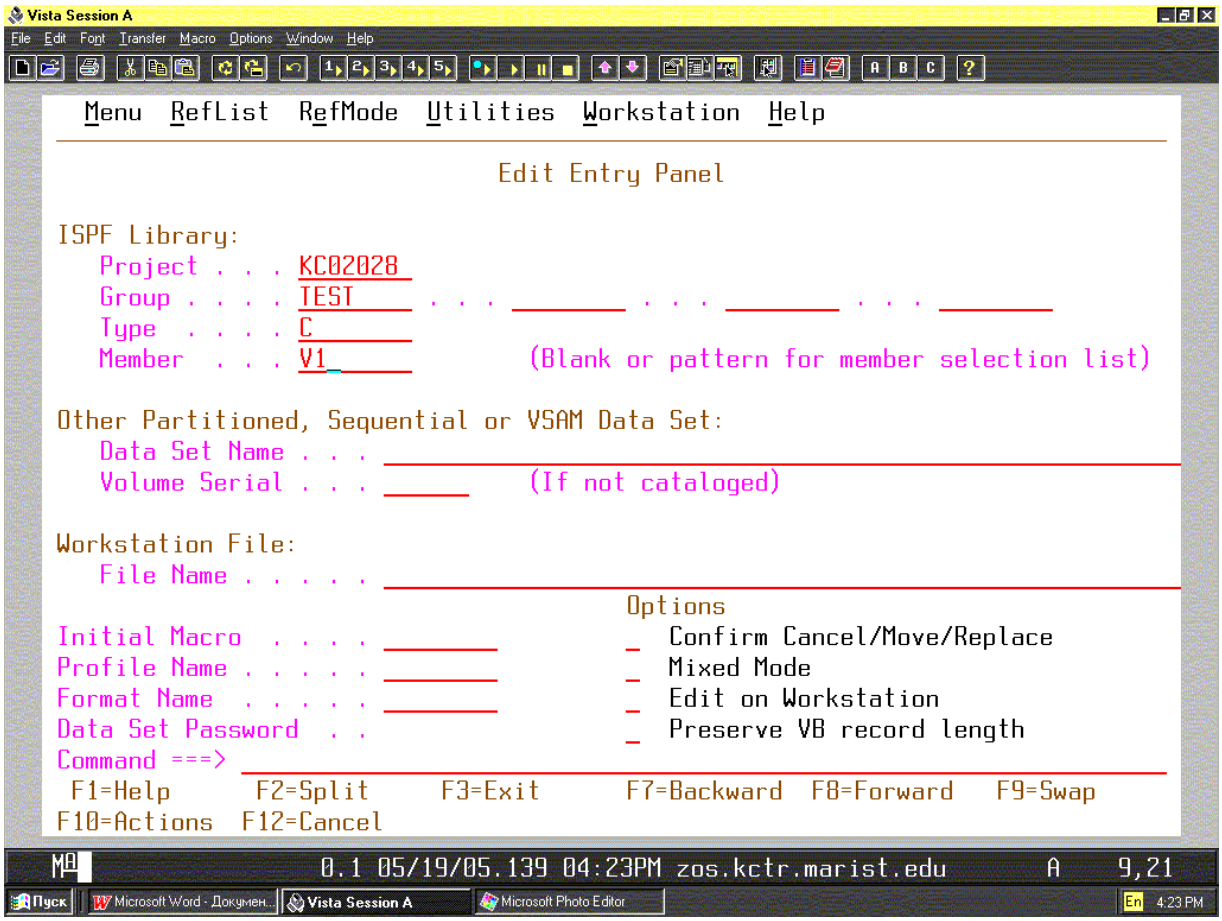


Рисунок 4.

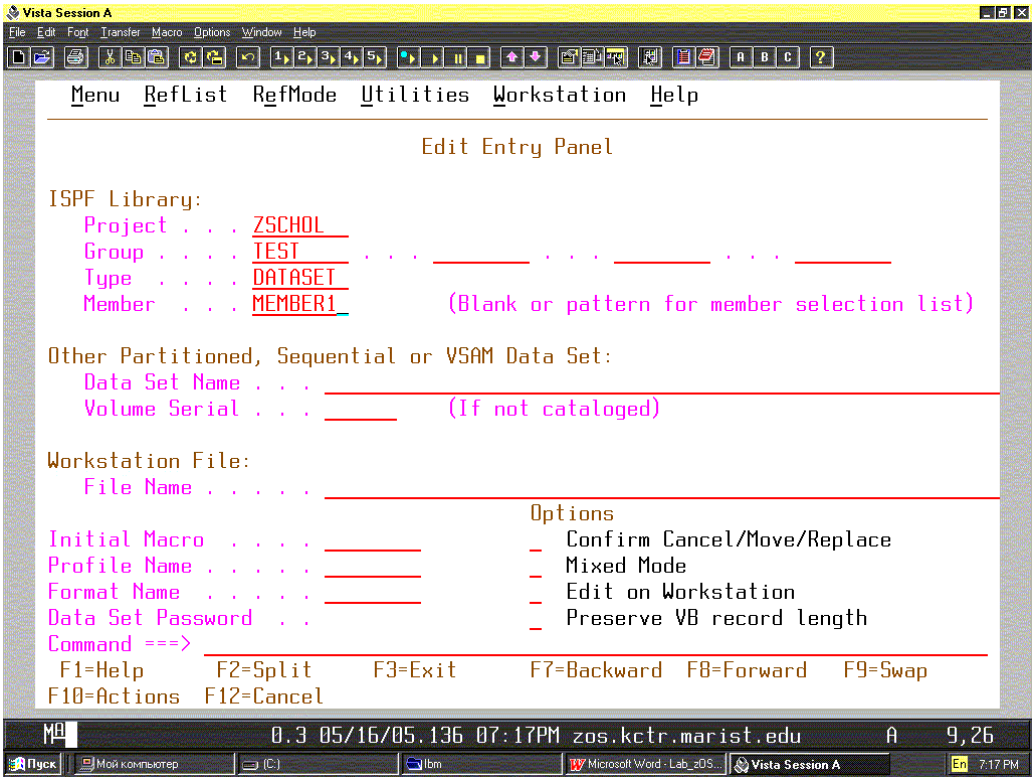


Рисунок 5. Окно входа в режим редактирования Edit Entry Panel

Первый раз после распределения памяти для нового контейнера (набора данных) экран редактирования будет иметь вид, показанный на рисунке 6.

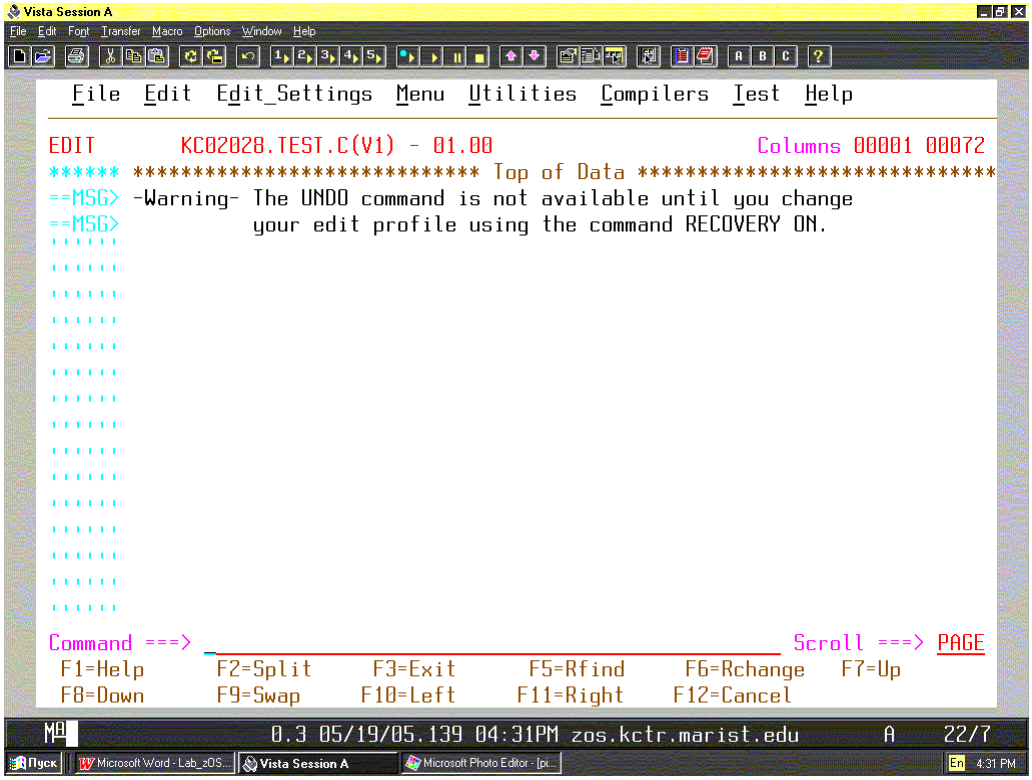


Рисунок 6. Пустое окно редактирования ISPF

Прочитайте внимательно сообщение в окне редактирования (и всегда внимательно читайте все сообщения системы!), начинающееся словом «Warning!», говорящее о том, что редактирование возможно только после ввода команды “RECOVERY ON” (рисунок 6).

После ввода этой команды вы вновь окажетесь в окне Edit Entry Panel, при этом в правом верхнем углу увидите надпись «Member V2 saved».

Используйте клавиши букв и цифр, чтобы ввести текст и не пользуйтесь никакими управляющими клавишами, кроме клавиши Del. Курсор можно двигать клавишами со стрелками.

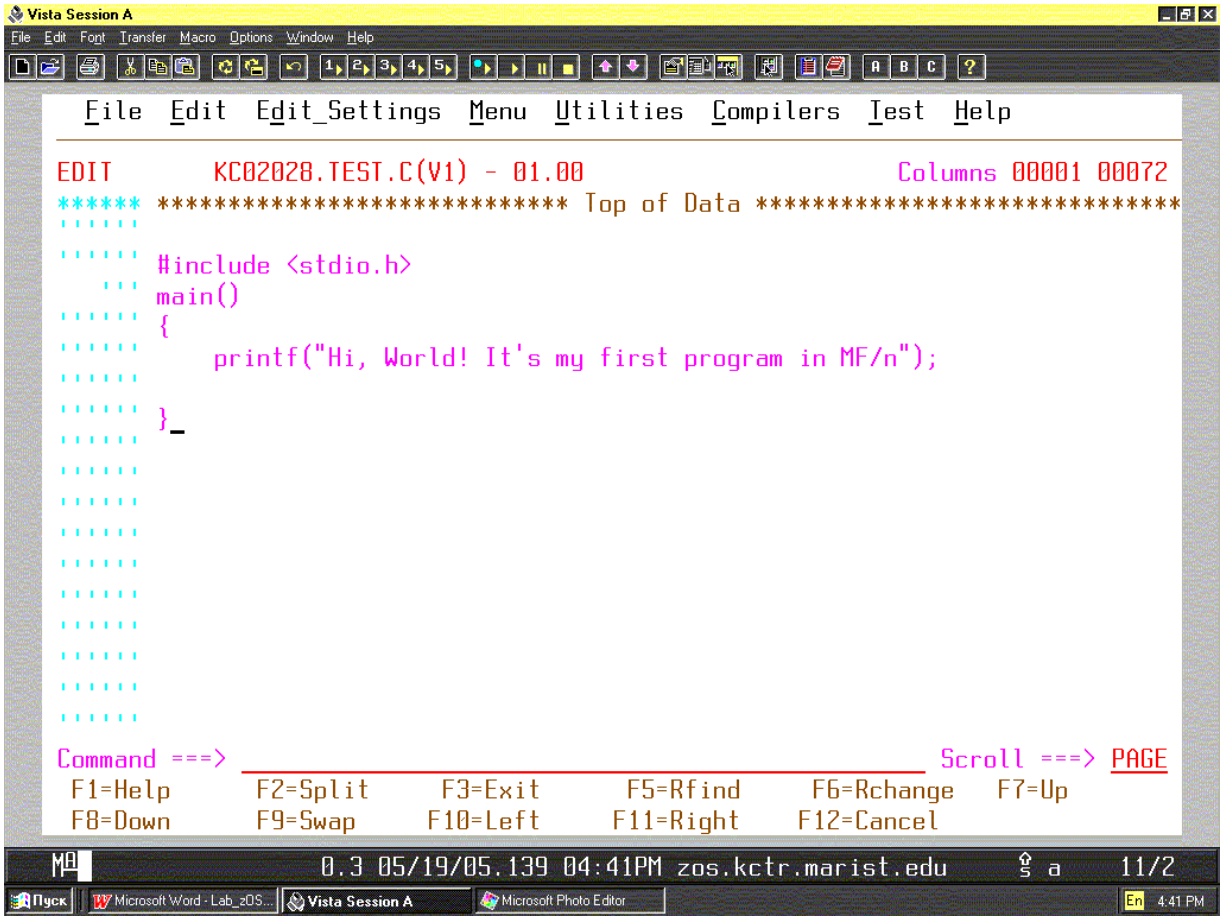


Рисунок 7. Окно подпрограммы редактирования ISPF с набранной программой на C

Наберите программу, как показано на рисунке 7. Разместите символы точно как показано. Программа выведет текстовую строку на ваш экран. Затем нажмите F3 для возврата в главное окно. Наша программа будет автоматически сохранена.

Внимание:

Все ключевые слова в программных кодах должны быть написаны **незаглавными** буквами. Может так случиться, что редактор ISPF автоматически конвертирует эти буквы в заглавные. Если это произойдет, введите в командной строке экрана, показанного на рисунке 7, команду TSO «CAPS OFF», затем нажмите Enter.

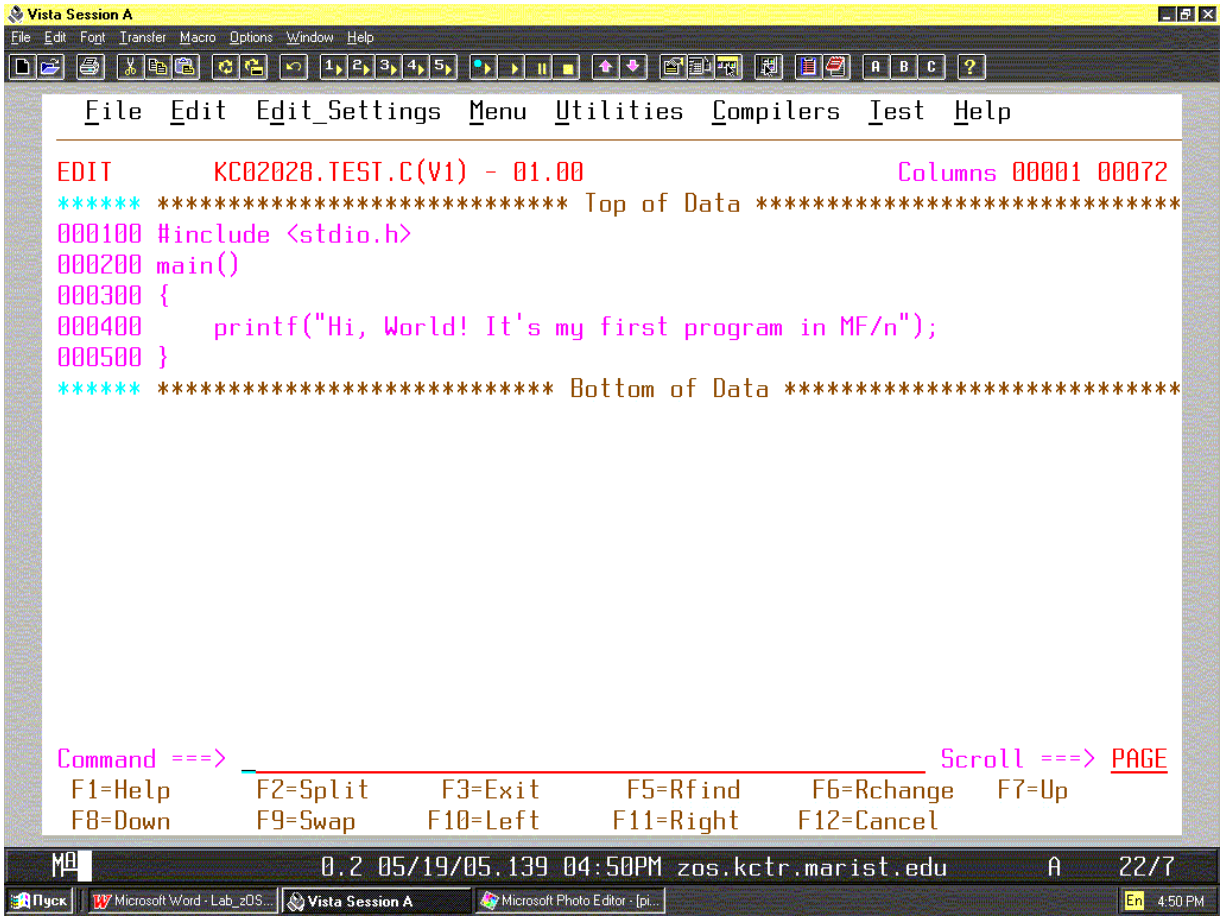


Рисунок 8. Окно подпрограммы редактирования ISPF с набранной программой на C и с номерами строк

В правом верхнем углу на экране в окне входа в подпрограмму редактирования ISPF после написания исходных кодов появится надпись о том, что наш файл был сохранен. Вызываем опять редактор ISPF, введя имя файла V1 в поле “Member” и нажав Enter. На рисунке 7 редактор добавил нумерацию строк к нашей программе.

Для освоения функций редактирования в редакторе ISPF вы должны получить у преподавателя учебный курс в электронном виде (на CD-ROMe), изучить его и пройти автоматическое тестирование по этому курсу.

Создайте еще один файл ZSCHOL.TEST.CNTL(V1). Нажмите F3.

3. Создание JCL-скрипта и выполнение его

В среде UNIX мы используем “Make File”, чтобы транслировать C-программу. В среде TSO мы используем “Compile Script”. Мы храним этот скрипт, как файл (member) в контейнере формата partitioned dataset ZSCHOL.TEST.CNTL(V1). Скрипт скомпилирует нашу программу в исходных кодах, свяжет ее (link it) и сохранит ее как выполняемый файл (в машинных кодах) в файле контейнера KC02028.TEST.LOAD. Для этого будет использоваться программа JCL (z/OS Job Control Language).

Приготовьтесь испытать шок! Язык JCL выглядит ужасно, такого вы еще никогда не видели. Однако, он очень мощный, на удивление легкий для изучения и использования. Я думаю, вы пришлете свои комплименты относительно языка JCL в IBM, а потом и начнете его использовать.

Операционная система z/OS поддерживает и другие языки скриптов (приложений): широко используется также REXX. REXX похож на Perl и доступен на многих других платформах, кроме z/OS. Однако, даже если вам трудно в это поверить, язык JCL – это язык скриптов по определению (script language of choice).

Введите “CNTL” в поле “Type” и “V1” в поле “Member”. Осталось нажать Enter.

4. Выход из операционной системы z/OS (z/OS Logoff)

Нажмите несколько раз клавишу F3, чтобы вернуться в основное окно ISPF. Очередное нажатие клавишы F3 приведет вас в окно «Specify Disposition of Data Set». Система хочет знать, что вы собираетесь сделать с теми наборами данных, которые вы создали, и предоставляет вам на выбор список действий (рисунок 9). Выберите «3».

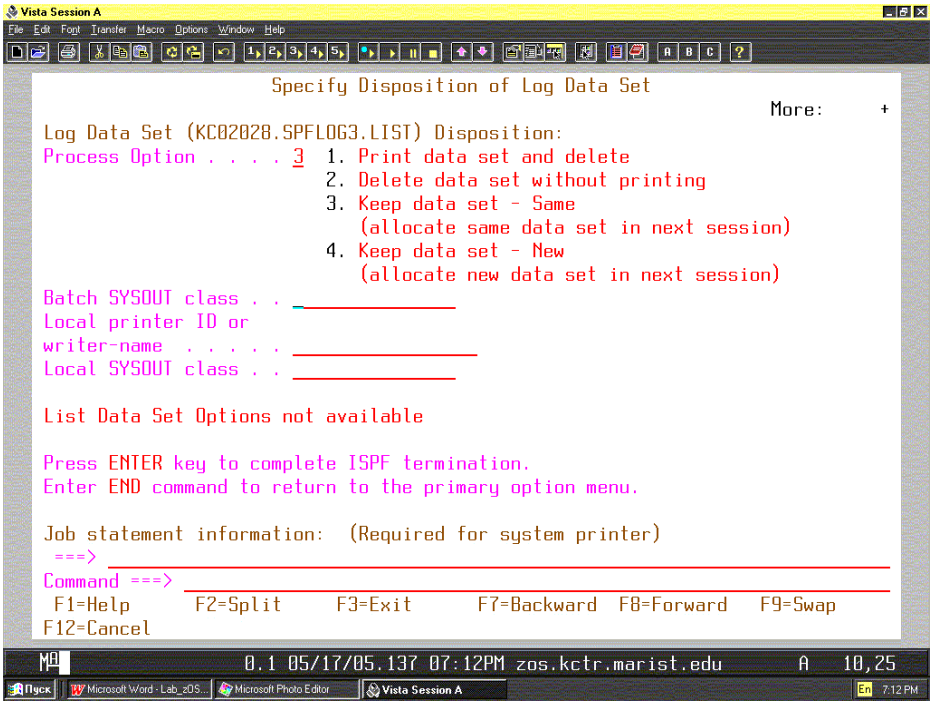


Рисунок 9. Окно «Specify Disposition of Data Set»

Сообщение TSO о том, что созданный набор данных будет доступен для работы при следующем вашем обращении к системе. При этом набор данных KC02028.SPFLOG1.LIST, на который ссылается система, был создан ею автоматически. Здесь слово «READY» означает ожидание команды от пользователя. Введите слово «LOGOFF» и нажмите Enter.

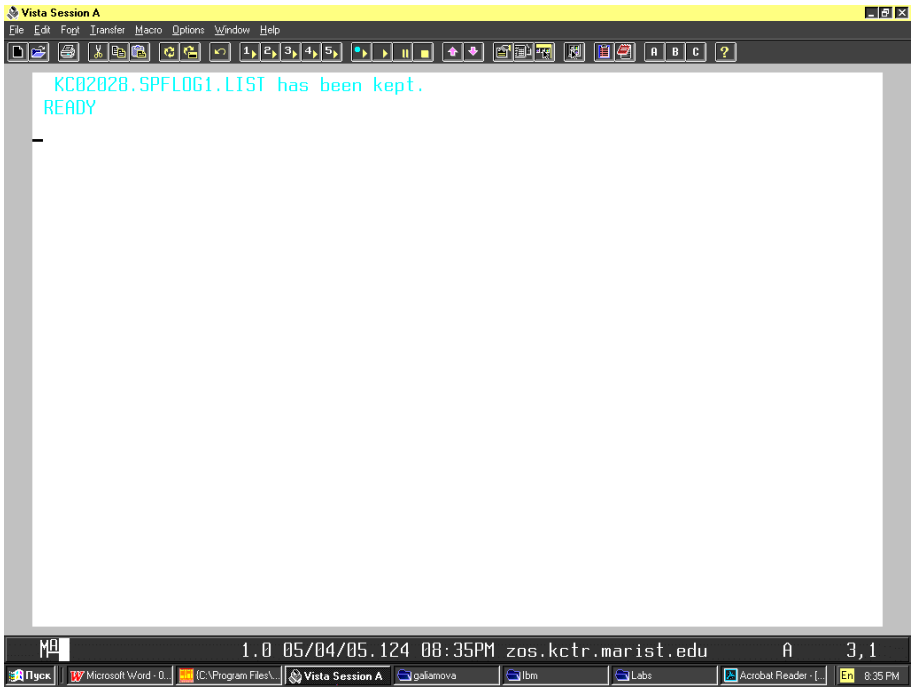


Рисунок 9. Окно ожидания команды TSO

Рис. 31. Окно завершения работы в ISPF

Эмулятор 3270 вывел вас опять во входное окно операционной системы z/OS, ожидающее от вас команд.

На этом вы можете закончить сессию связи с виртуальной средой большой вычислительной машины MAINFRAME.

Лабораторная работа №3.

Работа с системными сервисами UNIX операционной системы z/OS

Цель лабораторной работы - знакомство с основами использования системных сервисов операционной системы большой вычислительной машины z/OS UNIX, а также с командной оболочкой SHELL.

Задачами лабораторной работы являются:

1. Освоение структуры иерархической файловой системы HFS (Hierarchical File System) операционной системы z/OS и основных команд для работы с ней.
2. Изучение командной оболочки shell.
3. Изучение скриптового языка Awk.
4. Создание простой С программы.

1. Теоретическая часть

1.1. Основы z/OS UNIX

В z/OS Unix реализовано два открытых системных интерфейса:

- интерфейс системных вызовов API и
- интерактивный интерфейс пользователя.

Интерфейс системных вызовов (API) позволяет запускать стандартные Unix приложения, написанные на C, в z/OS. Интерактивный интерфейс пользователя (оболочка shell) позволяет выполнять Unix команды, утилиты и скрипты в z/OS.

Ядро z/OS UNIX интегрировано в базовую управляющую программу z/OS и служит для реализации функций интерфейса системных вызовов (API UNIX), связанных с управлением процессами, файловой системой HFS и коммуникациями. Активизируется при загрузке z/OS и работает в собственном адресном пространстве MVS.

Основная единица работы в операционной системе UNIX – процесс, он соответствует находящейся в стадии выполнения программе со всеми выделенными ей ресурсами. Процессы выполняются исключительно в адресных пространствах MVS. При использовании системного вызова fork() всегда создается новое адресное пространство, являющееся копией родительского. При использовании системного вызова spawn() может быть создано новое адресное пространство, или запущена новая задача внутри родительского AS.

Процессы: бывают системные (например, демоны - работающие в фоновом режиме и предназначенные для поддержки вспомогательных системных сервисов - вывод на печать, электронная почта, запуск программ по расписанию и т. д.) и пользовательские.

Каждый процесс имеет уникальный идентификатор PID и может по своей инициативе порождать новые (дочерние) процессы с помощью системных вызовов fork() и spawn(). Также каждый процесс имеет родительский процесс, который определяется идентификатором родительского процесса PPID (см. рисунок 1).

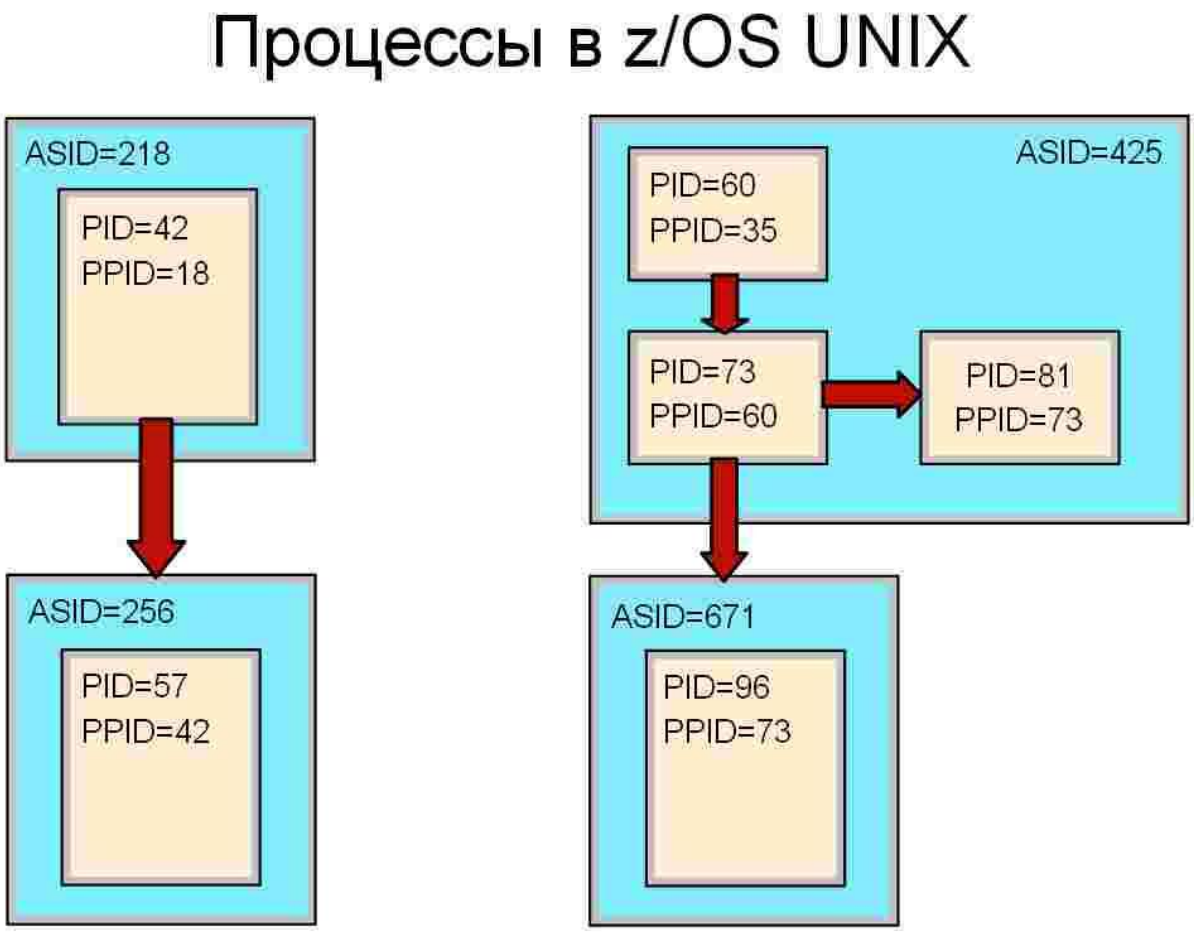


Рисунок 1 - Процессы в z/OS UNIX (этот рисунок и все последующие взяты из учебных материалов компании IBM).

Механизм выполнения приложения UNIX в z/OS показан на рисунке 2. Для ядра z/OS UNIX во время инициализации системы создается отдельное адресное пространство (OMVS), функционирование которого зависит от настроек в разделе BPXPRMxx системного реестра SYS1.PARMLIB. Одновременно с этим создается файловая система HFS (подробнее о файловой системе см. в п.2) и создается адресное пространство BPXOINIT, выполняющее процесс прародитель (PID = 1) для всех процессов. В первую очередь BPXOINIT порождает необходимые системные процессы UNIX.

Рисунок 2 - Механизм выполнения приложения UNIX в z/OS.

Новые адресные пространства для приложений UNIX создаются по запросу ядра к менеджеру управления рабочей нагрузкой WLM. WLM использует для создания нового адресного пространства специальную STC-процедуру BPXAS.

Как правило, для каждого интерактивного пользовательского сеанса командные интерпретаторы shell запускаются в отдельных адресных пространствах. Команды и утилиты пользователя могут запускаться как внутри адресного пространства shell, так и в новых адресных пространствах.

Процесс-демон INETD обеспечивает доступ к shell для удаленных пользователей в TCP/IP-сети с использованием протоколов telnet и rlogin.

1.2. Иерархическая файловая система HFS

Файлы UNIX обрабатываются системой как простая совокупность байтов, без деления на логические записи. Имена файлов могут содержать до 255 алфавитно-цифровых символов, при этом различают прописные и строчные буквы.

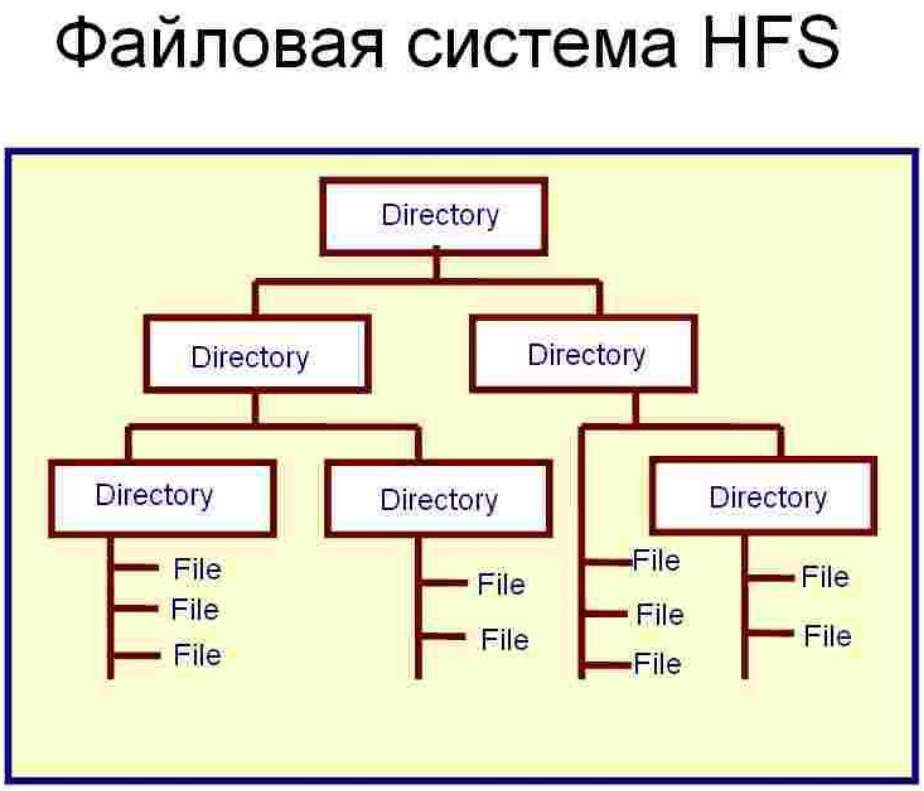


Рисунок 3 – Иерархическая древовидная структура файловой системы HFS.

В UNIX выделяют следующие типы файлов:

- обычные - файлы общего назначения, используемые для хранения программ и данных любого типа;
- каталоги - служат для размещения справочной информации о размещении файлов, принадлежащих данному каталогу;
- устройства - ассоциируются с устройствами ввода-вывода;
- символические ссылки - содержат ссылки на другие файлы;
- именованные каналы - служат для обмена данными между процессами;
- сокеты - служат для реализации сетевого взаимодействия.

Для работы с файлами и каталогами в z/OS UNIX используется командная оболочка shell. С помощью специальных команд вы можете создавать и удалять каталоги и файлы, изменять права доступа, работать с жесткими и символическими ссылками.

Файлы группируются по каталогам, которые представляют собой иерархическую древовидную структуру (см. рисунок 3). Вершиной дерева и единой точкой входа в файловую систему является корневой каталог (/). Таким образом, у каждого файла существует **полный или абсолютный путь**, однозначно определяющий его местоположение в файловой системе, например:

- /u/user1/docs/abc,
- /u/user2/prg и т.п.

Наиболее важные системные программы, данные и конфигурационные файлы UNIX размещаются в специальных каталогах:

- /bin - команды и утилиты;
- /usr - файлы для поддержки решения пользовательских задач;
- /dev - специальные файлы устройств ввода-вывода;
- /etc - утилиты администрирования и конфигурационные файлы;
- /lib - включаемые библиотеки C/C++;
- /tmp - временные файлы;
- /var - сообщения и системные журналы;
- /samples - примеры программ и настроечных файлов.

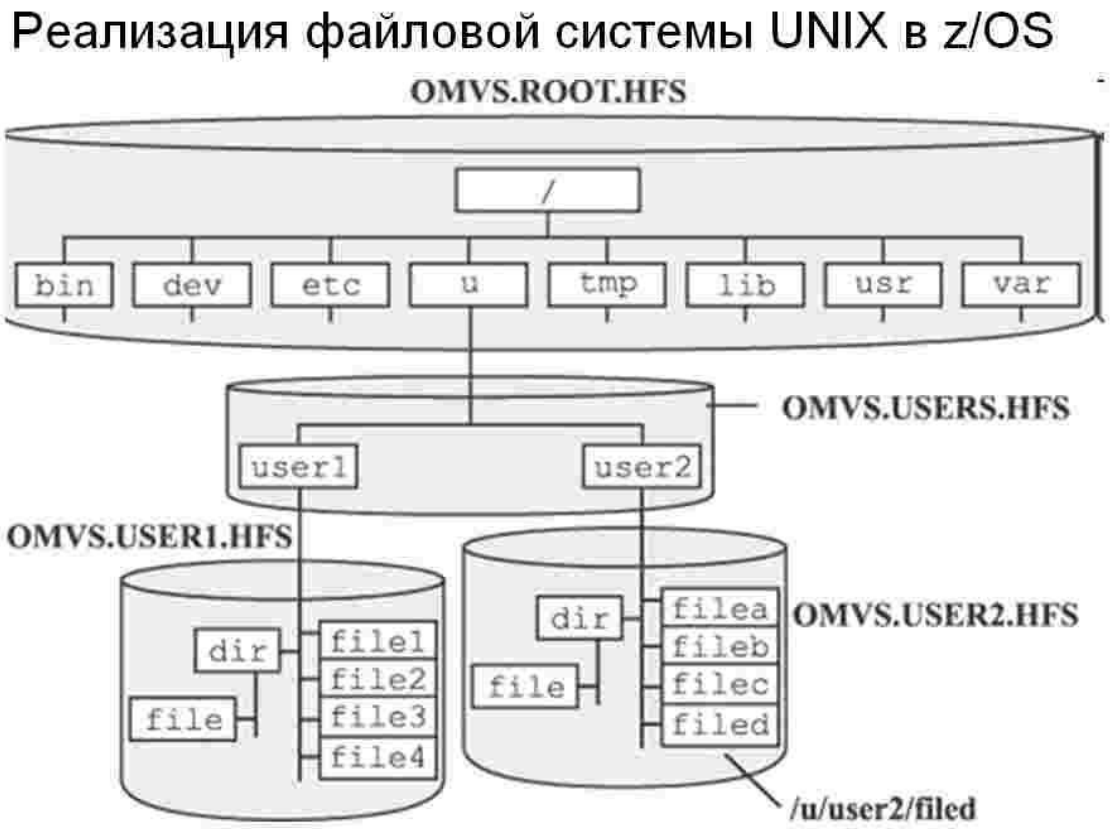


Рисунок 4 -

Все особенности иерархической файловой системы HFS поддерживаются системными сервисами UNIX в операционной системе большой ЭВМ - z/OS.

В операционной системе z/OS большой ЭВМ для размещения файлов UNIX и для реализации иерархической структуры доступа создаются специальные однотомные SMS-управляемые наборы данных, получившие название **наборов данных HFS** (см. рисунок 4). Каждый набор данных HFS содержит определенный сегмент файловой системы, точкой входа в который является один из каталогов.

Объединение сегментов HFS производится с помощью специальной операции "монтирования",

выполняемой на этапе инициализации системы или динамически. Первым всегда монтируется сегмент, содержащий корневой каталог файловой системы (/), к которому затем могут добавляться другие сегменты. Создание и управление наборами данных HFS осуществляется стандартным компонентом операционной системы z/OS - DFSMS (Data Facility Storage Management Subsystem) – подсистемой управления данными и внешней памятью.

1.3. Скриптовый язык awk

Язык Awk – это язык программирования для обработки информации, хранящейся в файле. Программы, написанные на языке Awk, позволяют:

- Выводить полностью или частично все данные, содержащиеся в файле.
- Выполнять вычисления с числовыми данными файла.
- Формировать отчеты на основе информации, хранящейся в файле.
- Проверять орфографию текста, определять частоту появления слов или букв и т. д.

Программист может объединять эти операции для решения более сложных задач. Язык Awk поддерживает большинство выражений современных языков программирования, такие как if-else, while и for циклы, вызовы функций и т. д. Реализация z/OS Awk основана на стандарте POSIX для Awk.

Как было сказано выше, Awk-программы работают с данными. Они могут обрабатывать данные, полученные от рабочей станции, или данные, выданные в результате выполнения команды (возможно, конвейера команд), но обычно они работают с файлами данных. Под файлами данных понимаются обычные текстовые файлы. Эти файлы содержат читаемый текст, например, слова, числа и символы пунктуации.

В качестве примера рассмотрим текстовый файл hobbies.

```
Jim     reading         15      100.00
Jim     bridge          4       10.00
Jim     role    playing         5 70.00
Linda   bridge  12      230.00
Linda   cartooning     5       75.00
Katie   jogging 14      120.00
Katie   reading 10      60.00
John    role playing    8       100.00
```

Каждая строка этого файла содержит имя, род занятий и количество часов в неделю, уделяемых этому занятию, а также расходы за год. Таким образом, файл данных awk представляет собой набор записей. Запись состоит из информационных полей, описывающих один предмет. Записи обычно разделены символом “переход но новую строку - enter”. Каждая новая строка представляет собой отдельную запись. Первая запись содержит 4 поля: Jim, reading, 1, 100.00.

Awk-программа имеет следующую структуру:

```
шаблон {действие}
шаблон {действие}
шаблон {действие}
шаблон {действие}
```

Каждая строка awk программы – это отдельная инструкция. Awk последовательно просматривает файл данных и выполняет инструкции в заданном программой порядке над каждой записью данного файла.

Пример Awk инструкции: $2 == “jogging” { print }

Выражение == означает проверку тождественности. Таким образом, если второе поле записи “jogging”, то выводится вся запись.

Для запуска Awk программы используется shell команда: awk ‘program’ datafile.

Название программы заключается в одинарные кавычки. Аргумент datafile указывает на обрабатываемый файл данных.

1.4. Режимы доступа пользователей к z/OS UNIX

Для пользователей z/OS UNIX поддерживается несколько различных вариантов интерактивного доступа к системным сервисам UNIX, как с помощью shell, так и некоторыми другими способами, представленными на рисунке 5.

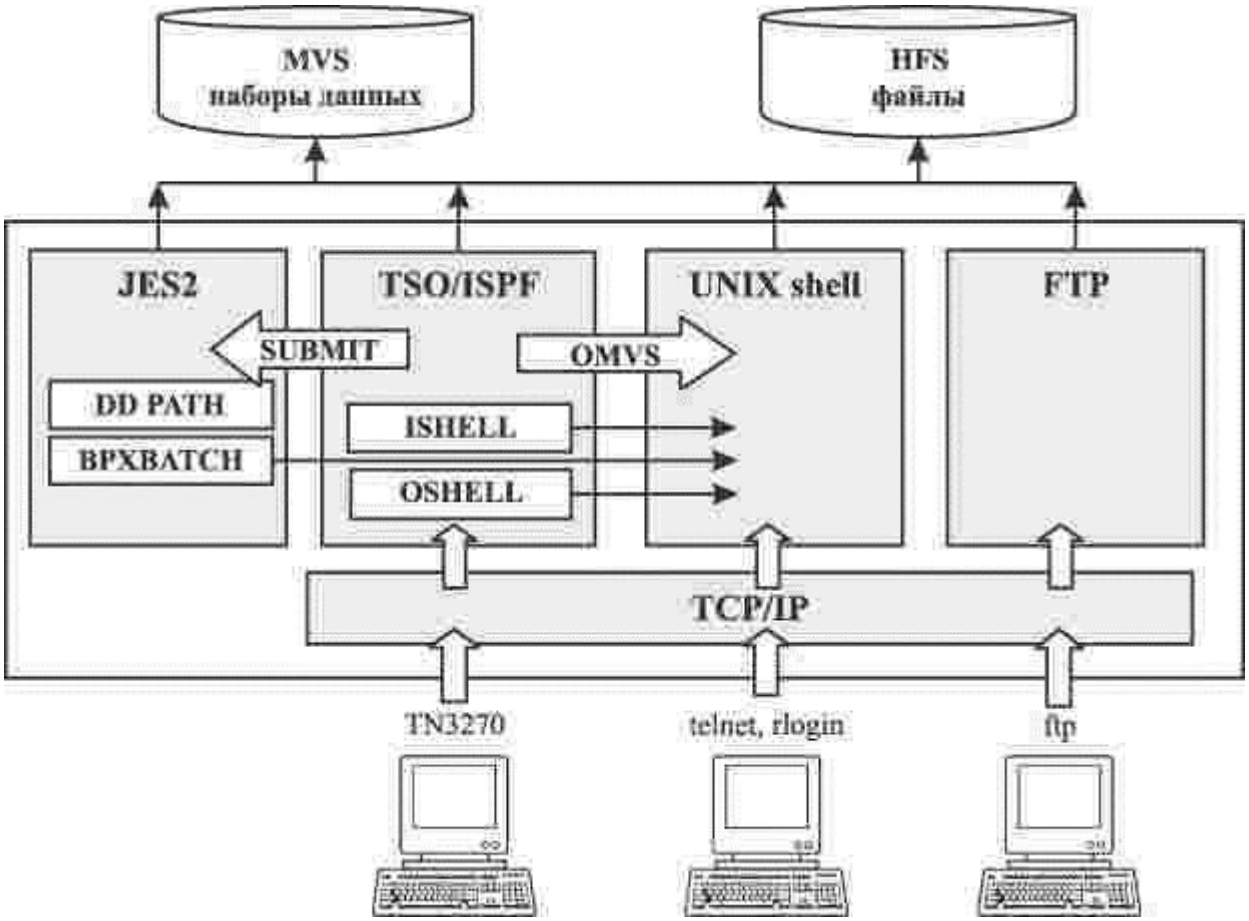


Рисунок 5 – Варианты доступа к системным сервисам UNIX.

Первый вариант подключения терминала пользователя через TCP/IP соединение является традиционным для пользователей z/OS и основан на использовании компонентов TSO/E и ISPF. В TCP/IP-сети

терминалы TSO поддерживаются на основе специального протокола TN3270, представляющего собой адаптированный вариант стандартного протокола telnet.

Второй вариант доступа к сервисам UNIX основан на использовании стандартных прикладных протоколов TCP/IP rlogin и telnet. Для работы используется асинхронный ввод, возможно использование текстового редактора vi, но ограничен доступ к командам TSO. Также необходима предварительная настройка серверных компонент telnet и rlogin, а также разрешение на доступ в профиле RACF пользователя.

Третий вариант доступа к сервисам UNIX основан на использовании ftp-протокола, также являющегося стандартным прикладным протоколом TCP/IP. В данном режиме можно получать доступ к данным MVS и UNIX.

2. Практическая часть

2.1. Запуск оболочки shell системных сервисов UNIX

Зарегистрируйтесь в системе TSO (Вы уже знаете, как это сделать, поскольку выполнили лабораторную работу №1 «»).

Запустите интерфейс системных сервисов OS/390 UNIX shell.

Запустите эмулятор терминала 3270 и подключитесь к TSO, используя логин TSO ID, выданный Вам преподавателем. Введите пароль. Если процедура подключения не вызвала интерфейс ISPF, вы увидите командную строку “TSO READY”. Если же процедура подключения вызвала ISPF, то вы увидите на экране главное меню интерфейса ISPF, как показано на рисунке 6. В этом случае выберите пункт 6 «`Command Enter TSO or Workstation commands TSO`», появится окно, показанное на рисунке 7. Введите команду OMVS.

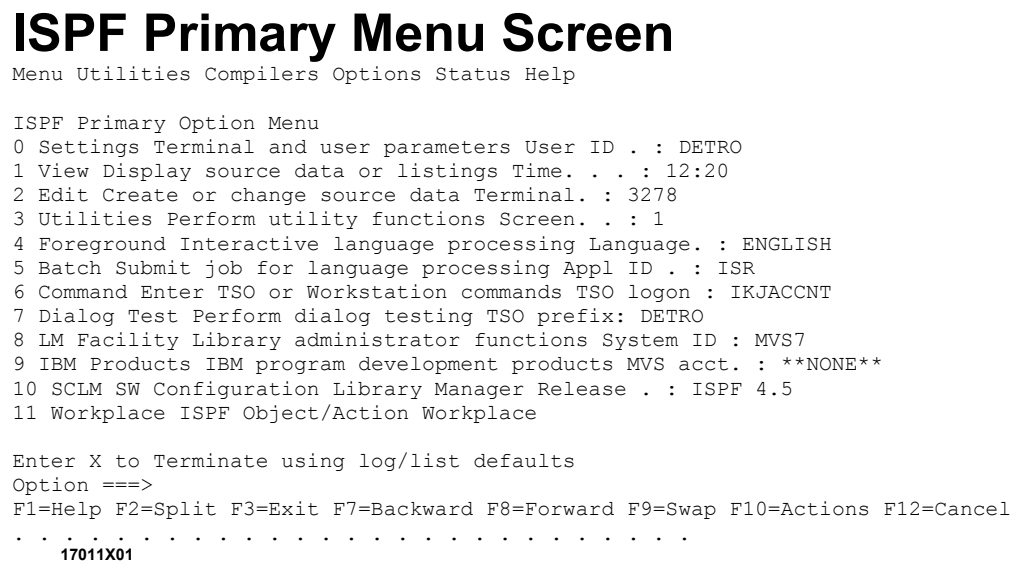


Рисунок 6 - Главное меню ISPF.

Если вы видите строку TSO READY на черном экране, введите команду **omvs**, чтобы вызвать оболочку системных сервисов UNIX - USS shell.

```
IBM
Licensed Material - Property of IBM
5647-A01 (C) Copyright IBM Corp. 1993, 1997
(C) Copyright Mortice Kern Systems, Inc., 1985, 1996.
(C) Copyright Software Development Group, University of Waterloo, 1989.
All Rights Reserved.
U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.
IBM is a registered trademark of the IBM Corp.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- Improve performance by preventing the propagation -
- of TSO/E or ISPF STEPLIBs -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
$
===>
INPUT
ESC=ў 1=Help 2=SubCmd 3=HlpRetrn 4=Top 5=Bottom 6=TSO 7=BackScr 8=Scroll
9=NextSess 10=Refresh 11=FwdRetr 12=Retrieve
```

Рисунок 7.

Введите команду **pwd** для просмотра вашего текущего рабочего каталога**.** Посмотрите, как называется ваш рабочий каталог, и зафиксируйте это, вы должны будете указать его имя в отчете. Найдите корневой каталог.

Введите команду для вывода всех файлов вашего текущего рабочего каталога **ls –a.** Сколько элементов было выведено в результате предыдущей команды?

Замечание: Если не был выведен ни один элемент, то это потому, что вы не указали опцию **-a**, в качестве параметра команды **ls**.

Введите команду для перехода в родительский каталог вашего домашнего каталога - **cd /u.** Посмотрите, является ли он вашим домашним каталогом, если не является, то посмотрите, как называется родительский каталог вашего домашнего каталога.

Еще раз введите команду для вывода имени текущего рабочего каталога **pwd.** Как теперь называется ваш текущий рабочий каталог? Является ли этот каталог вашим домашним каталогом? Изменился ли ваш домашний каталог?

Выведите все файлы данного каталога вместе с их атрибутами - **Команда: ls –al.** Посмотрите, к какому типу относится большинство файлов данного каталога, как выглядит строка разрешения, присвоенная вашему домашнему каталогу, как эти разрешения будут выглядеть в восьмеричном представлении и показан ли на экране номер inode для этих файлов.

Замечание. Если результаты команды **ls –al** не помещаются на экран, и экран автоматически прокручивается, используйте PF клавиши F7 и F8 для прокрутки экрана.

Введите команду для вывода номера **inode** вместе с другой информацией из предыдущего вопроса. **Команда: ls –ali.** Посмотрите, какой inode номер у вашего домашнего каталога.

Существует множество способов для перехода в ваш домашний каталог, это команды:

- **cd/u/xxxxxx ;**
- **cd xxxxxx ;**
- **cd $HOME ;**

- **cd . ;**
- **cd ;**

Где **xxxxxx** ваш TSO user id в нижнем регистре.

Вернитесь в ваш домашний каталог, используя одну из перечисленных выше команд.

2.2. Работа с файлами и каталогами

2.2.1. Создание и копирование файлов и каталогов

Создайте новый каталог с именем **subdir1** из вашего домашнего каталога. Для этого используйте **команду mkdir subdir1.**

Из вашего домашнего каталога создайте еще один каталог под названием **subdir2**. Перейдите в каталог subdir1. Создайте в нем файл hobbies и запишите в него список файлов из вашей домашней директории и вернитесь в домашний каталог.

Команды: cd subdir1;

touch hobbies ;

ls –l > hobbies ;

cd $home

Просмотрите содержимое файла **hobbies**. Для этого используйте команду **cat hobbies**

Скопируйте файл **hobbies** из каталога **/u/subdir1/** в ваш домашний каталог. Не изменяйте ваш текущий рабочий каталог.

Команда: cp /u/kc03c11/subdir1/hobbies /u/kc03c11/

Перечислите атрибуты вашего нового файла вместе с номером inode. Для этого используйте команду **ls -li hobbies.** Посмотрите, какие разрешения установлены для созданного вами нового файла hobbies. Они должны быть следующими: **rwxr-xr-x**. Если это не так, тогда они такие же, как и для файла /u/samples/uss/hobbies . Посмотрите, какой inode номер у этого файла и когда был создан этот файл **(**эти значения могут меняться).

Выведите содержимое файла **hobbies**

Команда: cat hobbies.

Создайте файл **comics.lst** в каталоге **subdir2**.

Команды: cd subdir2;

touch comics.lst ;

ls –li > comics.lst;

cd $home

Скопируйте файл **comics.lst** из каталога **/u/kc03c11/subdir2** в ваш каталог **subdir1**.

Команда: cp /u/kc03c11/subdir2/comics.lst /u/kc03c11/subdir1

Переименуйте ваш подкаталог **subdir2** в **links**.

Команда: mv subdir2 links

Выведите содержимое файла **comics.lst**.

Команда: cd subdir1 cat comics.lst

Чтобы при выводе содержимого файла его содержимое последовательно выводилось на экран, имеется команда**: more comics.lst**

Создайте файл под названием **mtfile** в вашем домашнем каталоге, используя команду **touch**, затем выведите атрибуты этого файла. Обратите внимание, что размер этого файла **zero (0) bytes.**

Введите команду **touch** для файла **hobbies** в вашем домашнем каталоге. Команда **touch** обновляет последнюю измененную дату и timeof существующий файл или она создает новый пустой файл.

Подсказка: Посмотрите на время последнего обновления файла.

Введите команду, которая выведет все файлы из вашего домашнего каталога и перенаправит вывод в файл **myfiles**.

Команда: ls -l > myfiles

Добавьте текущие время и дату к содержимому файла **myfiles**.

Команда: date >> myfiles

Выведите содержимое файла **myfiles**.

Команда: cat myfiles

2.2.2. Жесткие и символические ссылки

Создайте жесткую ссылку в каталоге links и назовите ее hlink. Эта ссылка должна указывать на файл comics.lst из каталога subdir1.

Команда: ln subdir1/comics.lst links/hlink

Выведите номера inode и остальные атрибуты файлов comics.lst и hlink.

Команда: ls -li subdir1/comics.lst links/hlink

Посмотрите, отличаются ли номера inode? Они должны отличаться, тогда как остальные атрибуты (дата, размер, и т. д.) совпадают. На эти файлы указывают две жесткие ссылки. Посмотрите, сколько сейчас существует файлов, содержащих данные из comics.lst **Подсказка:** Один.

Создайте символическую ссылку в каталоге links и назовите ее slink. Эта ссылка должна указывать на comics.lst из каталога subdir1.

Команда: ln -s /u/kc03c11/subdirl/comics.lst /u/kc03c11/links/slink

Замечание: Лучше использовать полные пути для формирования этих ссылок из-за способа их разрешения.

Выведите номера inode и другие атрибуты файлов comics.lst и slink.

Команда: ls -li subdir1/comics.lst links/slink

Как видите, не совпадают номера inode, остальные атрибуты файлов (дата, размер и т.д.) **с**овпадают, но не все. На этом этапе вашей работы одна жесткая ссылка указывает на файл slink и 2 на comics.lst. При этом, существует все еще только один файл с данными comics.lst.

Примечание: ***Жесткая ссылка*** - это объект каталога, имеющий тот же номер inode, что и исходный файл, и являющийся, по сути, еще одним указателем на файл. ***Символическая ссылка*** – это тип файла, содержащий указатель на каталог исходного файла, а не на сам файл.

Создайте новый подкаталог в домашнем каталоге и назовите его **newdir**, переместите в него файл **myfiles**.

Команды: **mkdir newdir ;**

mv myfiles newdir

Выведите все файлы, подкаталоги и их содержимое, начинающиеся из вашего домашнего каталога.

Команда: ls –aR

Результат работы показан на рисунке 8.

```
$ ls –aR
.:
. .profile hobbies mtfile subdir1
.. .sh_history links newdir
./links:
. .. hlink slink
./newdir:
. .. myfiles
./subdir1:
. .. comics.lst
```

Рисунок 8 – Результат работы

Скопируйте файл comics.lst в каталог newdir.

Команда: cp subdir1/comics.lst newdir

Создайте жесткую ссылку в каталоге links и назовите ее hlink. Эта ссылка должна указывать на файл comics.lst из каталога subdir1.

Команда: ln subdir1/comics.lst links/hlink

2.2.3. Удаление файлов, каталогов и ссылок

Догадайтесь, что произойдет при удалении файла comics.lst. Что произойдет с символической ссылкой? Ссылка останется, но будет указывать на несуществующий файл. Она будет отображаться при листинге, но открыть ее будет невозможно. А что произойдет с жесткой ссылкой? С жесткой ссылкой ничего не произойдет. Останется одна ссылка. Что при этом произойдет с данными на физическом уровне? С данными ничего не случится. Данные сохраняются до тех пор, пока существует хотя бы одна жесткая ссылка на них.

Удалите comics.lst и проверьте эти предположения.

Команда: rm subdir1/comics.lst

Что произойдет с символической ссылкой, если вы скопируете comics.lst обратно в каталог subdir1 из /u/kc03c11/newdir? Символическая ссылка будет переназначена.

Что произойдет с жесткой ссылкой? С жесткой ссылкой ничего не произойдет. Новая копия comics.lst будет иметь другой номер inode, то есть, это будет другой файл.

Если вы скопируете comics.lst в ваш домашний каталог, то с символической ссылкой ничего не произойдет. Символическая ссылка будет указывать на каталог subdir1, а не на домашний каталог.

Скопируйте comics.lst обратно в каталог subdir1.

Команда: cp /u/kc03c11/links/comics.lst subdir1

Попробуйте удалить каталог **newdir.**

Команда: rmdir newdir.

Если каталог не пуст, то его нельзя удалить. Удалите файлы myfiles и comics.lstr из каталога newdir.

Команда: rm newdir/myfiles newdir/comics.lst

Можете ли вы теперь удалить каталог. Вы могли бы удалить каталог newdir, не удаляя все его файлы, воспользовавшись командой **rm -r newdir**

2.2.4. Сортировка файлов

Отсортируйте содержимое файла hobbies из вашего каталога /samples, не используя опций команды.

Команда: sort /samples/hobbies.

Записи в файле были отсортированы в алфавитном порядке, по первому символу каждой строки. При этом файл hobbies не изменяется. Отсортированные данные выводились только на экран.

Отсортируйте файл hobbies, используя второе поле каждой записи.

Команда: sort –k2 /samples/hobbies.

Снова отсортируйте файл hobbies, используя второе поле каждой записи, но на этот раз укажите в параметрах команды игнорировать первый символ пробела в любом поле.

Команда: sort -b -k2 /samples/hobbies или sort -b +1 /samples/hobbies.

Посмотрите, корректна ли сортировка на этот раз.

Отсортируйте файл hobbies в соответствии с числами в последнем поле каждой записи.

Команда: sort -n -k4 /samples/hobbies или sort -n +3 /samples/hobbies.

Отсортируйте файл comics.lst из вашего каталога subdir1. Отсортируйте второе поле файла, используя символ (:) в качестве разделителя полей.

Команда: sort -t: -k2 /samples/comics.lst или sort -t: +1 /samples/comics.lst.

После выполнения этих команд файл был отсортирован по второму полю, но не в числовом порядке, а в соответствии с кодом ASCII.

Отсортируйте этот файл снова, но в числовом порядке.

Команда: sort -t: -n -k2 /samples/comics.lst

2.3. Изучение основных команд оболочки shell

Введите **shell** команду **id**. По этой команде система выводит номер **uid**, имя **userid, gid** номер и имя группы.

Введите команду **date.** Используйте TSO команду OBROWSE для просмотра файла hobbies или любого другого текстового файла из вашей домашней директории.

Введите команду **tty.** Данная команда выводит имя файла-терминала из каталога /dev, который представляет соединение терминала студента (с мейнфреймом).

Введите команду **logname**. Команда **logname** выводит logon имя пользователя.

Введите команду **env** и объясните значение выходных данных. Команда **env** выводит установленные переменные среды и их значения. Исходя из значений переменных среды, сколько строк текста будет выводиться на экран. Количество выводимых строк определяется переменной “lines”, обычно оно равно 20. Переменная для хранения домашнего каталога называется **HOME**.

Оболочка **Shell** расположена в **/bin/sh.** Если вы выведите информацию по команде **date**, используя команду **type**, будет выведено сообщение **Date is cached /bin/date.** Это означает, что команда **date** расположена в директории **/bin,** и что значение даты кэшируется в памяти для повышения производительности.

Просмотрите информацию о других известных вам командах.

> **Примечание:** Для получения справки по синтаксису, параметрам и функциям команд используйте команду: **man** <*имя команды*>. Например, набрав “**man cp**” вы сможете увидеть справочную информацию о команде **cp** (команда cp, например, копирует файлы). Для перемещения по страницам текста используйте клавишу **Enter**. Некоторые **man** описания

достаточно длинные. Для выхода из справки **man** окна, нажмите **q** и вы вернетесь в режим командной строки.

Узнайте размер свободного пространства в смонтированной файловой системе, используя команду **df.** Посмотрите, чему равен размер свободного пространства в вашей файловой системе. Это значение меняется. Результат выводится в блоках (единицах).

Выведите содержимое каталога **root** с подробной информацией о его компонентах. **Команда: ls -l /.**

Вы можете отличить каталог от файла, поскольку для каталога первый символ в выводимой строке “d”. Пятое выводимое поле означает размер компонента в байтах.

Выведите содержимое вашего домашнего каталога вместе с номерами **inode.**

Команда: ls –l /u/kc03c11.

Номер inode присваивается системой UNIX для управления файлами.

Выведите содержимое вашего домашнего каталога, перенаправив вывод в файл listout.

Команды: cd $home

ls -li > listout

Воспользуйтесь командой **cat** для просмотра файла **listout.**

Создайте новый каталог в каталоге **newdir** и назовите его **Dir1A**.

Команды: mkdir newdir

mkdir newdir/Dir1A

Введите команду для просмотра битов разрешения вашего нового каталога.

Команда: ls -al newdir/DirlA.

Биты разрешения для DirlA равны **rwxr-xr-x** или **755**. Измените биты разрешений для данного каталога, так чтобы владелец каталога имел права чтения, записи и выполнения, члены группы и все остальные пользователи получили только право чтения.

Команда: chmod 744 newdir/DirlA или

chmod go-x newdir/DirlA

Отсортируйте содержимое ранее созданного файла listout по значению поля “размер”. Перенаправьте вывод результатов в файл sorted. Поместите файл sorted в каталог Dir1A.

Команда: sort -k5 listout > newdir/DirlA/sorted

Объедините команду вывода содержимого каталога и команду сортировки и создайте конвейер команд, позволяющий просмотреть содержимое каталога в отсортированном по размеру порядке.

Команда: ls -l | sort -k5.

Имейте в виду, что вы не сможете сделать это непосредственно в одной команде.

Используйте команду **grep** для вывода только тех записей файла hobbies, которые имеют отношение к “links” (поиск текста в файле).

Команда: grep links hobbies .

Выведите содержимое каталога **/etc.** Обратите внимание на поле дата и убедитесь, что компоненты каталога отличаются по году или месяцу.

Команда: ls –l /etc.

Объедините следующие задачи в одну командную строку, используя конвейер: выведите содержимое каталога /usr, выберите записи с определенным месяцем или днем месяца, отсортируйте выходные данные по

размеру.

Команда: ls –l /usr | grep "Apr" | sort -k5

Выведите содержимое вашего домашнего каталога и перенаправьте вывод в файл outlist. Запустите эту команду в фоновом режиме.

Команда: ls -l > curlist &

В результате вы получите ID задачи (job id) и PID фонового процесса.

Повторите команду, запустив ее в фоновом режиме и перенаправив вывод в файл outlist.

Команда: ls -l /usr | grep "Apr" | sort -k5 > curlist &

Убедитесь, что была выведена одна ID задача (job ids) и три ID процесса. Имейте в виду, что id группы (GID) равен ID первого процесса. Введите команду **history.** Эта команда выводит последние введенные 16 команд. Выберите одну команду из истории команд и перезапустите ее, напечатав **r** перед номером, соответствующим выбранной команде. **Команда: r 173**

Выведите содержимое файла **sh_history**.

Команда: cat .sh history.

Файл .sh_history содержит список ранее введенных команд. Обратите внимание на разницу между списком команд, полученным при помощи команды **history** и содержимым файла **.sh history**. Команда history выводит последние 16 команд вместе с номерами, присвоенными этим командам. Файл .sh_history содержит гораздо больше команд, но не содержит их номера.

2.4. Настройка оболочки shell

Введите команду echo для вывода значений переменных LOGNAME и PWD.

Команда: echo $LOGNAME $PWD.

Переменная LOGNAME равна регистрационному имени пользователя. (logon name), а переменная PWD равна текущему рабочему каталогу.

Вы можете настроить оболочку shell таким образом, чтобы она выводила login и текущий рабочий каталог. Значение, присваиваемое переменной необходимо заключить в одинарные кавычки, тогда имя рабочего каталога будет корректно отображаться при вводе команды cd.

Команда: PS1 = '$LOGNAME:$PWD: >' `

Используя команду OEDIT, добавьте команду из предыдущего пункта к файлу .profile из вашего домашнего каталога. После завершения редактирования файла нажмите клавишу F3, чтобы его закрыть.

Команда: oedit .profile

2.5. Написание shell скриптов

Создайте новый каталог scripts в вашем домашнем каталоге.

Команда: mkdir scripts

Создайте новый файл myscript в каталоге scripts. Используя конвейер команд напишите команду для вывода содержимого текущей рабочей директории, отсортировав результаты по размеру, и сохраните данную команду в файле scripts.

Скрипт: **ls -l | sort -k5**

touch myscripts;

ls –l | sort –k5 > myscripts

Запустите ваш новый скрипт.

Команда: scripts/myscript.

На экран будет выведено следующее сообщение: “Cannot execute, permission denied.” Если вы увидели это сообщение об ошибке, исправьте биты разрешения и перезапустите скрипт.

Команды: chmod +x scripts/myscript;

script/myscript

Напишите скрипт, спрашивающий у пользователя его имя и возраст и генерирующий предложение на основании полученных данных. Назовите его **hello** и запустите. Команда **shell - read** может считывать значение и присваивать его переменной. Команда echo может использоваться для вывода сообщения на экран.

Скрипт: echo Please enter your name:

```
read name
echo Please enter your age:
read age
echo Hello, $name. You are $age years old.
```

Напишите скрипт, который делает следующее: запрашивает у пользователя его имя и год рождения; вычисляет возраст пользователя и выводит предложение содержащее этот возраст. Сохраните файл под именем **age**. Используйте текущий год как константу при расчетах. Выполните скрипт. Откройте редактор и наберите скрипт.

Команда: oedit age

Скрипт: echo Please enter your name:

```
read name
echo Please enter the year of your birth:
read year
let age=2006-year
echo Hello, $name. You are $age years old.
```

Напишите скрипт dirlist, который выполняет следующие действия: запрашивает директорию для вывода; если пользователь ничего не вводит, то выводится домашняя директория пользователя.

Скрипт: echo Directory:

```
  read dir
if test -z "$dir"
then dir=$HOME
fi
ls -l $dir
```

Напишите скрипт dirlist2, который выполняет следующие действия: Запрашивает имя директории до тех пор, пока оно не будет введено. После этого выводит содержимое данной директории.

Скрипт: while test "$1" = ""

```
do
echo Directory:
read dir
set "$dir"
```

```
done
ls -l $1
```

2.6. Создание awk программ

Перед выполнением данной части лабораторной работы должны быть созданы следующие текстовые файлы **sport** и **Cars**:

```
Jim   Fost  student    11.10.87   5   football
Mike  Brown student    12.09.86   6   football
John  Rodd  student    05.09.86   3   swimming
Bryan Talbot lecturer  04.07.75   10  basketball
Nick  Shaw  student    03.08.87   5   swimming
Ann   Brook lecturer   08.04.79   6   swimming
Jane  Fell  student    14.05.83   2   volley-ball
```

Рисунок 9. Содержимое файла sport

```
BMW : Paul : Brown : 12389
BMW : Jane : Tod :  34567
Chrysler : Mike : Bell : 67890
Chevrolet : Ben : Walsh : 45634
```

Рисунок 10. Содержимое файла Cars

Введите awk команду, которая выведет первое поле каждой записи файла sport.

Команда: awk ' {print $1}' sport

Введите awk команду, которая выводит все поля для всех записей, начинающихся с имени “Jim”.

Команда: awk '$1 == "Jim" {print}' sport

Введите awk команду, которая для выводит первое и последнее поле для всех записей, начинающихся с имени “Jim”.

Команда: awk '$1 == "Jim" {print $1,$6}' sport

Запустите ISPF редактор при помощи команды oedit и создайте файл awk1. В файле awk1 напишите программу, которая выполняет следующие действия: для всех записей, начинающихся с “Jim”, выводит первое и последнее поле, и для всех записей, начинающихся с “Ann”, выводит второе и третье поле. В качестве входного файла укажите sport. Запустите программу.

Программа:
```
$1 == "Jim"  {print $1,$6}
$1 == "Ann" {print $2,$3}
```

Запуск на исполнение:

Команда: awk -f awk1 sport

Напишите awk программу awk2 , которая выводит название каждого поля файла sport. Названия полей: "Name Surname Occupation Date of Birth Exper Sport". Запустите вашу программу.

Программа:
```
BEGIN {print " Name Surname Occupation Date of Birth Exper Sport"}
      {print}
```

Команда: awk -f awk2 sport

Напишите awk программу awk3, которая выполняет следующие действия: подсчитывает количество человек, занимающихся футболом и плаванием и выводит результаты.

Программа: BEGIN {footcount = _ swimcount = _}

$6 == "football" {footcount = footcount + 1}

$6 == "swimming" {swimcount = swimcount + 1}

END {print footcount " people enjoy jogging."

print swimcount " people enjoy bridge."}

Команда: awk -f awk3 sport

Напишите программу, которая выполняет следующие действия: использует символ (:) в качестве разделителя полей входного файла и выводит первое и второе поля, разделенные символом (:), затем выводит третье и четвертое поля, разделенные символом (,). В качестве входного файла укажите cars.lst. Запустите программу.

Программа: BEGIN {FS = ":"}

{print $1 FS $2 ", " $3 ", " $4}

Команда: awk -f awk4 cars.lst

Напишите awk программу awk5, которая выполняет следующие действия: использует символ (:) в качестве разделителя полей входного файла, считывает все записи, содержащие заглавную букву C и выводит эти строки без последнего поля.

Программа: BEGIN {FS = ":"}

/C/ {print $1 ":" $2 ":" $3}

Команда: awk -f awk5 cars.lst

Напишите awk команду awk6, которая выполняет следующие действия:

- Использует символ (:) в качестве разделителя полей входного файла.
- Считывает все записи, содержащие заглавную букву C.
- Выводит эти записи.
- Подсчитывает сумму по последним полям выведенных строк.
- Выводит результат суммирования.

Программа: BEGIN {FS = ":"

total = _}

/C/

{amount = substr($4,2)

total = total + amount

print $_}

END {print "The total is $"total}

Команда: awk -f awk6 cars.lst

Создайте новый подкаталог и назовите его bin. Скопируйте в него все созданные программы.

Команды: **mkdir bin;**

mv awk* bin

2.7. Создание программ на языке C

Используя редактор **oedit**, наберите следующую C программу и сохраните ее как **hello.c** .

Программа: #include <stdio.h>

main()

```
{
printf(“Hello World \n”);
}
```

Используя команду **c89** откомпилируйте и скомпонуйте программу.

Команда: c89 hello.c

Компилятором будут созданы файлы **hello.o** и **a.out**. Исполняемый файл: **a.out**, биты разрешений для исполняемого файла установлены следующие: **-rwxr-xr-x.** Запустите программу.

Команда: a.out

3. Контрольные вопросы

Чем отличается рабочий каталог от домашнего каталога?
Как называется родительский каталог домашнего каталога?
Как называется корневой каталог?
Можете ли вы создать два каталога одной командой?
Какие разрешения установлены для новых файлов?
Какой inode номер у файла?
Как вывести содержимое файла так, чтобы его содержимое последовательно выводилось на экран?
Как определить размер файла?
Что делает команда touch?
Является ли жесткая ссылка файлом?
Кто создает файл .sh history?
Каким образом используются переменные среды?
Сколько ID процессов создается при выполнении конвейера команд?
Какой файл хранит параметры интерфейса пользовательской консоли?
Как узнать размер свободного пространства в смонтированной файловой системе?
Что такое shell скрипт?
Какой командой можно вывести произвольную текстовую строку?
Что такое Awk и для чего он используется?
Из каких частей состоит Awk программа?
С какими типами файлов работает Awk программа?
Будет ли Awk программа корректно работать с текстовыми файлами Windows?
Как запустить Awk программу?
Как называется ваш рабочий каталог?
Является ли он вашим домашним каталогом?
Как называется родительский каталог вашего домашнего каталога?
Как называется корневой каталог?
К какому типу относится большинство файлов домашнего каталога?
Как выглядит строка разрешения, присвоенная вашему домашнему каталогу?
Как разрешения будут выглядеть в восьмеричном представлении?
Показан ли на экране номер inode для файлов?
Перечислите способы перехода в домашний каталог
Можете ли вы создать два каталога одной командой?
Какие разрешения установлены для созданного вами нового файла hobbies?
Как вывести содержимое файла так, чтобы его содержимое последовательно выводилось на экран?
Что делает команда touch?
В чем разница между жесткими и символическими ссылками?
Каким образом может быть отсортирован файл?
Изменяется ли файл после того, как он отсортирован?
Как называется переменная для хранения домашнего каталога?
Где находится оболочка shell?
Как можно отличить каталог от файла?
Для чего нужен номер inode?
Что означает пятое выводимое поле?
Что хранит файл .sh_history?
Чему равна переменная PWD?

Лабораторная работа №4

Создание клиент-серверных приложений на языке EGL, серверная часть программы

Целями лабораторной работы являются:

3. Установка подключения серверного приложения к выбранной базе данных.
4. Создание серверного приложения, которое обслуживает запрос к базе данных и выдает результат успешного обращения или сообщение о неудаче.
5. Тестирование серверной части программы.

1. Теоретическая часть

В этой работе мы создадим программу, которая будет использовать 2 разных интерфейса – TUI и WEB.

TUI – text user interface – пример такого интерфейса это эмулятор терминала Vista 3270, т.е. если программу, которую мы разработаем, поместить на сервер или на мэйнфрейм – мы сможем подключиться с помощью Vista 3270 – и запустить нашу программу (для этого нам необходимо создать клиентскую программу, для поддержку TUI – об этой см. приложение к Лабораторной работе №1)

WEB интерфейс – мы можем запустить нашу программу, используя браузер (например Internet Explorer), сейчас этот вариант становится очень популярным, т.к. для работы с программой нам не нужен ее дистрибутив, достаточно иметь только интернет-соединение с сервером, на котором установлена серверная часть программы и клиентская, которая будет выводить данные на экран браузера в формате веб-страницы. Веб-интерфейс также часто применяется для управления сетевыми и прочими устройствами (например маршрутизаторами или модемами). Клиентскую часть программы для реализации WEB-интерфейса мы рассмотрим в лабораторной работе №2.

Для работы мы будем использовать программный пакет **IBM WebSphere Developer for zSeries** версии 6.0.1, но стоит отметить, что можно использовать и другие средства разработки IBM, поддерживающие технологию EGL. (например IBM Rational Software Development Platform версии 6 и др.)

Так же будет задействована база данных DB2, которая сейчас используется почти во всех решениях от IBM. Она является открытой (это ее преимущество перед базой данных Oracle) и более функциональной, чем база данных mySQL (mySQL предназначена в основном для хранения данных сайта или портала). DB2 – можно установить как на сервер по управлением ОС Windows или ОС Unix/Linux, так и на мэйнфрейм. Обращение к ней происходит средствами языка SQL.

Несколько слов о программе: это простая программа, которая будет делать запрос по идентификатору userid к таблице STAFF (часть базы данных DB2), и если идентификатор будет найден серверная часть вернет нам имя, которому соответствует идентификатор, в противном случае выдаст сообщение о том, что такого идентификатора нет. Стоит помнить, что на практике серверная часть программы может быть очень сложной (например система учета билетов РЖД).

Очень удобно писать программы по схеме “back to the front” – сначала писать серверную часть программы, ее логику, а потом интерфейс. Сначала определимся с записями в базе данных, потом с рабочими областями (working areas) (EGL Data parts), а затем займемся программной логикой (EGL Logic parts).

Нам нужно создать 2 модуля: первый будет размечать(maps) таблицу в базе DB2, а второй будет определять рабочую область между клиентской частью (TUI/WEB) и серверной.

Теперь немного о модулях – программа на EGL состоит из набора модулей (parts), каждый из которых - это модуль, в котором содержатся объявления (declarations). Можно объявлять переменные в любом, удобном для вас порядке, и присваивать им имена.

Существуют 3 типа модулей (parts):

Data определяет структуры данных, которые будут использоваться в программе. В некотором роде их можно сравнить с COBOL *Copybook* или COBOL *Data Division*

Logic определяется выполняемую последовательность, которую мы напишем на языке EGL. Это можно сравнить с COBOL *Procedure Division*

Build определяет множество характеристик процесса компиляции и build. В этой работе нам понадобяться *build descriptor* и *linkage options.*

2. Выполнение лабораторной работы

План работы:

- Создать проект для работы с серверной частью, используя EGL
- Создать модули
- Провести отладку и тестирование программы

2.1 Начальная настройка студии

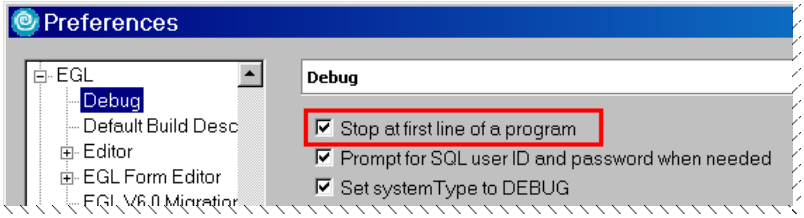
Если используется VMware, то настройка не нужна, т.к. там уже все настроено для проведения этой работы.

Необходимо сделать следующее:

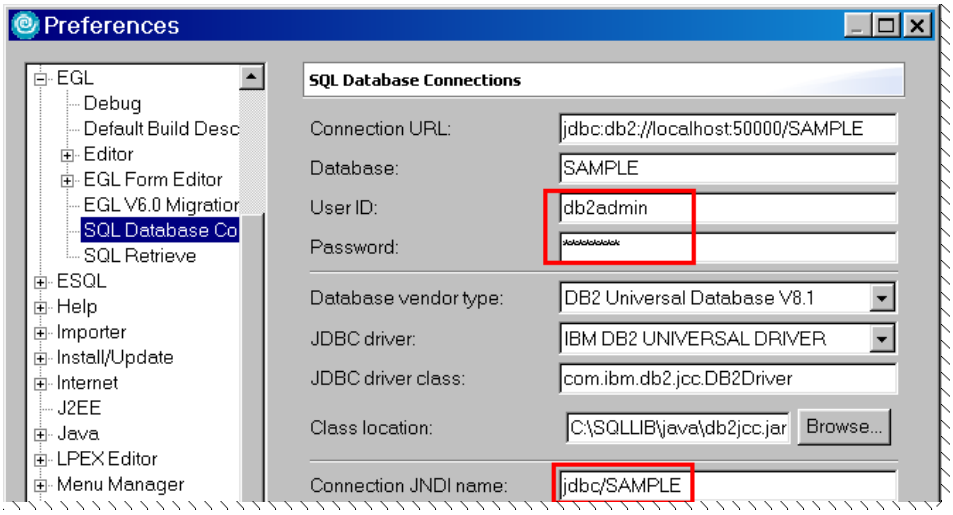
1) подключить EGL: Window → Preferences → Workbench → Capabilities → EGL Developer

2) настроить EGL: Window → Preferences → EGL -→

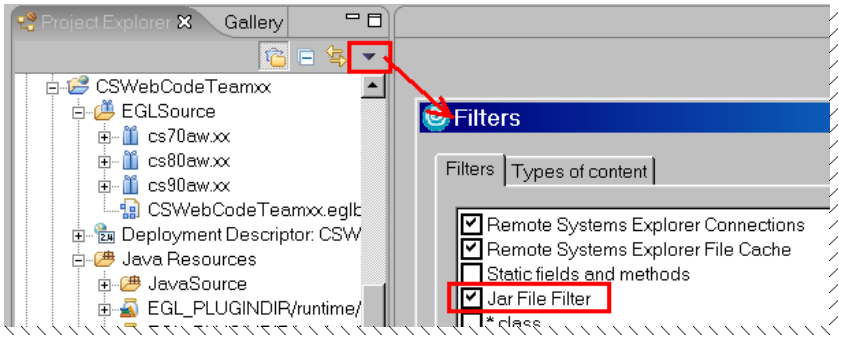
2.1.1 – Debug: Stop at first line of a program



2.1.2 – SQL Database Connections : настроим DB2, введите userid и password defaults, используем JNDI default : jdbc/SAMPLE



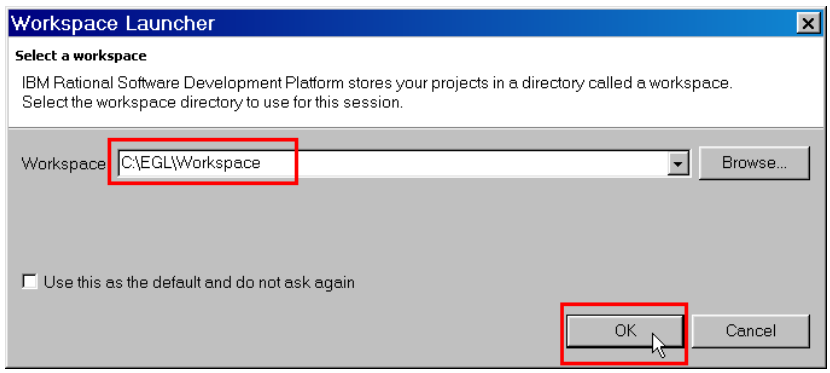
2.1.3 Добавим JAR filter чтобы не засорять Project Explorer:



2.2 Создание проекта в программном пакете **IBM WebSphere Developer for zSeries (**далее WDz)

2.2.1 Запускаем программный пакет

2.2.2 При загрузке всплывет окно, предлагающее выбрать рабочее пространство (WorkSpace) – указываем его, но ***не*** отмечаем флажком – использовать по умолчанию.



Программный продукт WDz загрузится. Это может занять некоторое время (особенно, если вы используете VMWare)

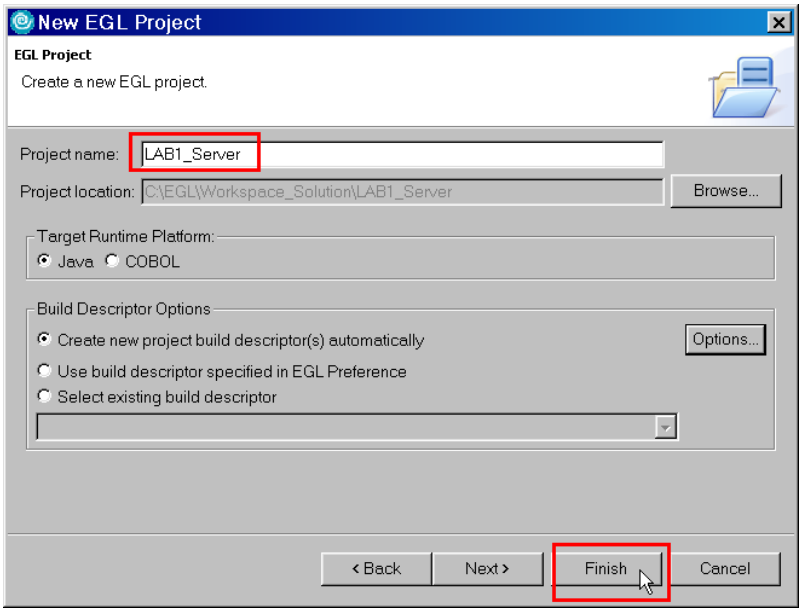
2.3 Настроим новый проект:

Используем мастер Новый проект: **File → New → Project**

Выбираем **EGL Project**

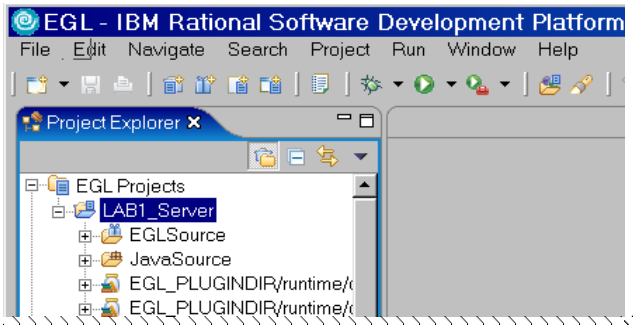
Введите имя проекта (например **LAB1_Server**), отметим **Create new project build descriptor(s) automatically**

Примечание: Если можно выбрать (в зависимости от версии и типа вашего программного пакета) выбираем **Java** (для генерации Java), если выбрать нельзя, значит будет генерироваться только Java-код.



Согласитесь со всплывшим окно, настраивающим вид студии.

Получиться должно следующее:



2.4 Создаем модули EGL

2.4.1. Создание модуля SQL RECORD EGL Data parts.

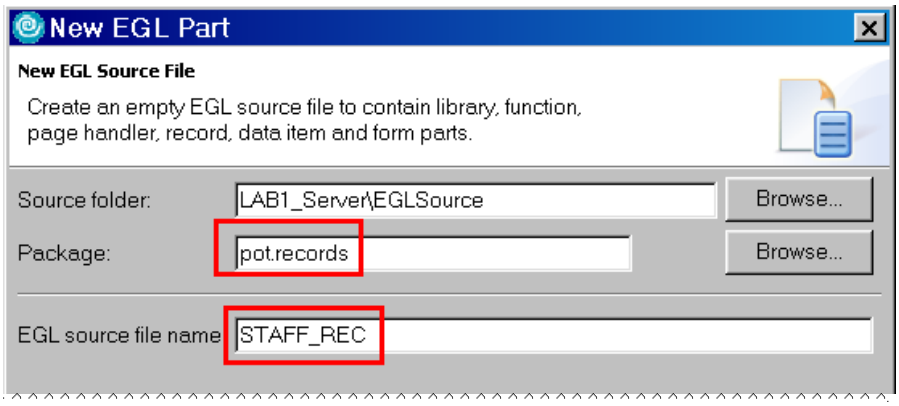
Нам нужно создать 2 модуля:

- модуль STAFF_REC будет размечать (maps) таблицу в базе DB2,
- модуль STAFF_WORK (Working area EGL Data) будет определять рабочую область между клиентской частью (TUI/WEB) и серверной.

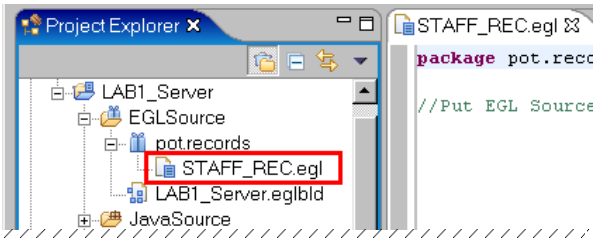
Для создания записи *STAFF_REC* делаем следующее:

Выбираем наш проект, нажимаем правой кнопкой по нему и **New → EGL Source File**

Назовем package **pot.records**, и назовем файл - **STAFF_REC**

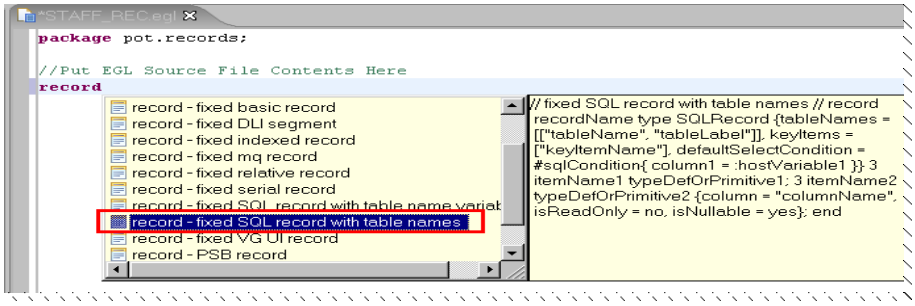


В дереве иерархии слева найдем только что созданный файл и раскроем его:



В открывшемся EGL editor воспользуемся функцией помощи автозаполнения. Установив курсор после комментария нажимаем **Ctrl + Space** – в списке выбираем запись **record – fixed SQL record with table names.**

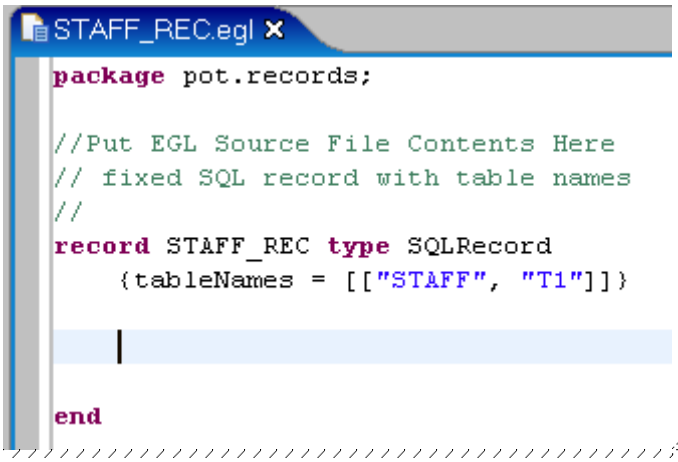
Примечание: Если нет фиксированной записи, то выбираем обычную - **SQL record with table names**)



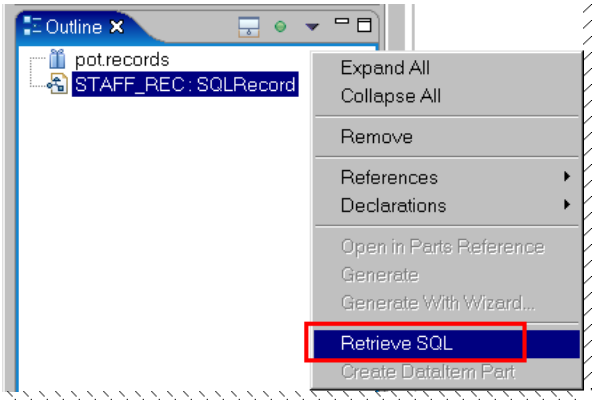
Заполним выведенный на экран шаблон – вместо recordName ставим **STAFF_REC,** вместо tableName ставим ***.STAFF** и **T1** вместо tableLabel.

Примечание: * - идентификатор первого уровня – имя пользователя Windows

Остальное удалим и закроем операторную скобку - }. Должно получиться следующее:



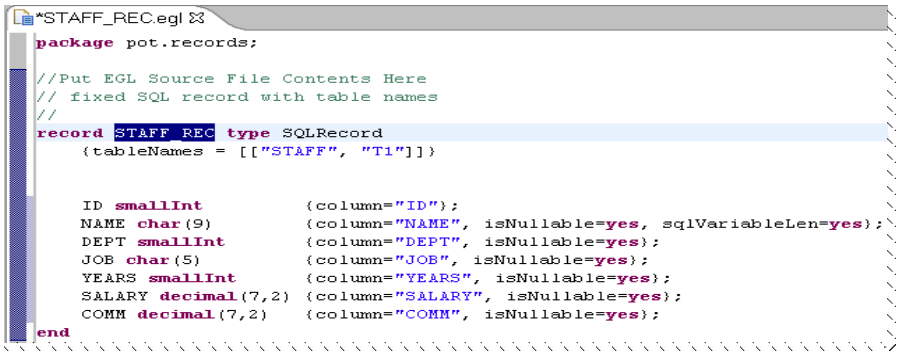
Теперь используя окно *Outline* (оно находится в нижнем левом углу) выберем **Retrieve SQL** (вернуть SQL) для **STAFF_REC** как показано ниже:



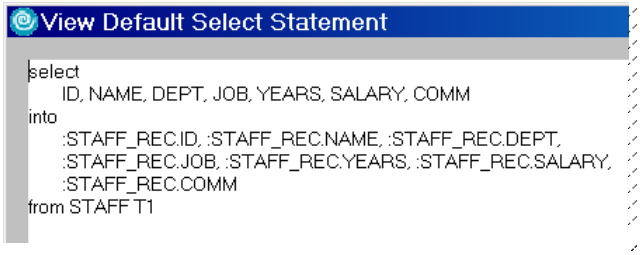
Результат возврата показан на следующем рисунке. Если возникла проблема с доступом к базе данных, то необходимо проверить настройки (Windows → Preferences → EGL → SQL Database Connections) userid и пароль.

Стоит отметить, что эта таблица DB2 находится на вашем компьютере, но она может находиться и на сервере и на мэйнфрейме.

В этом случае мы должны изменить только настройки, а не EGL код.



Теперь выведем SQL запрос на экран – в редакторе щелкнем правкой кнопкой мыши по **STAFF_REC** и выберем **SQL Record → View Default Select.** Получиться должно следующее:



Закроем это окно. И сохраним пакет package (**Ctrl+S).**

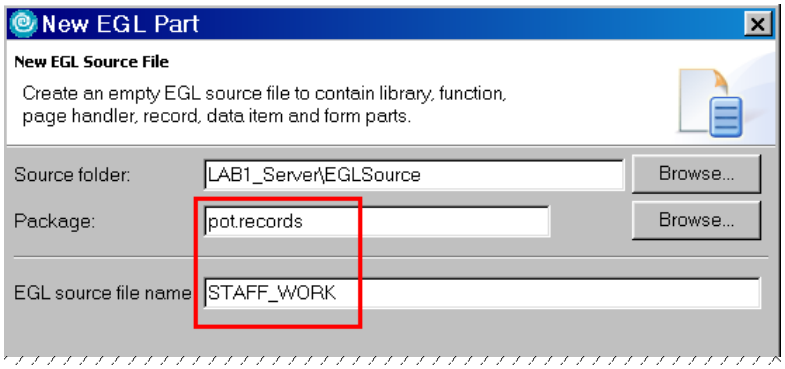
2.4.2. Создание второго модуля типа Data part (Working area EGL Data parts)

Создадим рабочую область, в которой сравнивается DB2 запись с записью, введенной нами и которая используется как область, в которой происходит связь между DB2 и клиентом. Назовем ее STAFF_WORK.

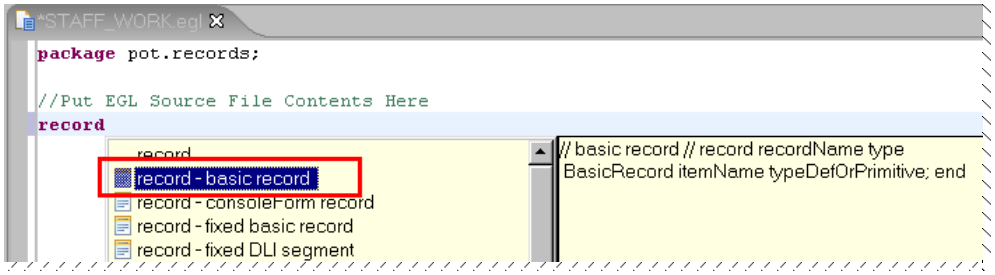
Для ее создания проделаем аналогичные шаги:

New → EGL → EGL Source File

Укажем уже созданный ранее package **pot.records** (нажатием кнопки Browse) и введем имя файла – STAFF_WORK.

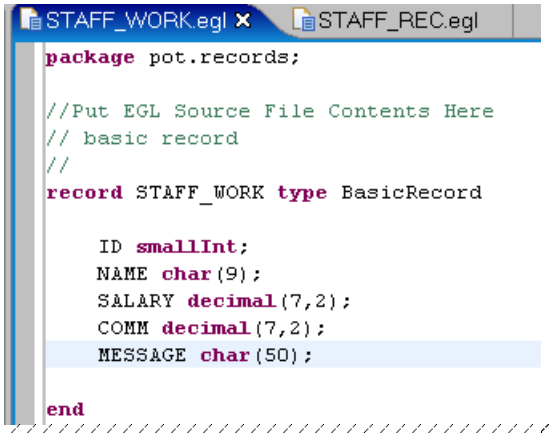


Опять используем автозаполнение (Cntl+Пробел), но на сей раз выбираем **record – basic record** .



Введем **STAFF_WORK** вместо recordName.

Далее опишем запись следующим образом (как показано на рисунке) снизу – просто перепишем поля.



Сохраним модуль STAFF_WORK, и закроем редактор (используем крестик в данном окошке сверху).

2.5 Создание модулей второго типа – EGL логики (EGL logic)

Модуль второго типа **Logic** - это главный логический модуль, который используется для генерации всех компонент:

- Java программы,
- Java-wrapper или
- Enterprise JavaBean session bean.

Назовем наш модуль TE01A и она будет получать рабочую область STAFF_WORK. Делаем следующее:

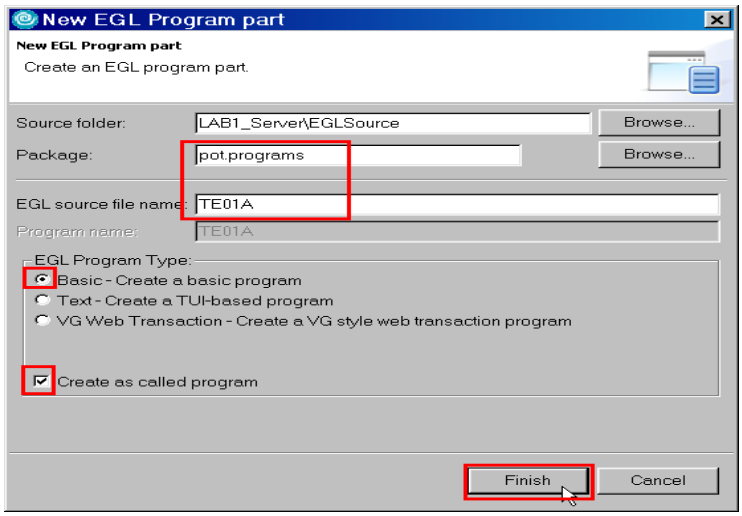
New → EGL → Program

Создадим новый package **pot.programs** - для этого просто введем это название в поле package

Ведем имя программы - TE01A

Выберем тип программы – **BASIC**

Выберем **Create as called program**



Следует отметить, что в созданном коде есть ошибки. Это нормально, так как нам необходимо ввести логику программы. Также стоит отметить, что вызываемая программа (called program) должна получать рабочую область (working area) STAFF_WORK и будет использовать SQL запись STAFF_REC, чтобы читать данные из базы DB2.

Нижеследующий текст программы необходимо ввести, можно использовать copy-paste из этой методички в электронном виде, которая у Вас есть на CDROMe.

```
package pot.programs;
import pot.records.*;  // импорт полей записи
// basic called program
program TE01A type basicProgram (anWork STAFF_WORK)  // указание рабочей области
anRecord STAFF_REC; // SQL Record Data declaration
 function main()
```

```
 counter  int;  // счетчик, увеличивающийся на 1 каждый раз при доступе к данным
  move anWork.id to anRecord.id; // перенос из рабочей области в запись в базе
// чтение DB2 таблицы
 try
  get anRecord usingKeys anRecord.ID ; // DB2 select используя Where ID = :ID
 end
 // проверка на существование записи
  if  ( anRecord is noRecordFound)    /* если не найдено*/
           anWork.MESSAGE = "Record  does not exist";
        else
           anWork.MESSAGE = " ";
           move anRecord to anWork;     /* перенос из записи в рабочую область*/
           counter = counter + 1; // увеличиваем счетчик
     end
 end
end
```

Для справки:

Немного о том, что делает эта программа:

- Данные, которые получает программа (кроме параметров), например STAFF_WORK, необходимо объявить - видим в тексте программы объявление SQL Record Data - anRecord STAFF_REC; STAFF_REC это *recordName* (имя записи) и anRecord переменная, которую используем в логике. Каждый раз, когда необходимо обратиться к записи SQL – пользуемся переменой *anRecord*.
- Так же определена переменная-счетчик counter int; эта переменная типа int. Объявление переменных в языке EGL схоже с объявлением переменных в Java. К счетчику будет добавляться 1 при каждом успешном вводе/выводе данных из базы.
- Очень часто используют строчные буквы для обозначения переменных, например anWork, anRecord и counter
- Первая инструкция для выполнения программы – это функция main(). Она будет перемещать полученный в рабочей области anWork id в поле id, определенное в записи SQL anRecord.
- Вторая инструкция – это get, которая определяет чтение данных из базы, используя ID в качестве ключа.
- Если данные получены из базы, то все поля anRecord будут перемещены в соответствующие поля рабочей области anWork. Счетчик увеличится на 1.
- Если запись не найдена в базе (код ошибки SQLCODE=100) сообщение будет отправлено в поле message рабочей области anWork.

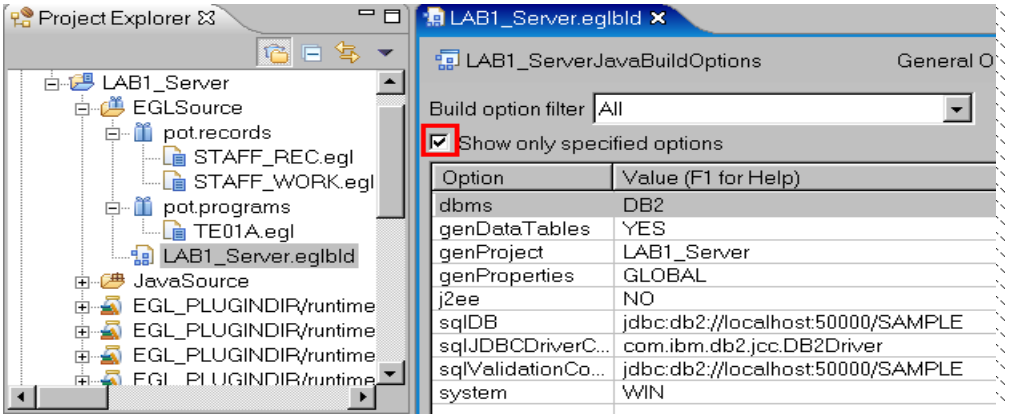
Сохраним программу **Ctrl + S.** Если не возникло ошибок, значит на этом этапе у нас все получилось. Закрываем все редакторы.

2.6 Просмотр модуля типа **Build**

Модуль третьего типа **Build,** Build descriptor или build part был автоматически создан, когда мы создали EGL проект. Build descriptor всегда требуется в EGL и контролирует процесс генерации кодов (Java или COBOL). Его настройки указывают, как сгенерировать и подготовить выходные данные EGL. В данной работе нам не нужно изменять Build descriptor, но его необходимо будет изменить, если нам понадобится сгенерировать COBOL (эта опция поддерживается в другой версии RAD, в WebSphere Developer for zSeries и iSeries).

Слева в иерархии откроем файл **LAB1_Server.eglbld** . Выберем **Show only specified options.** Подробности по опциям можно получить во встроенной справке (F1 для вызова).

Модуль Build descriptor для EGL Server показан ниже:



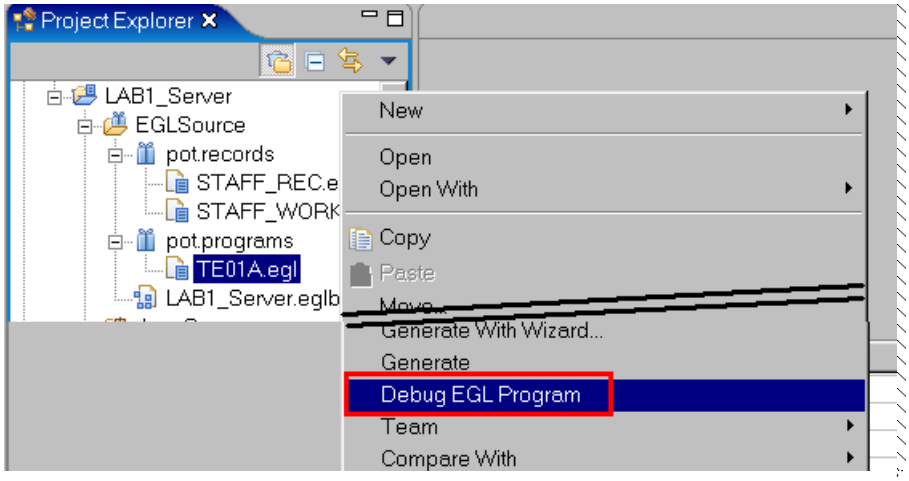
Позже нам потребуется изменить Build descriptor, когда мы будет делать нашу серверную часть программы как удаленный сервер, используя TCP/IP протокол.

В выпадающем списке system видно, что мы можем сгенерировать нашу программу под разные операционные системы, включая z/OS. Оставим выбранным пункт win и закроем окно.

2.7 Запуск и отладка программы

Теперь мы готовы к отладке нашей программы. Отладчик языка EGL позволяет проводить отладку без предварительной генерации выходных данных программы (output).

Нажмем правой кнопкой по файлу с программой TE01A.egl и выберем **Debug EGL Program**



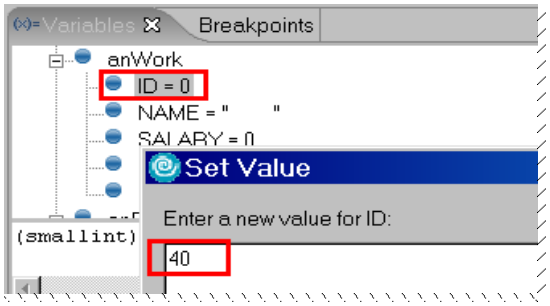
Согласитесь к Warning Message. Необходимый для отладчика параметр зададим на следующем шаге. Также согласитесь с предложением сменить вид.

Примечание: Следует отметить, что в нашей программе используются 3 переменные:

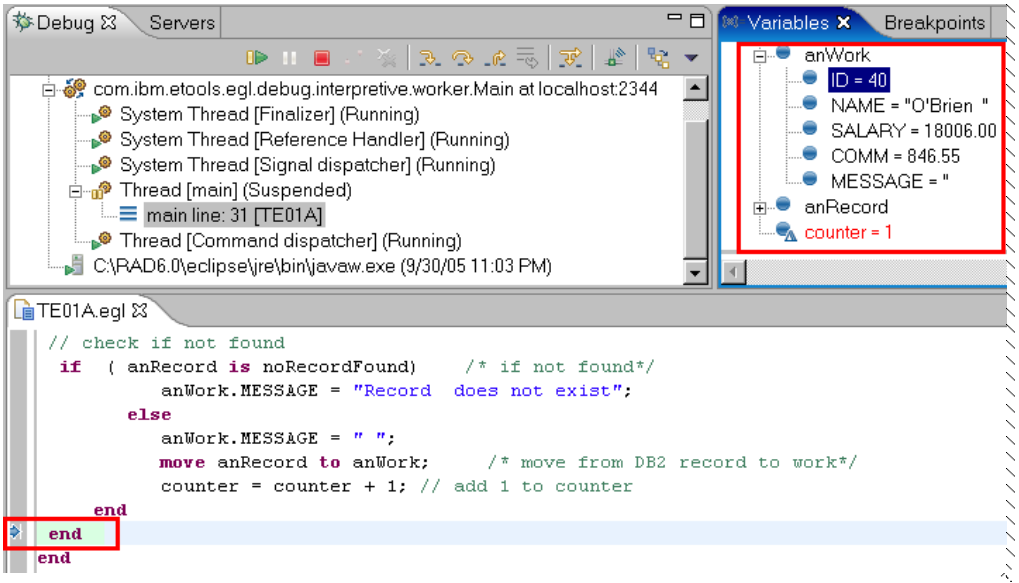
- *anWork*,
- a*nRecord* и
- *counter*.

2.7.1. Проверка существующей записи в базе данных

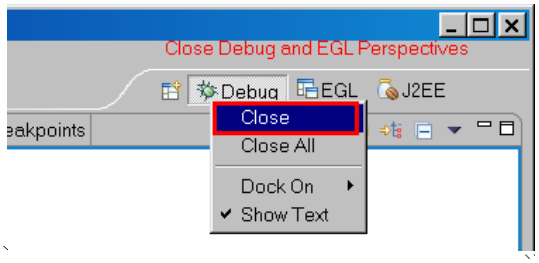
Используя окно Variables в верхнем углу, развернем **TE01A** и **anWork,** и присвоим полю ID значение, которое заведомо существует в базе данных (узнайте этот параметр у преподавателя).



Прогоняем программу кнопкой  (step into или кнопкой F5) до момента завершения запроса к базе и смотрим результаты.



Выполним программу до конца кнопкой  (Resume) и закроем отладчик.



Поздравляем с выполнением лабораторной работы!

3. *Контрольные вопросы*

3.1 Что такое TUI?

3.2 Что такое EGL?

3.3 Какой программный продукт используется для создания серверной части Web-приложения для большой вычислительной машины?

3.4 Какие типы модулей необходимы для работы программы-приложения, написанной на языке EGL?

3.5 Возможно ли автоматическое создание модуля Build?

Лабораторная работа № 5

Создание клиент-серверных приложений на языке EGL, клиентская часть программы

Цель работы:

Получить опыт создания простейшей клиентской программы WEB интерфейса (на основе JSP) для клиент-серверного приложения.

1. Теоретическая часть

В лабораторной работе № 3 мы создали TUI-клиент для серверной программы. В этой работе мы создадим еще один клиент, но уже на основе GUI (graphical user interface). Этот тип – наиболее популярен, так как для работы с такой программой необходим только браузер. Не нужен даже дистрибутив программы, и ничего не нужно устанавливать на компьютере клиента.

В этой работе мы создадим WEB приложение, которое будет использовать JAVA SERVER FACES (JSF) и будет вызывать серверную программу, созданную нами в лабораторной работе № 3.

Структура программы показана на рисунке 1:

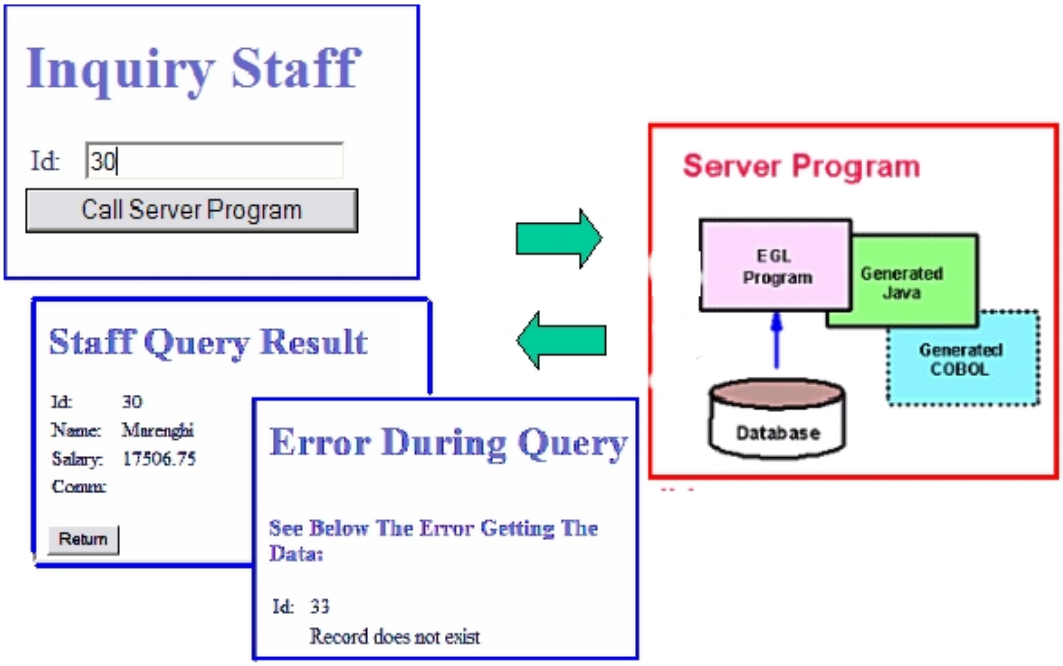


Рисунок 1. Структура программы (клиентская и серверная части)

Поясним принцип работы программы:

- пользователь вводит ID и нажимает кнопку, запуская тем самым серверную часть программы, и передает ей ID через рабочую область (working area)
- серверная программа считывает из базы данных DB2 записи, используя поле ID в качестве первичного ключа
- если запись в базе с таким ID найдена, то серверная программа перемещает поля этой записи (Name, Salary, Commission) в рабочую область (working area) и далее на страницу Result JSP
- если запись не найдена, то серверная программа отправляет сообщение об ошибке в рабочую область и далее оно отображается на Error JSP странице нашего интерфейса.

Мы могли бы создать все компоненты в одном WEB проекте, но причина это не является целью лабораторной работы. Наша задача – создать один проект с серверной частью и второй проект с клиентской часть. Нам необходимо четкое разделение серверной части программы (согласно модели MVC - model) от клиентской части (согласно модели MVC – view/controller).

Примечание: Следует отметить, что в этой работе мы будем использовать JSF технологию. Семейство программных продуктов Ration Application Developer

(RAD) имеет необходимые инструменты для JSF, они называются faces-components. JSF – предоставляет GUI компоненты и возможность разработки динамических WEB интерфейсов.

План работы:

- Создать WEB проект для работы с JSF/EGL
- Создать JSF/EGL компоненты
- Протестировать и отладить программу, используя EGL отладчик

2. Практическая часть

2.1 Создание проекта с клиентской частью

Запускаем WDz v6.

Выбираем необходимое нам рабочее пространство (workspace см. рисунок 2).

ВАЖНО! НЕ ставим флаг «использовать это пространство по умолчанию».

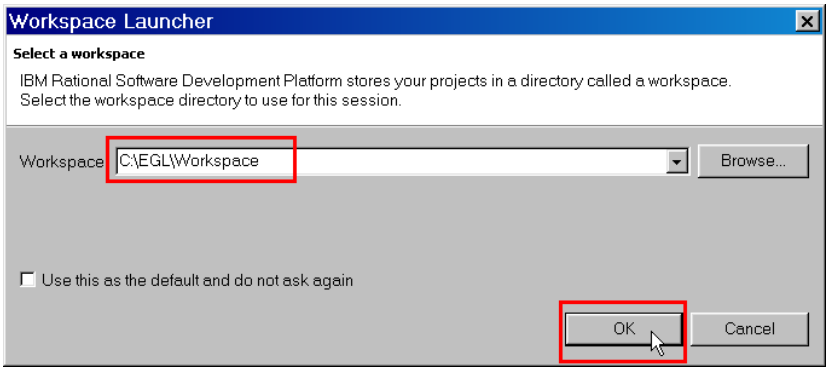


Рисунок 2. Выбор рабочего пространства

WDz v6 запустился. Стоит отметить, что при использовании VMWare загрузка может занять довольно длительное время.

Создадим новый WEB проект:

File → New → Other → развернем пункт **EGL → EGL Web Project →** нажмем **Next,** как показано на рисунке 3.

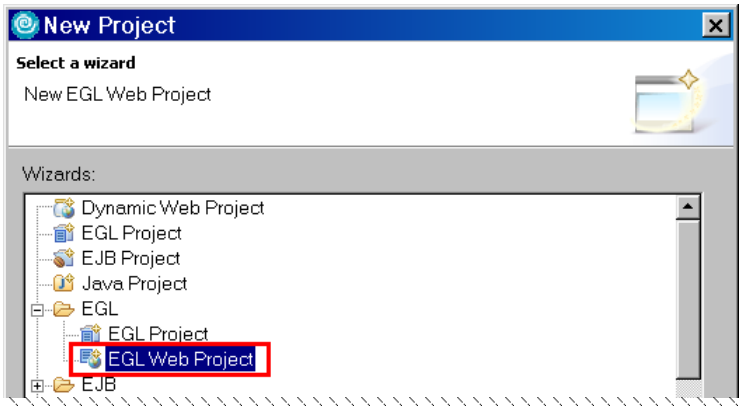


Рисунок 3. Создание нового Web-проекта

Введем имя проекта (например LAB2_JSF_Client), обязательно выберем автоматическое создание build descriptor (см. рисунок 4). И в поле SQL Connection введем jdbc/SAMLE.

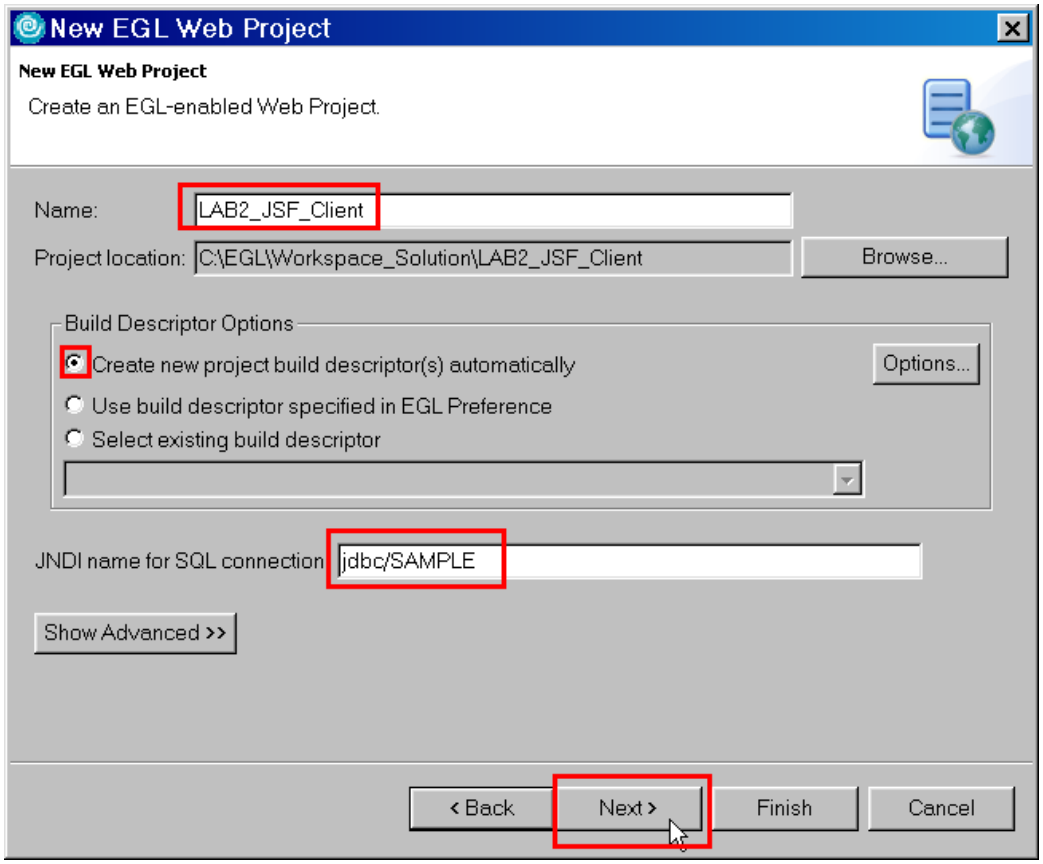


Рисунок 4. Ввод имени проекта и соединение с базой данных

В появившемся окне (рисунок 5) снимем флаг с **WDO Relational database runtime.**

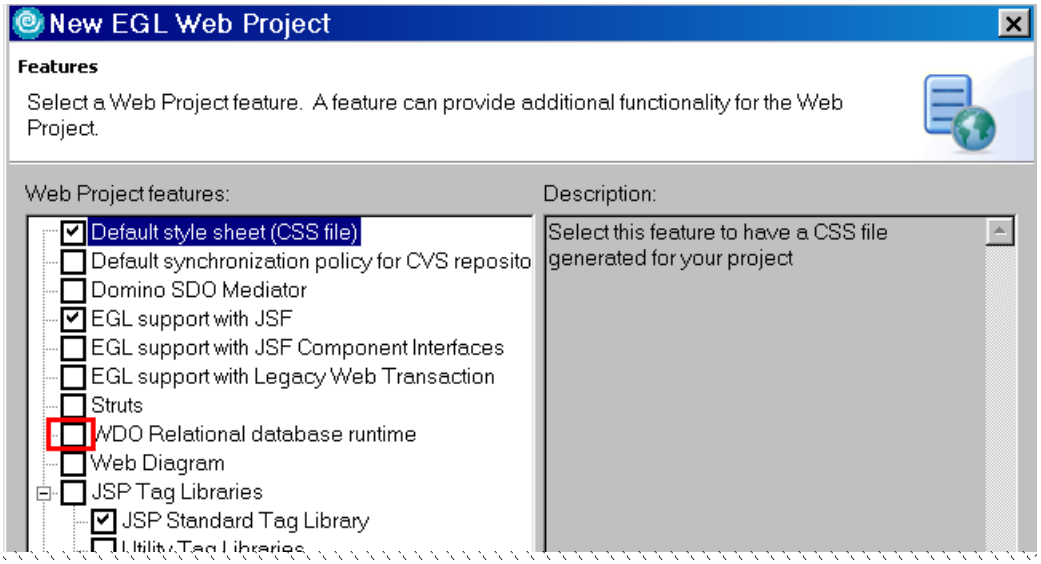


Рисунок 5

После загрузки – согласимся с изменением вида.

Слева в иерархии – развернем наш проект, и увидим, что EGL build descriptor создан и имеет такое же имя, как и проект (рисунок 6).

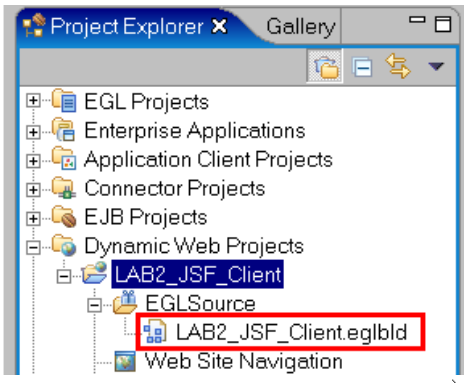


Рисунок 6

Двойным щелчком мыши откроем его и поставим флаг **Show only specified options** (рисунок 7)**.** Стоит отметить, что поддержка JAVA включена автоматически. J2EE – yes.

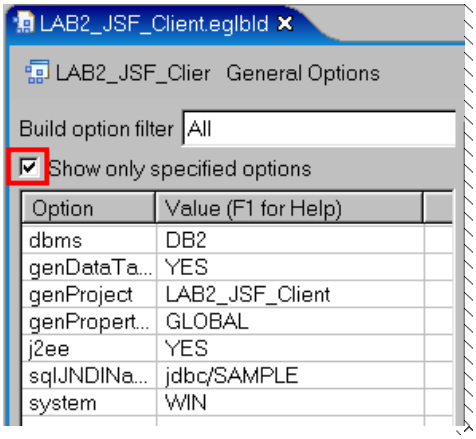


Рисунок 7

Закроем редактор.

2.2 Создание JSF/EGL компонентов.

Перед созданием компонентов мы должны убедиться, что компоненты, которые мы создали для серверной программы – видимы, т.к. мы хотим использовать уже созданную рабочую область для передачи данных между клиентом и сервером.

2.2.1. Редактирование EGL WEB project build path

Необходимо включить серверную программу в наш проект.

- Используя Project Explorer, нажмем правой кнопкой на наш проект и выберем Properties.
- Выберем Build Path
- Отметим в списке проект LAB1_server (рисунок 8).

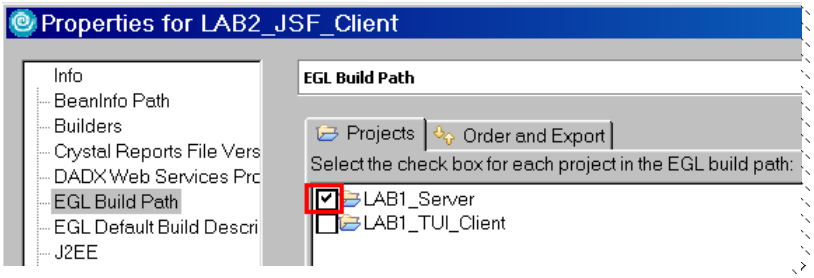


Рисунок 8.

Вот теперь мы готовы к созданию Java Server Pages (jsp). Мы создадим 3 страницы:

- страницу ввода LAB2_input,
- страницу результатов (LAB2_Result) и
- страницу, которая говорит об ошибке (LAB2_Error).

2.2.2. Создание LAB2_input.jsp

Это страница ввода. Пользователь будет вводить ID и затем нажимать на кнопку для вызова серверной программы. Правой кнопкой нажимаем на проект, затем выбираем **New → Faces JSP File** (рисунок 9)

В открывшемся мастере введем имя страницы и согласимся с папкой по умолчанию.

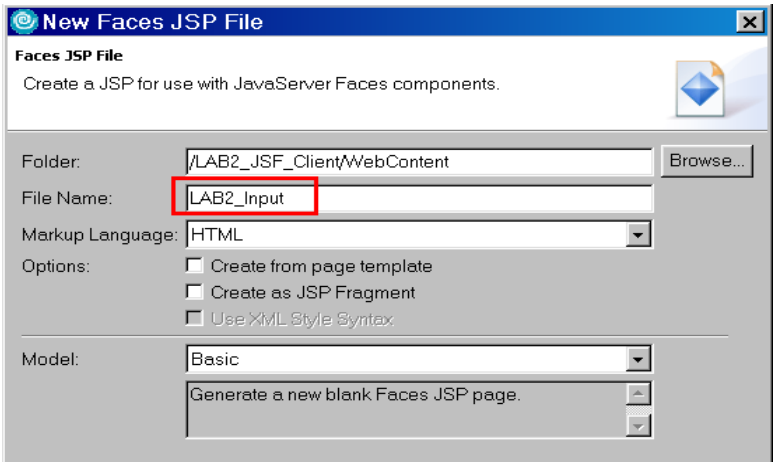


Рисунок 9

JSP файл создан и соответствующий page handler компонент сгенерирован. (этот компонент нужен для правильной интерпретации студией файла со страницей.) Файл открылся в Page Designer – редакторе страниц. Раскроем package под названием pagehandlers в папке EGL Source и убедимся, что файл *LAB2_Input.egl* сгенерирован. Введем название страницы – вместо Place content here (рисунок 10).

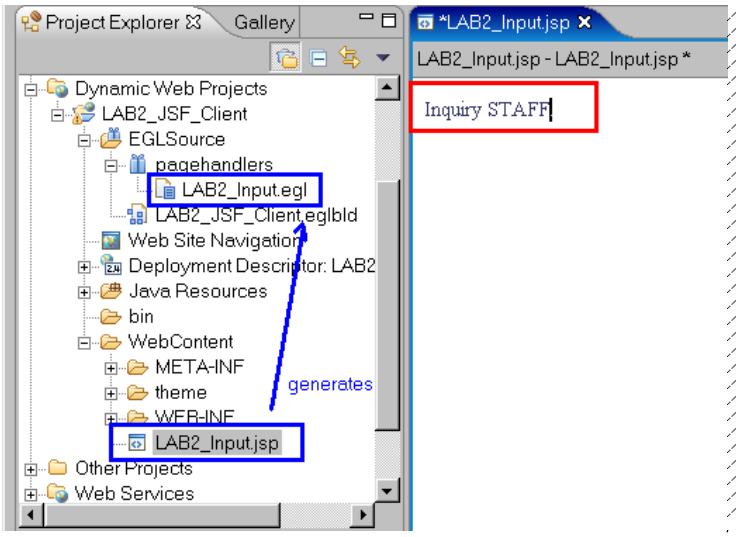


Рисунок 10

Итак, редактор – представляет собой графический редактор, в котором есть возможность создавать Face Components. Здесь мы можем использовать ранее созданные компоненты (рабочая область).

Используя редактор, сделаем следующее:

Пока указатель мыши находится в области названия страницы, щелкнем мышью и выберем Properties, затем выберем Heading 1 в качестве типа, чтобы получить укрупненный шрифт (рисунок 11).

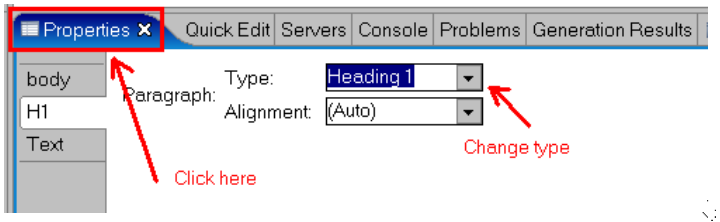


Рисунок 11

В «палитре» выберем пункт EGL и перетащим пункт Record в свободную область, как показано на рисунке 12:

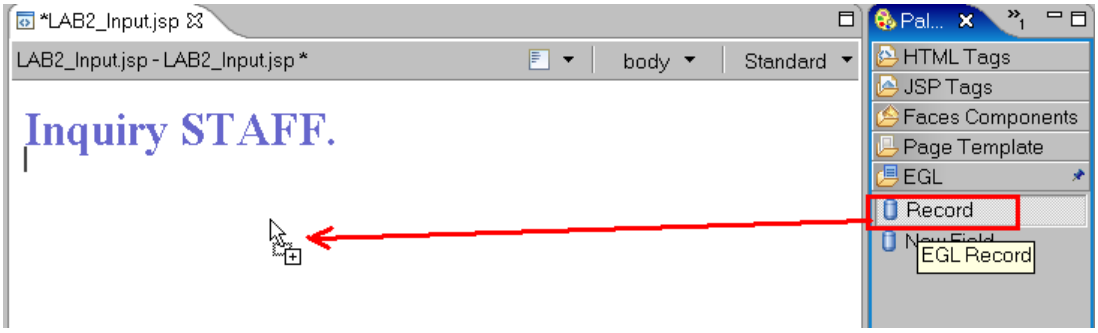


Рисунок 12

Далее появится окно выбора записи, выберем знакомую нам STAFF_WORK (рисунок 13).

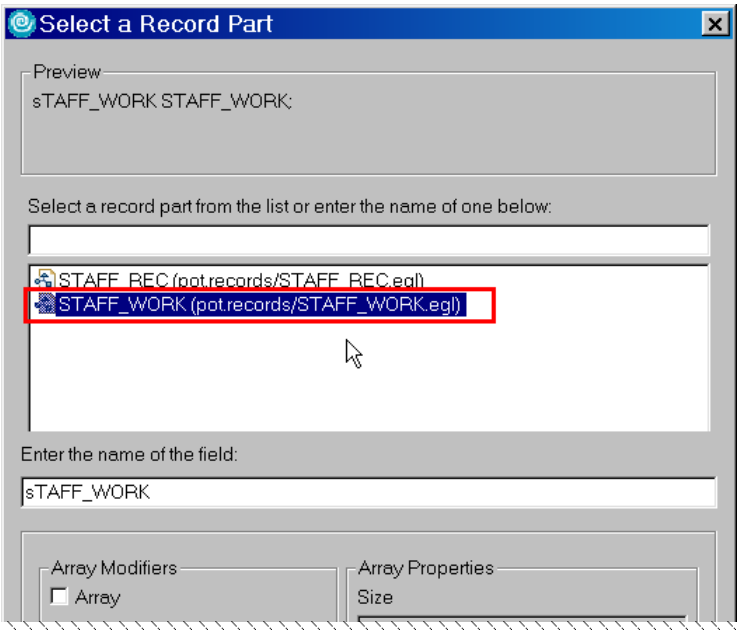


Рисунок 13

Далее появится окно Insert Control (рисунок 14). В нем выберем пункт Updating An Existing Record. Выберем поле ID как поле ввода и снимем выбор с остальных полей.

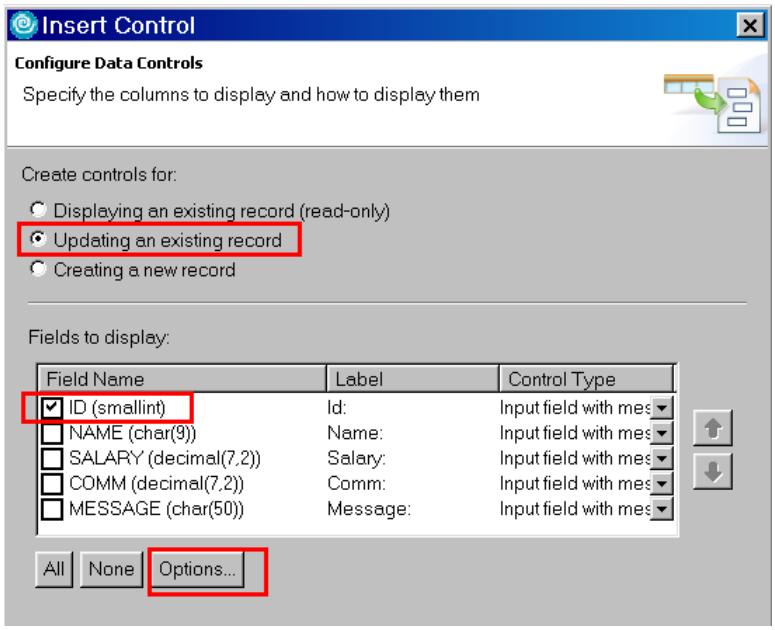


Рисунок 14

Далее нажмем Options (рисунок 15). Выберем Submit Button и введем название этой кнопки Call Server Program. Снимем флаг с Delete Button.

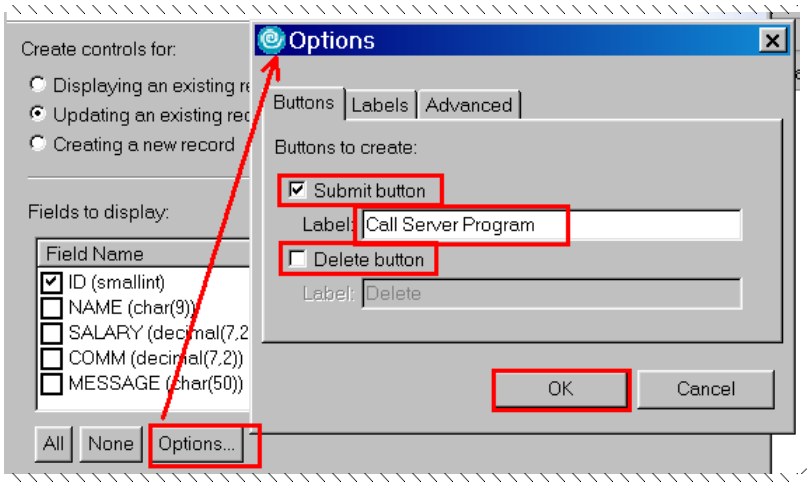


Рисунок 15

Итак, мы почти создали нашу первую страницу. Сделаем еще несколько настроек:

Удалим 

, расположенный рядом с кнопкой запуска серверной программы (но не перепутайте с Error Message for text1) – просто выделим и нажмем кнопку Delete.

Выберем EGLID поле (рисунок 16) и зайдем в его настройки и заглянем на страницу Validation, чтобы указать параметры поля ID: длина от 1 до 3х символов, Только цифры.

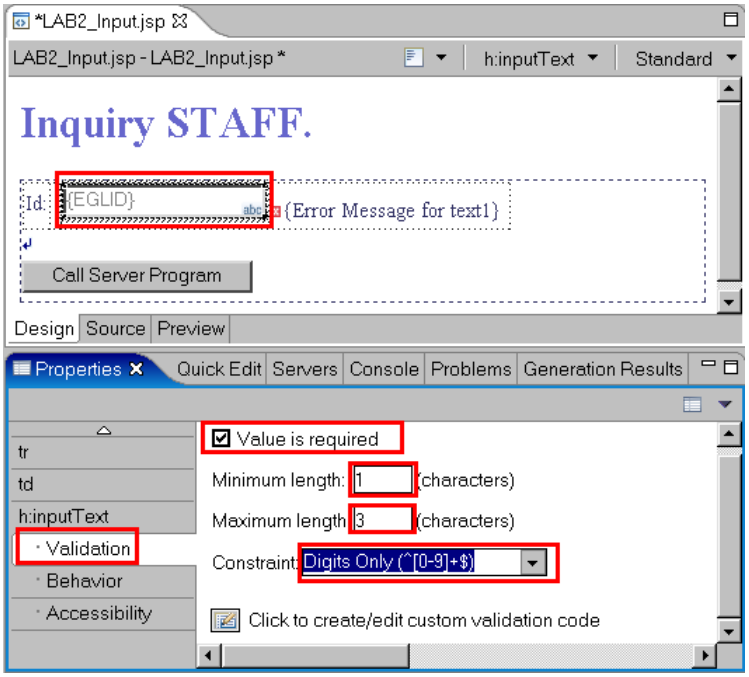


Рисунок 16

Сохраним страницу и закроем редактор. Результат показан на рисунке 17.

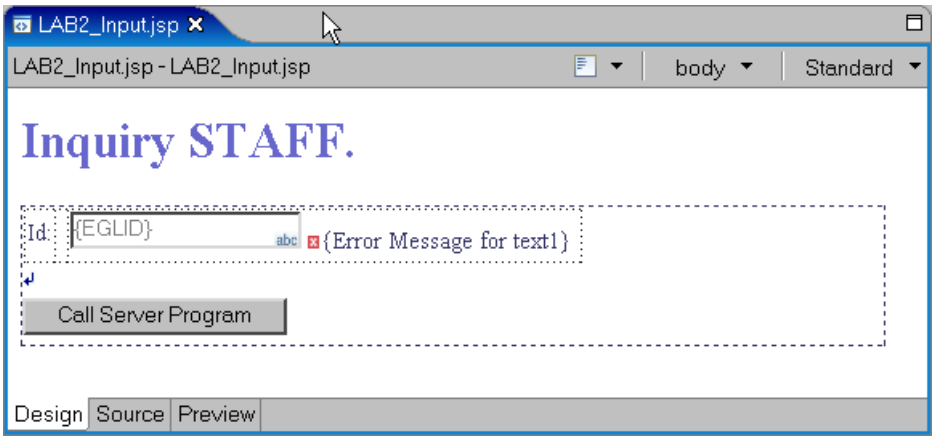


Рисунок 17

2.2.3. Создание страницы LAB2_Result.jsp

Эта страница будет отображать результат успешного запроса к базе. Также как и для предыдущей страницы выполняем аналогичные шаги. Правой кнопкой мыши щелкнем по нашему проекту, выберем **New → Faces JSP File.** Назовем эту страницу LAB2_Result.

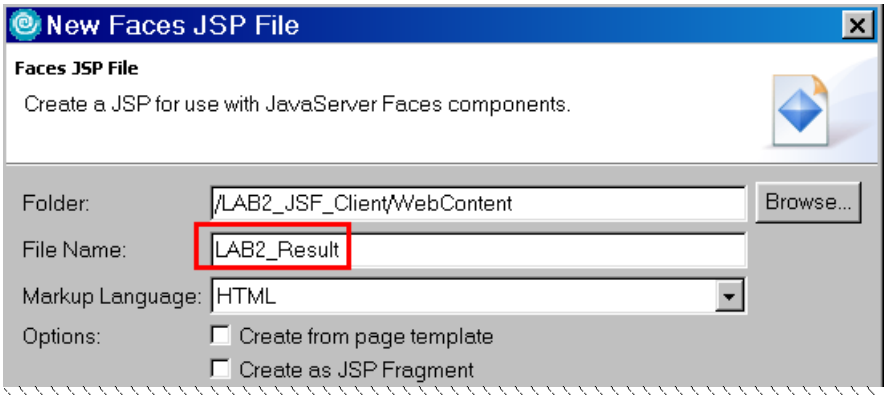


Рисунок 18

Введем название STAFF Query Results. Аналогично изменим шрифт.

Перетащим из «палитры» Record, в окне выберем запись STAFF_WORK (рисунок 19).

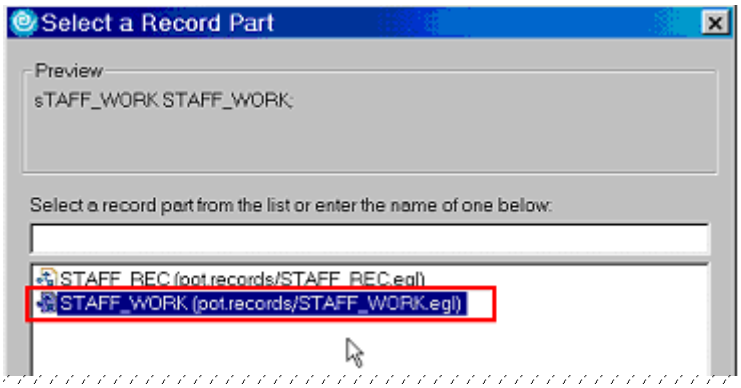


Рисунок 19

В появившемся окне Insert Control выберем Displaying an existing record, выберем все поля, кроме Message. И нажмем Finish (рисунок 20).

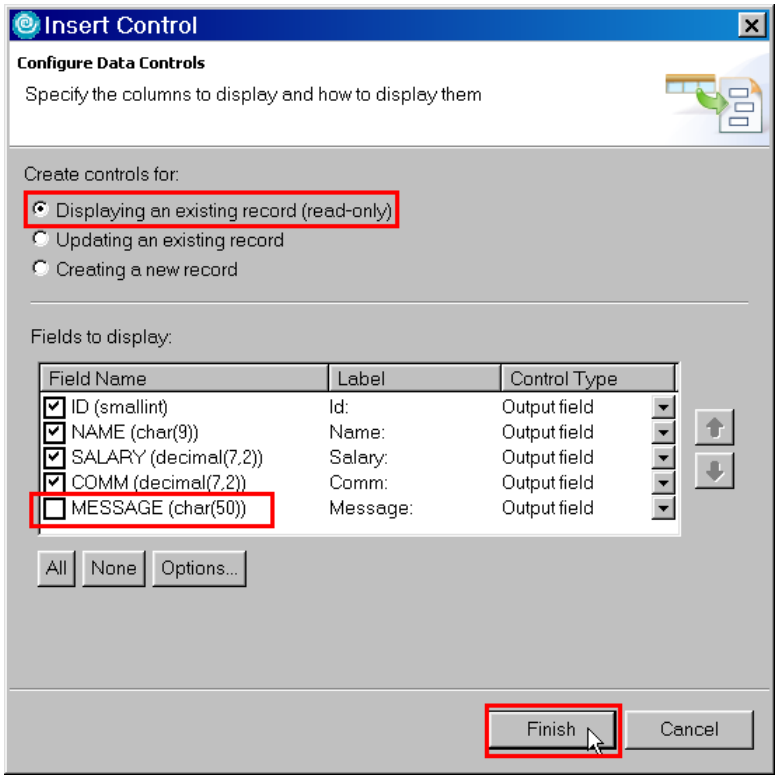


Рисунок 20

Далее – удалим поле 

Из Faces Components на «палитре» добавим кнопку Command Button () под таблицу на странице и присвоим ей имя Return в пункте меню Properties->Display Options (рисунок 21).

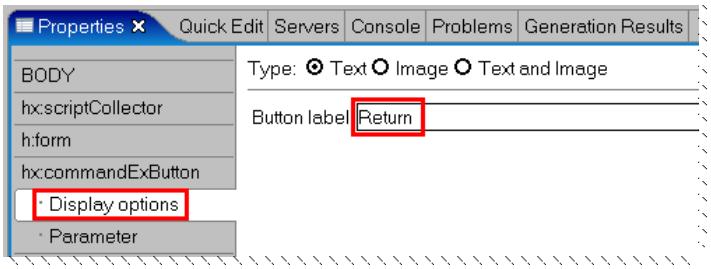


Рисунок 21

Сохраним страницу и закроем редактор. Следует отметиться, что в этот момент произошла генерация – EGL (page handler) и Java коды были созданы. Игнорируем возможные сообщения об ошибках и предупреждения (warnings), у вас в результате не должно быть ошибок. Результат показан на рисунке 22.

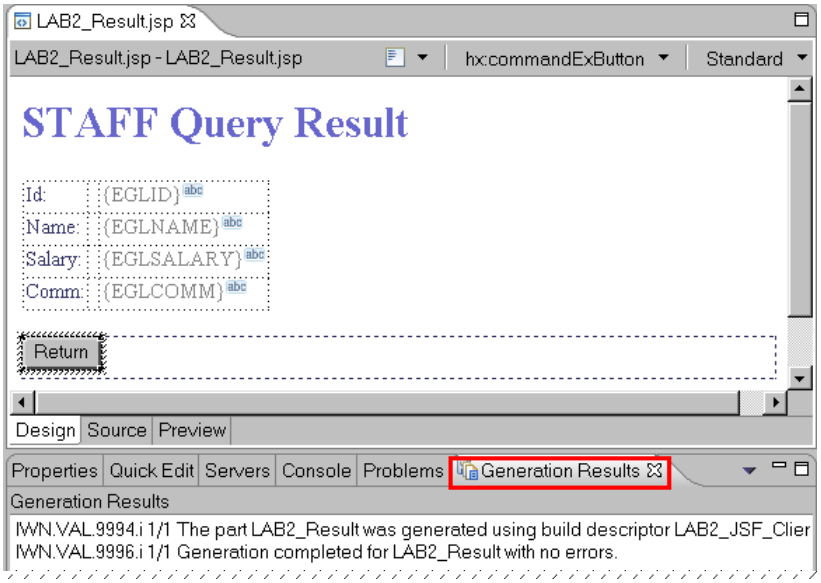


Рисунок 22

2.2.4. Создание страницы LAB2_Error.jsp

Эта страница будет показывать сообщение об ошибке, в том случае, если ID, посланная пользователем, не найдена в базе.

Также как и для предыдущих страниц – создайте новую, назовите ее LAB2_Error (рисунок 23).

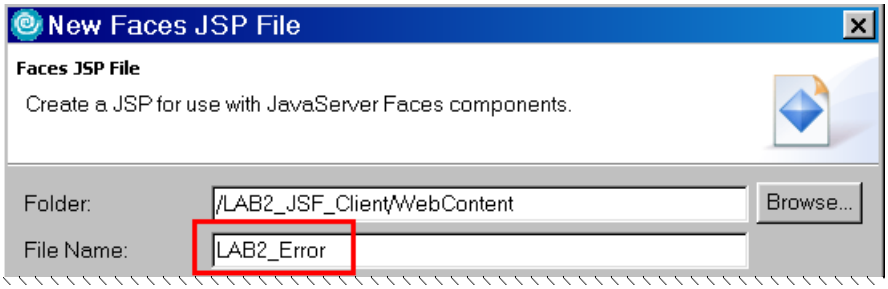


Рисунок 23

В редакторе – измените название страницы на Query Error.

Передвинем указатель мыши ниже названия и добавим break lines – просто нажав Enter. Добавим строку и напишем “See below the error message during query”. В свойствах названию присвоим Heading 1, а тексту Heading 4.

Рисунок 24

Перетащим Record из «палитры». Укажем запись – STAFF_WORK.

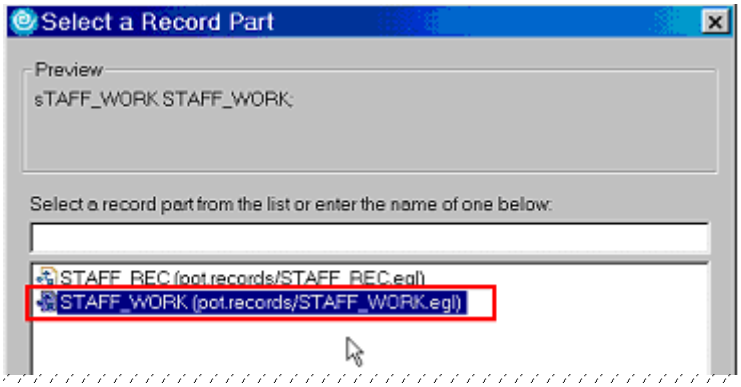


Рисунок 25

В окне Insert Control выберем Displaying an existing record. Выберем только 2 поля – ID и Message(у него очистим Label).

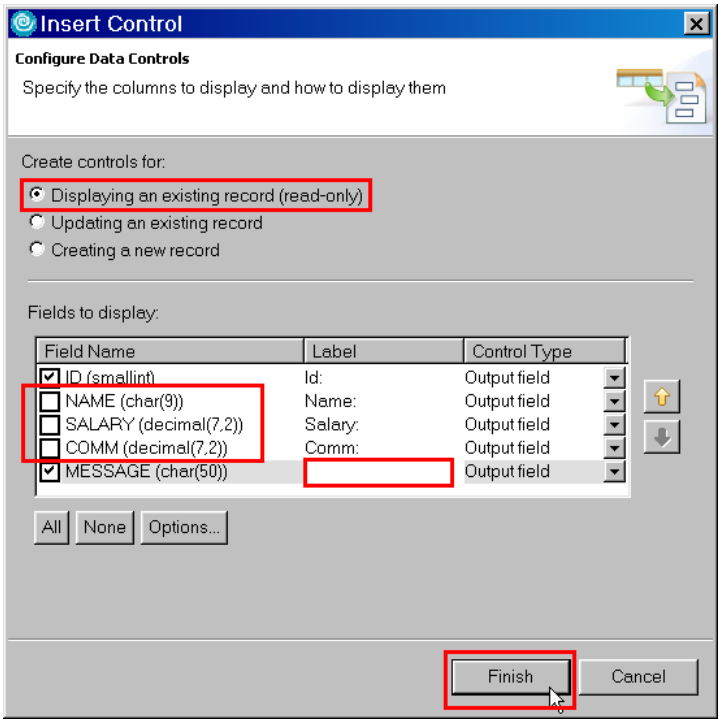


Рисунок 26

Как обычно, удалим 

. Из Faces Components «палитры» перетащим кнопку Command Button и присвоим ей имя Return.

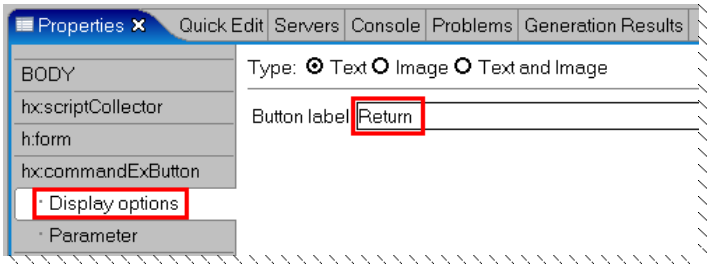


Рисунок 27

Сохранимся и закроем окно редактора.

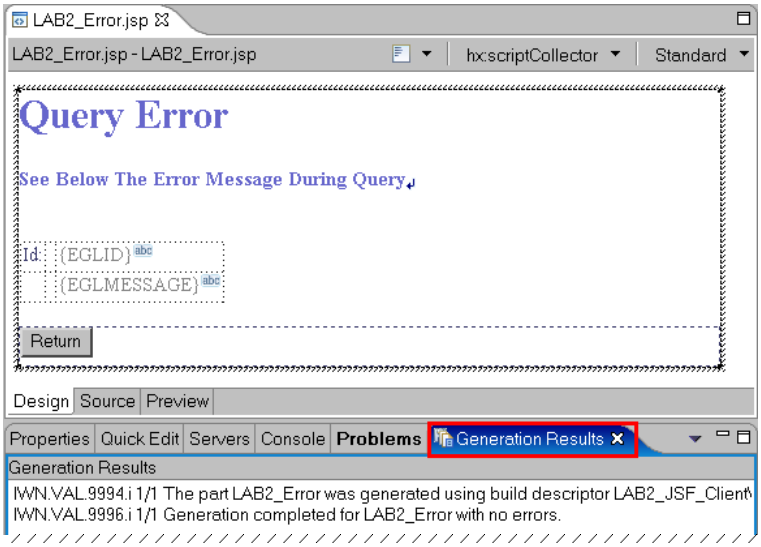


Рисунок 28

2.2.5. EGL Page Handler

Page Handler part – программа, отвечающая за управление каждой JSP страницей. Она не должна содержать бизнес логики. Она реализует controller в MVC модели.

Код этой программы может включать небольшие проверки правильности данных и т.д. Рекомендуется запускать другие программы для выполнения сложной бизнес-логики, иначе мы нарушаем правила MVC программирования. Например доступ к базе данных должен быть реализован в вызываемой программе, собственно мы так и сделали в работе №1.

Графическое представление модели MVC и модели JSF/EGL показано на рисунке 29.

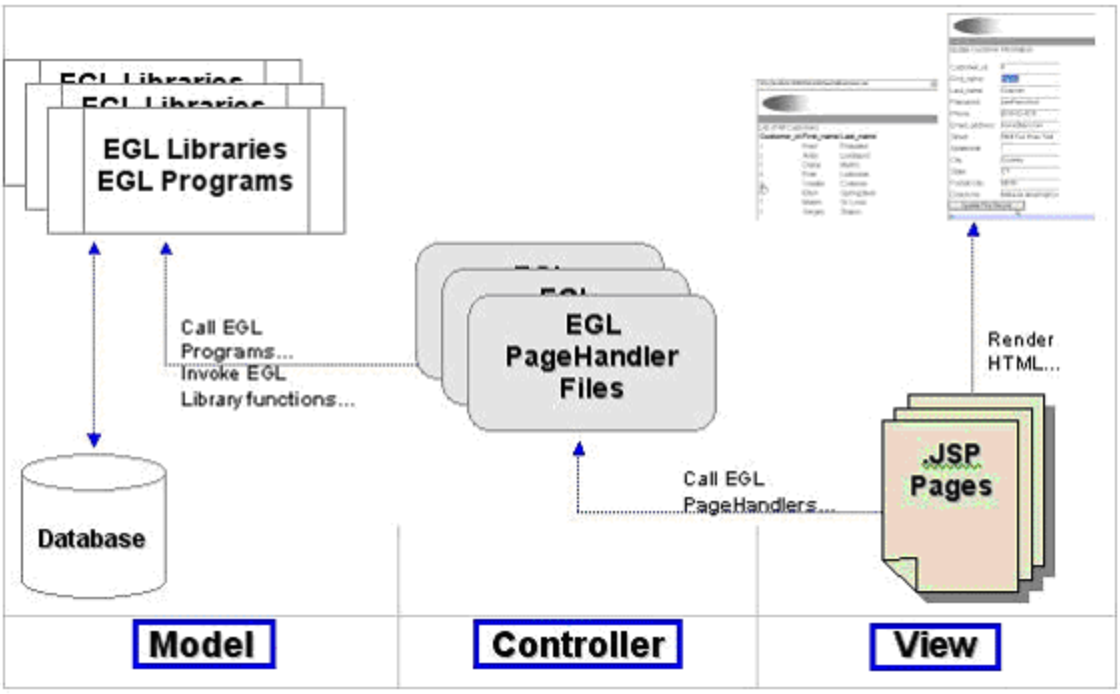


Рисунок 29

2.2.6. Корректирование EGL Page Handler для LAB2_Input.jsp

Нам необходимо дописать код обработки нажатия на кнопку на странице. Для этого:

В Project Explorer в нашем проекте открываем папку WebContent.

Используя Design View, нажимаем на кнопку Call Server Program и нажимаем на страницу Quick Edit, расположенную внизу.

Используя Quick Edit View, нажмем на Command на панели событий (event pane) она слева, затем щелкнем на редакторе (правая панель), в тексте появится функция function button1action()

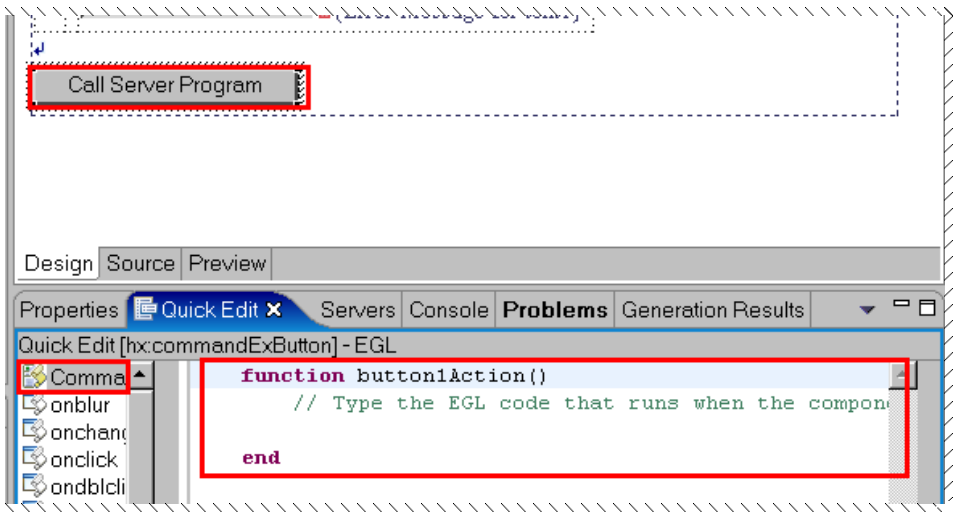


Рисунок 30

Удалим комментарии и введем код page handler, пользуясь уже известными нам функциями автозаполнения. Наберем CALL и нажмем Ctrl+Space

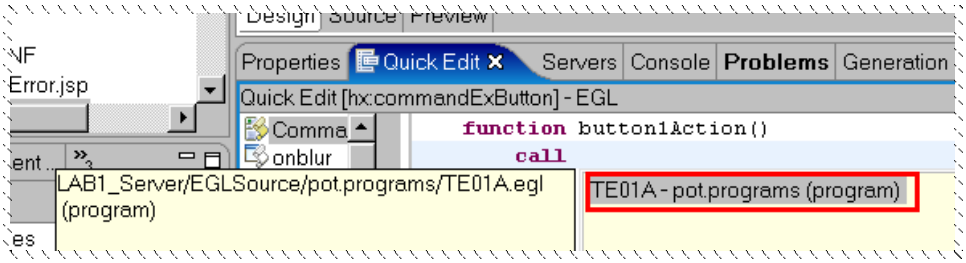


Рисунок 31

Далее введем текст как показано на рисунке ниже.

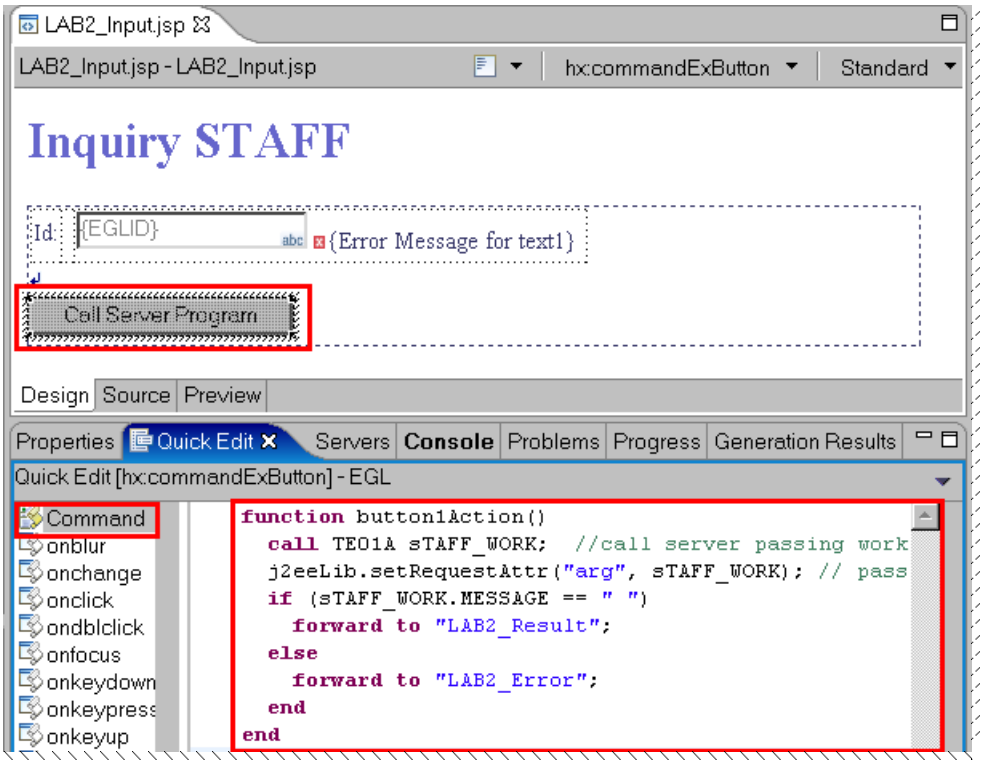


Рисунок 32

Итак, когда нажата кнопка серверная программа TE01A вызвана и значение ID прошло рабочую область sTAFF_WORK. После того, как вызов был осуществлен, рабочая область будет иметь либо поля найденной в базе записи, либо сообщение об ошибке. Системная функция setRequestAttr(key, argument) использует специальный ключ, чтобы поместить необходимый аргумент в запрашиваемый объект (sTAFF_WORK). Мы получим аргумент на страницах позже, после использования команды forward. Для получения значений мы будем использовать функцию getRequestAttr. Если поле sTAFF_WORK.Message – пустое (нет ошибок), то мы отобразим страницу с результатом (используем команду forward). Иначе отображаем страницу с сообщением об ошибке.

Сохраняем и закрываем.

В окне состояния вы должны увидеть следующее:

IWN.VAL.9994.i 1/1 The part LAB2_Input was generated using build descriptor LAB2_JSF_ClientWebBuildOptions from file

LAB2_JSF_Client/EGLSource/LAB2_JSF_Client.eglbld.

IWN.VAL.9996.i 1/1 Generation completed for LAB2_Input with no errors.

2.2.7. Корректирование EGL Page Handler для LAB2_Result.jsp

Для страницы результатов нам также необходимо добавить код. Эта страница будет получать управление, когда поле сообщения об ошибке будет пустым. Это значит, то запись найдена.

Откроем страницу – и нажмем правой кнопкой а любом свободном месте страницы – и выберем Edit Page Code.

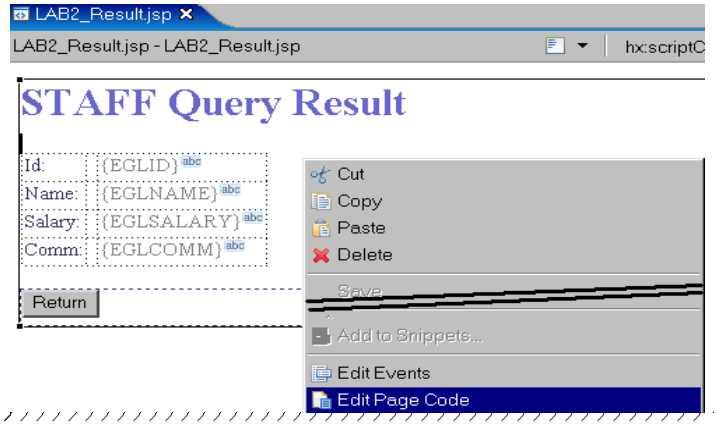


Рисунок 33

В открывшемся редакторе найдем функцию onPageLoad() и введем код, как показано ниже.

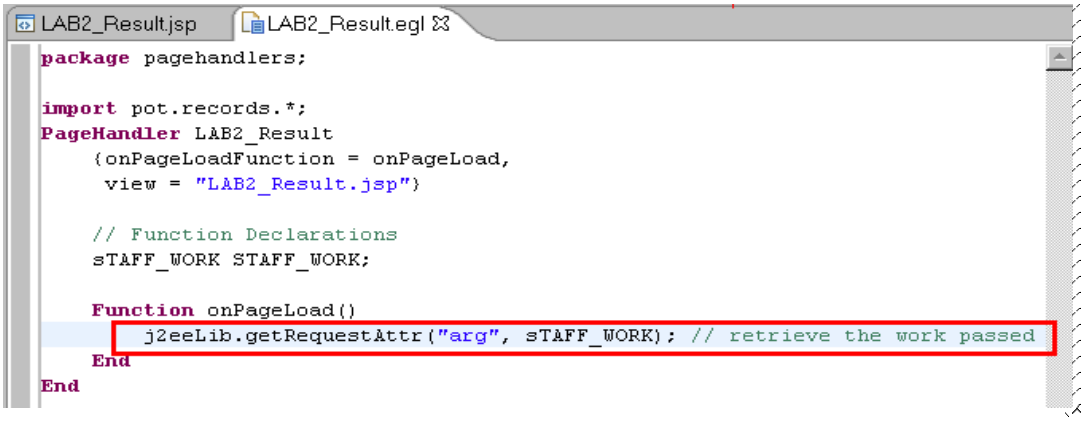


Рисунок 35

Сохраняемся и закрываем редактор.

Системная функция j2eelib.getRequestAttr использует специальный ключ (“arg”), чтобы передать аргумент от запрашиваемого объекта указанной переменной (sTAFF_WORK).

Осталось написать обработку кнопки для этой страницы. Делаем это аналогично предыдущей кнопке на предыдущей странице.

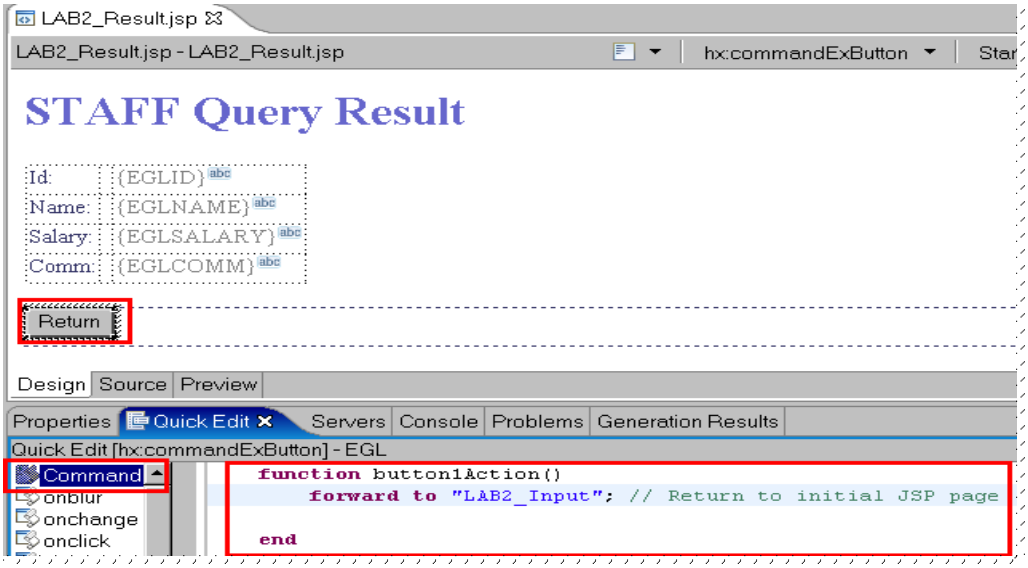


Рисунок 36

На нее «повесим» переход к странице ввода. Сохранимся и закроем редактор.

2.2.8. Корректирование EGL Page Handler для LAB2_Input.jsp

Для последней страницы добавим код кнопки и самой страницы. Страница будет получать управление, когда сообщение об ошибке не будет пустым. Это значит, что ID не найден в базе.

Итак – начнем с кнопки – запрограммируем ее как обычно на переход на страницу ввода.

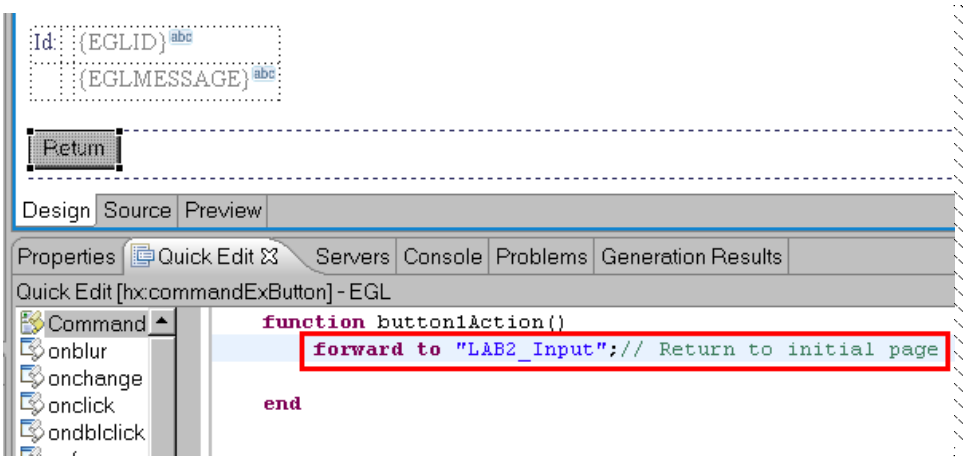


Рисунок 37

И дополним код самой страницы:

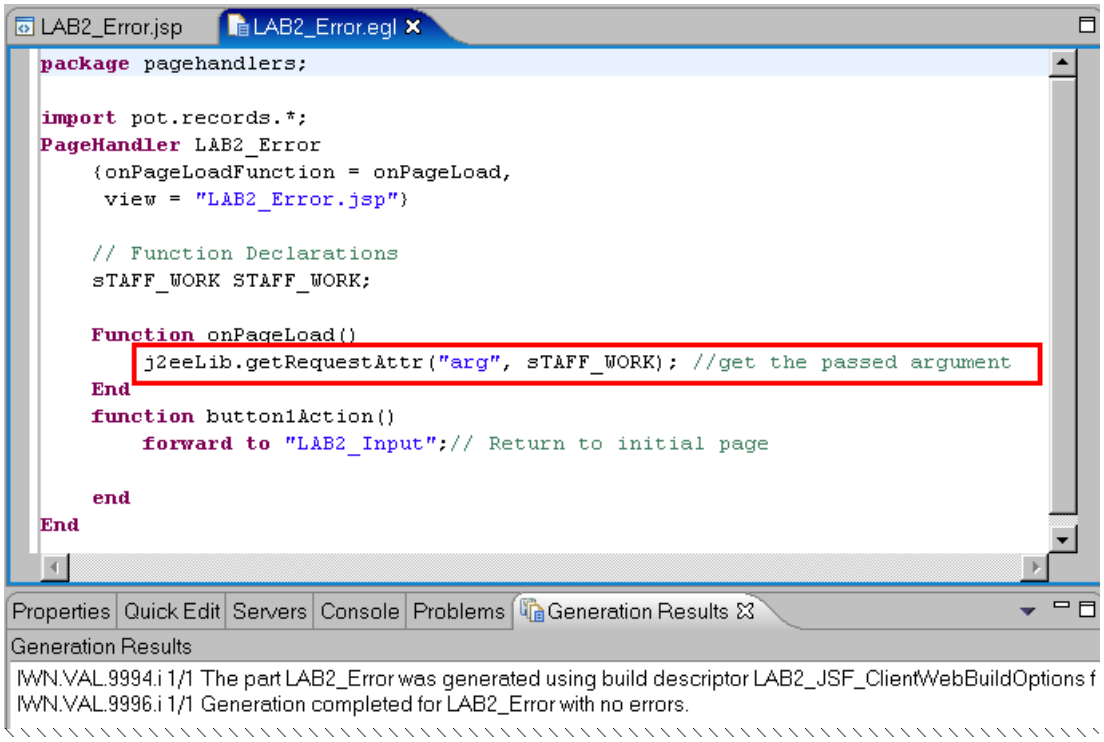


Рисунок 38

Сохраняемся и выходим.

2.3 Тестирование и отладка программы

Итак, наша серверная программа уже отлажена. Теперь мы должны отладить EGL PageHandler без отладки сгенерированного Java кода (чтобы не загромождать наше представление о программе).

Мы можем использовать компонент студии WebSphere Application Server test environment. Нам необходимо:

- Подготовить сервер к EGL WEB отладке
- Выбрать Debug build descriptor по умолчанию
- Добавить точки останова в программу
- Протестировать программу

2.3.1 Подготовка сервера

В комплект WDz входит модуль WebSphere Application Server Test Environment. Его мы сейчас и настроим, чтобы на нем выполнить тестирование и отладку нашей программы. Подготовительный шаг должен быть выполнен всего один раз для каждого сервера.

Перейдем на закладку Servers (она находится снизу). Нажмите правой кнопкой на **WebSphere Application Server v6.0** и выберите Debug.

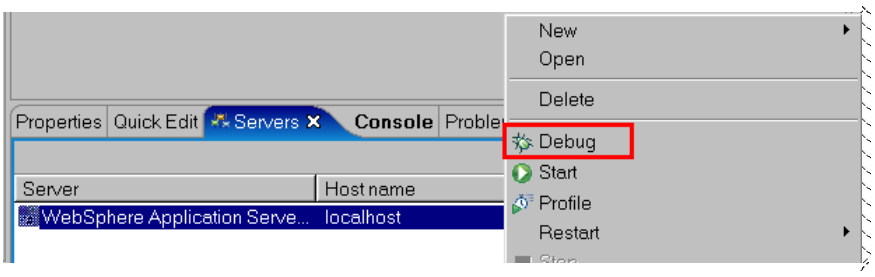


Рисунок 39

Сервер запустится, о чем вас проинформирует строка Status. Когда Status будет debugging – можно продолжать.

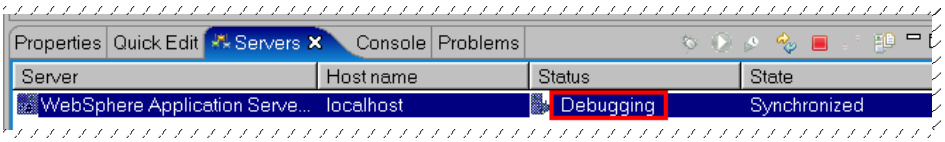


Рисунок 40

Нажимаем правой кнопкой по имени сервера и выбираем **Enable/Disable EGL Debugging.** Сервер перезапустится и появится окно, говорящее, что EGL отладка включена.

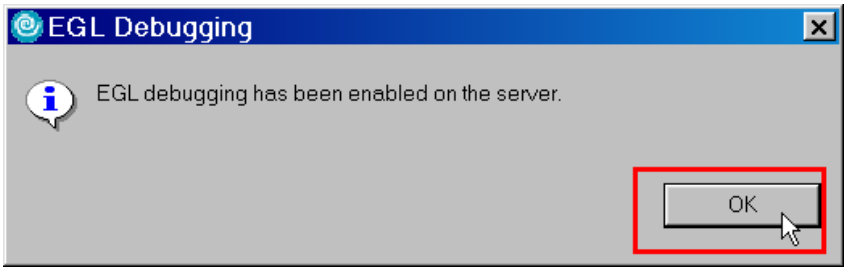


Рисунок 41

2.3.2 Добавление точек останова.

Добавим 3 точки: 2 в EGL Page handler до и после вызова серверной программы и одну на первой строке серверной программы.

Открываем файл **LAB2_Input.egl** либо напрямую, найдя его в иерархии, либо открыв соответствующую страницу и перейдя в режим редактирования ее кода. Поставим точки останова как показано ниже:

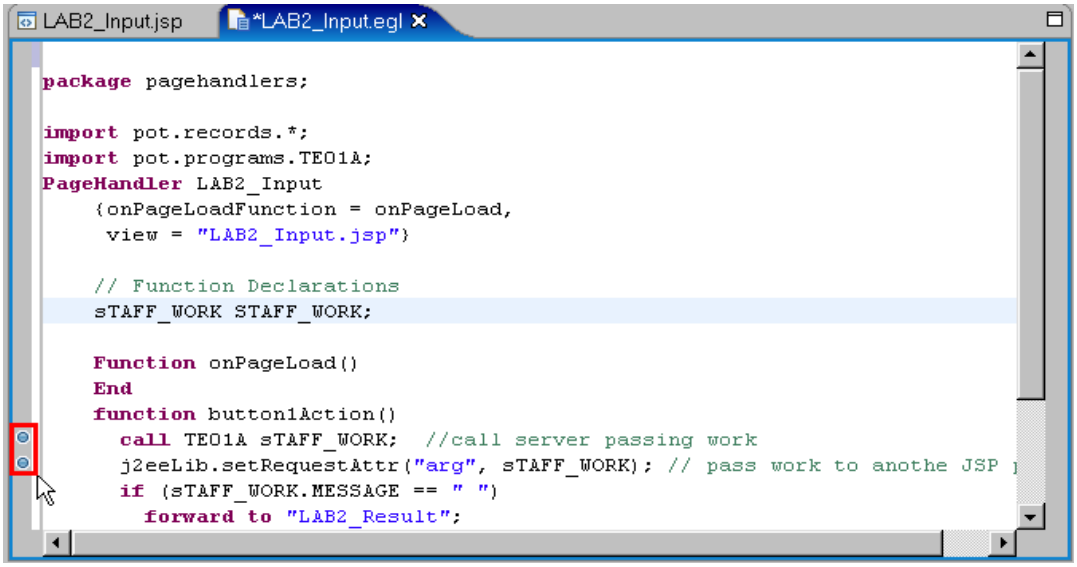


Рисунок 42

Закроем окно. Теперь откроем файл **TE01A.egl** , он находится в проекте первой работы.

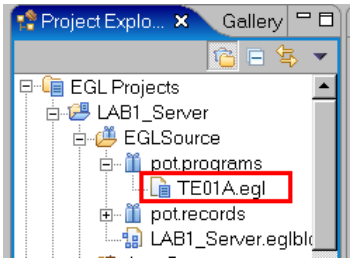


Рисунок 43

Добавим точку как показано на рисунке 44.

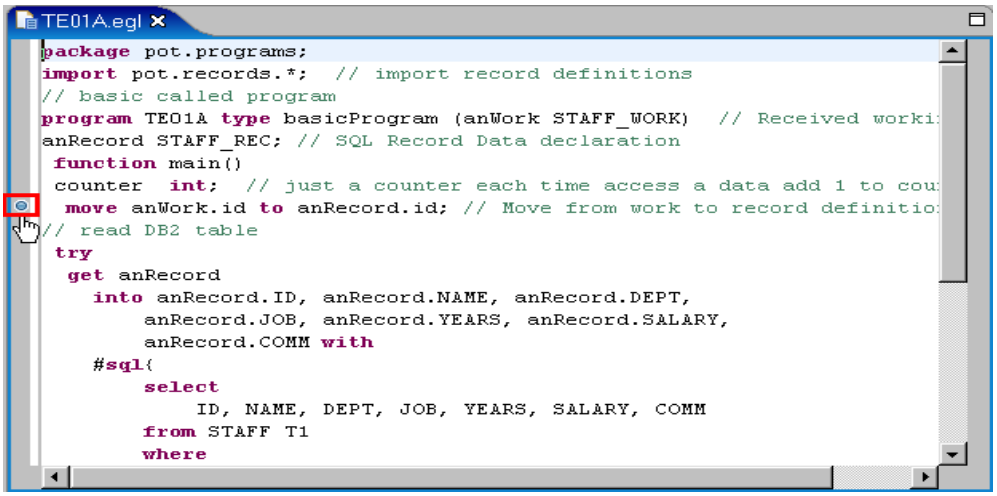


Рисунок 44

Закроем окно.

2.3.3 Тестирование в режиме отладки.

Правой кнопкой нажмем на файле **LAB2_Input.jsp** и выберем **Debug on Server...**

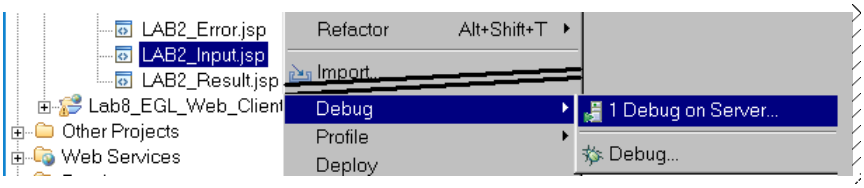


Рисунок 45

Если высветится окно, что выберем в нем **WebSphere Application Server v6.0** и отметим его по умолчанию, чтобы в следующий раз оно не возникало.

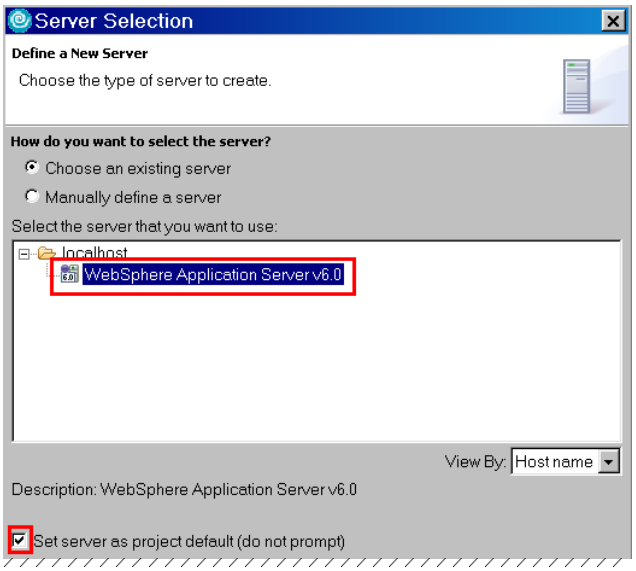


Рисунок 46

После загрузки появится окно Web Browser. В нем выведена страница ввода. Введем существующий ID и нажмем на кнопку.

Рисунок 47

Согласимся с изменением вида.

Выполнение программы остановилось на 1й точке останова.

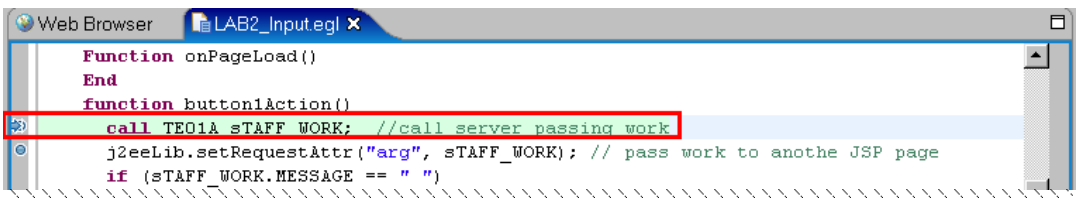


Рисунок 48

Используя окно Variables. Расположенное справа сверху, развернем **Lab2_Input** and **sTAFF_WORK**

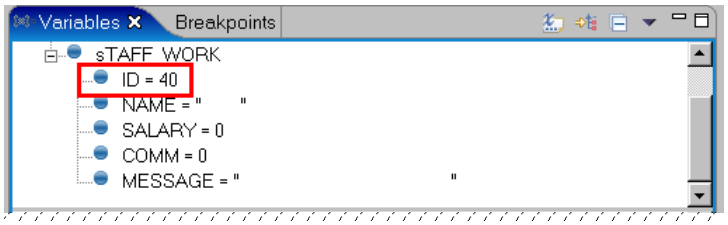


Рисунок 49

Делаем следующий «шаг» - нажимаем кнопку **Step Into (или F5).** Сейчас мы находимся перед вызовом серверной программы.

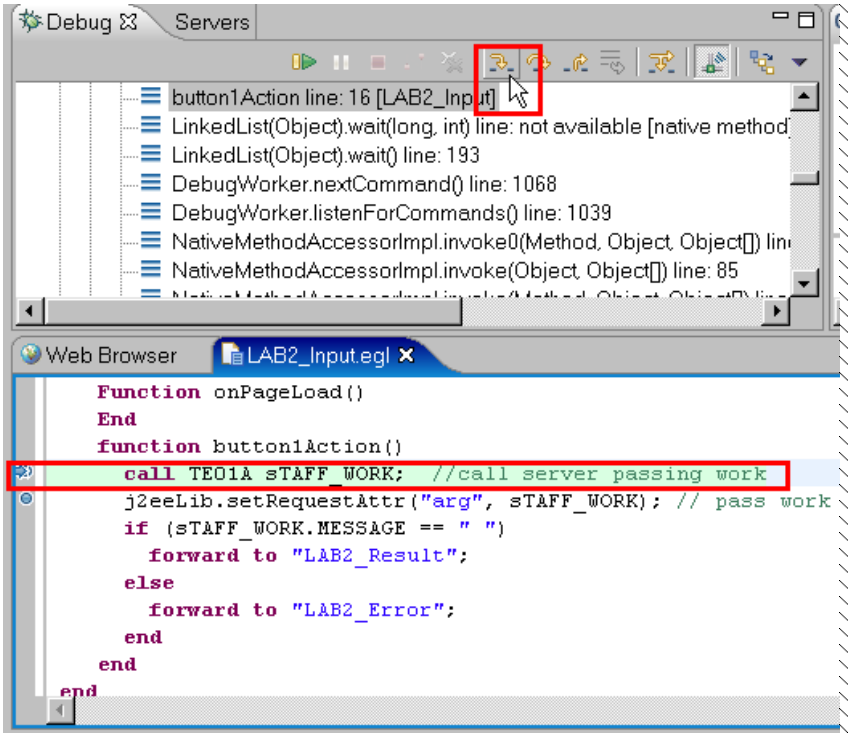


Рисунок 50

Делая следующий шаг, останавливаемся на первой инструкции серверной программы.

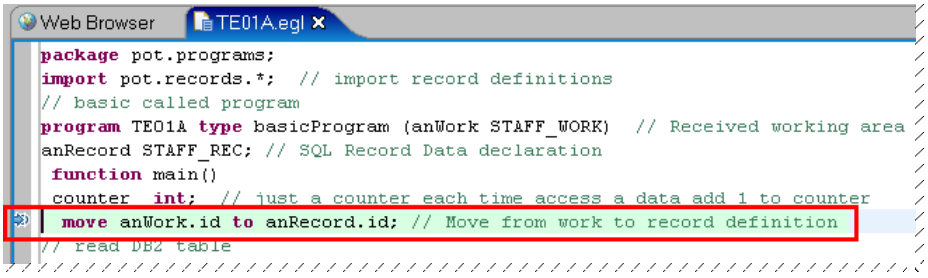


Рисунок 51

Теперь несколько раз делаем **Step Over**  или нажмем **F6,** просматривая текущие переменные.

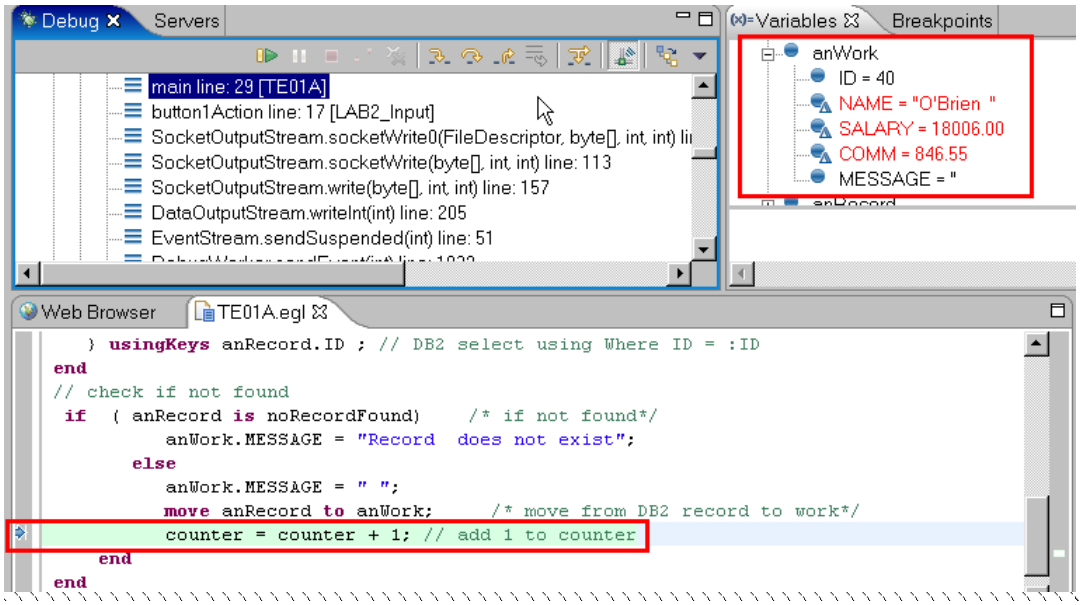


Рисунок 52

Счетчик иллюстрирует тот факт, что серверная программа используется при каждом нажатии на кнопку на странице. Нажимаем F8 – продолжая выполнение программы. В итоге мы увидим окно с результатами.

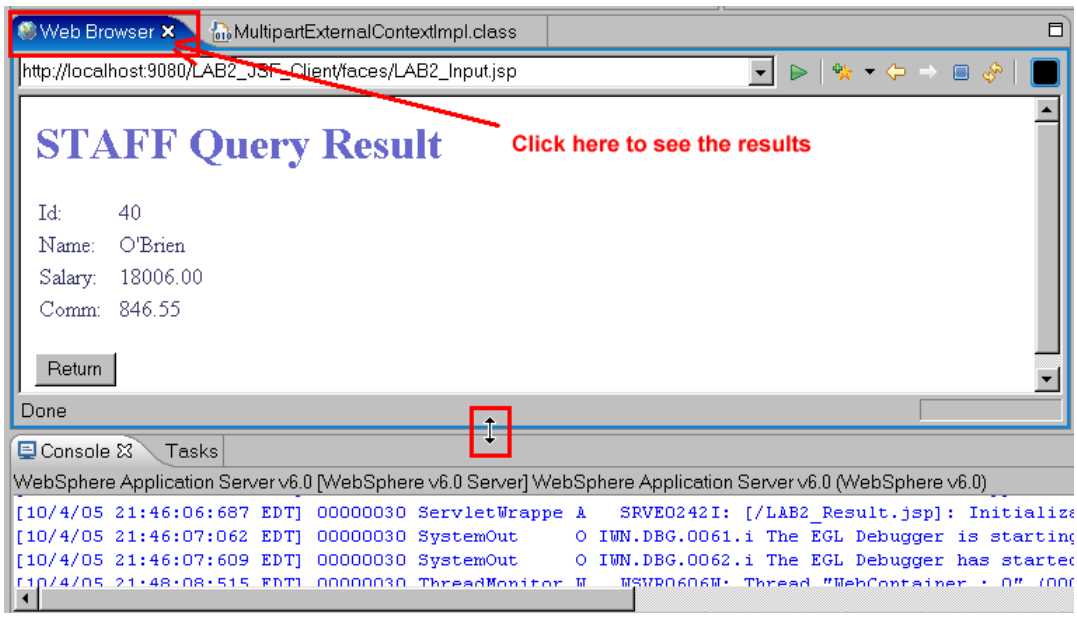


Рисунок 53

Ну и протестируем ошибочный запрос:

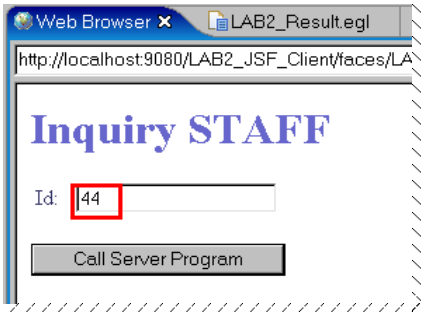


Рисунок 54

Нажимаем F8

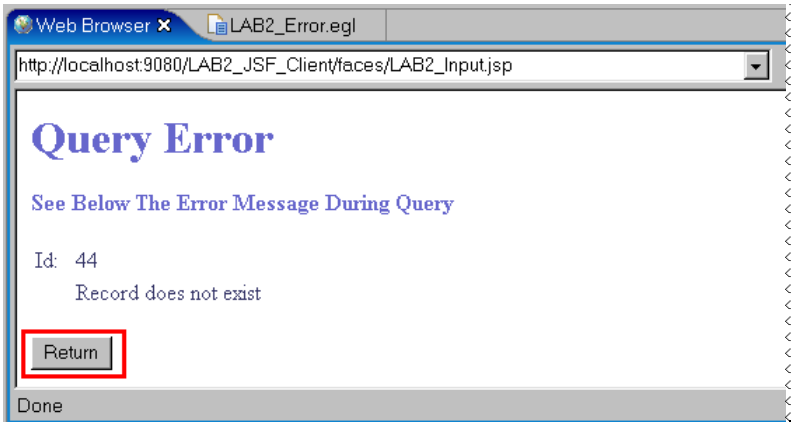


Рисунок 55

В конце тестирования – остановим сервер.

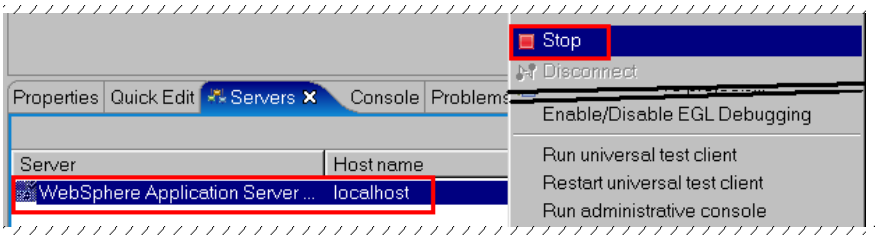


Рисунок 56

Поздравляем! Вы закончили лабораторную работу № 5.

Список литературы

1) Варфоломеев В.А. , Лецкий Э.К. , Шамров М.И. , Яковлев В.В. Архитектура и технологии IBM eServer zSeries Издательство: Интернет-университет информационных технологий - ИНТУИТ.ру Серия: Основы информационных технологий, Год выпуска: 2005, Объем: 640 стр.
2) Mike Ebbers, Wayne O'Brien. "An Introduction to the Mainframe: z/OS Basics", International Business Machines Corporation, 2004
3) John Ganci, Fabio Ferraz, Hari Kanangi и другие Rational Application Developer v6 Programming Guide, 06.2005, IBM International Technical Support Organization – 1426 стр.
4) Alex Louwe Kooijmans, Niek de Greef, Daniel Raisch, Eran Yona, The Value of IBM System z and z/OS in Service-Oriented Architecture, 07.2006, IBM International Technical Support Organization – 85 стр.
5) Regenaldo Barosa, Wilbert Kho, Tony Llopis и другие, материалы по EGL и EGL Web Services, 06.2006, IBM TechWorks, foils.
6) Галямова Е.В., Егоров М.А., Операционная система z/OS и основы программирования для больших ЭВМ (мейнфрейм) Методические указания по выполнению лабораторных работ, 2008. – 67 стр.
7) UNIX System Service User's Guide (SC28-1891), Appendix A – Shell Command Reference
8) z/OS Unix System Services, IBM zSeries Foils