Оглавление

1 Распределенная взаимоблокировка

Взаимоблокировка - постоянное блокирование некоторого множества процессов, которые либо конкурируют, пытаясь получить доступ к системным ресурсам, либо обмениваются информацией друг с другом.

Это определение справедливо как для централизованной, так и для распределенной системы.

Проблема взаимоблокировки, как и проблема взаимоисключения, в распределенной системе является более сложной, чем в системе с общей памятью.

Причинами этого усложнения является то, что ни один узел не обладает точными сведениями о текущем состоянии всей системы, а также то, что передача сообщения от одного процесса другому сопровождается непредсказуемой по длительности временной задержкой.

Рассматриваются два вида распределенных взаимоблокировок:

- **возникающие в связи с распределением ресурсов**
- **возникающие при обмене сообщениями.**

1.1 Взаимоблокировка, связанная с распределением ресурсов

Взаимоблокировка, связанная с ресурсами, возникает тогда, когда процесс пытается получить доступ к ресурсам, например, объектам данных из базы данных или ресурсам ввода-вывода, которые находятся на сервере.

При этом образуется группа процессов, каждый из которых запрашивает ресурс, которым обладает другой процесс из этой группы.

1.1.1 Условия возникновения

Взаимоблокировка в распределении ресурсов возникает только при соблюдении **всех** перечисленных условий:

- **Взаимоисключение.** Одновременно использовать ресурс может только один процесс.
- **Удержание и ожидание**. Процесс может удерживать выделенные ресурсы во время ожидания других ресурсов.
- **Отсутствие перераспределения**. Ресурс не может быть принудительно отобран у удерживающего его процесса.
- **Циклическое ожидание (Круговое распределение ресурса)**. Существует замкнутая цепь процессов, каждый из которых удерживает как минимум один ресурс, необходимый процессу, следующему в цепи после него.

Цель **алгоритма обработки взаимоблокировок** - либо предотвратить образование циклического ожидания, либо обнаружить, что оно возникло или может возникнуть.

В распределенной системе ресурсы распределены по разным узлам. Доступ к ресурсам регулируется управляющими процессами на этих узлах.

Любой отдельный управляющий процесс не обладает полными и свежими сведениями о **глобальном состоянии системы** и должен принимать решение, основываясь на локальной информации.

Таким образом, в **распределенной системе требуется новый алгоритм защиты от взаимоблокировок.**

1.1.2 Фиктивная взаимоблокировка

Одной из сложностей, возникающих при управлении распределенными взаимоблокировками, является явление фиктивной взаимоблокировки (**phantom deadlock**).

Пример такой взаимоблокировки**.**

Запись $\mathbf{P_1 \rightarrow P_2 \rightarrow P_3}$ означает, что процесс $\mathbf{P_1}$ приостановлен в ожидании ресурса, которым владеет процесс $\mathbf{P_2}$, а процесс $\mathbf{P_2}$ приостановлен в ожидании ресурса, которым владеет процесс $\mathbf{P_3}$.

Пусть вначале процесс $\mathbf{P_3}$ обладает ресурсом $\mathbf{R_3}$, а процесс $\mathbf{P_1}$ — ресурсом $\mathbf{R_1}$.

Предположим, что процесс $\mathbf{P_3}$ сначала генерирует сообщение, освобождающее ресурс $\mathbf{R_3}$, а затем — сообщение с запросом ресурса $\mathbf{R_1}$.

Если процесс выявления циклов сначала получит первое сообщение (освобождение ресурса), а затем второе (запрос ресурса), то взаимоблокировки нет.

В противном случае будет зарегистрирована взаимоблокировка.

На самом деле никакой взаимоблокировки не возникает; будет зарегистрирована ложная взаимоблокировка, вызванная отсутствием информации о глобальном состоянии системы.

1.1.3 Предотвращение взаимоблокировок

В распределенной среде можно использовать два метода предотвращения взаимоблокировок

1. **Возникновения циклического ожидания** можно избежать, линейно упорядочив типы ресурсов. Если процессу выделены ресурсы типа R, то он может последовательно запрашивать только те ресурсы, номер типа которых следует после R.

Основной недостаток этого метода состоит в том, что не всегда ресурсы можно запрашивать в том порядке, в котором они используются, поэтому ресурсы могут удерживаться процессом дольше, чем необходимо.

2. **Условий удержания и ожидания** можно избежать, потребовав, чтобы процесс запрашивал все необходимые ресурсы одновременно, и блокируя процесс до тех пор, пока такой запрос не сможет быть полностью выполнен в один момент времени.

Такой подход неэффективен по двум причинам. Во-первых, процесс может длительное время ожидать одновременной доступности всех затребованных ресурсов, в то время как реально он мог бы работать и только с частью из них. Во-вторых, затребованные процессом ресурсы могут значительное время оставаться неиспользованными, и в течение этого времени они оказываются недоступными другим процессам.

При использовании обоих методов нужно, чтобы процесс заранее мог указать свои требования к ресурсам. Это не всегда возможно; примером является приложение типа базы данных, в которую можно динамически добавлять новые элементы.

1.1.4 Избежание взаимоблокировок

Избежание взаимоблокировок — это метод динамического принятия решения о том, может ли предоставление данного ресурса по запросу привести к взаимоблокировке.

Избежание распределенных взаимоблокировок является непрактичным по таким причинам.

1. Каждый узел должен следить за глобальным состоянием системы, что требует значительных накладных расходов на хранение и обмен информацией.

2. Процесс проверки безопасного глобального состояния должен быть взаимоисключающим. В противном случае вопрос о предоставлении ресурса двум различным ресурсам может рассматриваться на двух узлах, и они оба одновременно могут принять решение о том, что удовлетворение запроса является безопасным. В результате возникнет взаимоблокировка.

3. В распределенной системе с большим количеством процессов и

ресурсов проверка безопасности состояния приводит к значительным накладным расходам на обработку запросов.

1.2 Взаимоблокировки, возникающие при обмене сообщениями

Взаимоблокировка, связанная с обменом сообщениями, возникает в том случае, когда требующимися процессу ресурсами являются сообщения; при этом **образуется группа процессов, каждый из которых ожидает сообщения от другого процесса из этой группы и ни один процесс не может отправить свое сообщение.**

Процесса $\mathbf{P_i}$ не может выполняться из-за ожидания сообщений от некоторого множества процессов **S**.

Процесс $\mathbf{P_i}$ сможет продолжить свою работу

- после получения **любого** из сообщений.
- либо только после получения **всех** ожидаемых сообщений.

1.2.1 Условие возникновения взаимоблокировки при передаче сообщений.

1. Все процессы множества S находятся в ожидании сообщений.

2. В каналах нет ни одного сообщения, передаваемого от одного члена множества S другому.

1.2.2 Блокировки, связанные с циклическим ожиданием сообщений

Все процессы множества S являются заблокированными, так как не может быть получено ни одно сообщение, благодаря которому хотя бы один процесс мог продолжить работу.

Процесс $\mathbf{P_1}$ ожидает сообщения либо от процесса $\mathbf{P_2}$, либо от процесса $\mathbf{P_5}$. Процесс $\mathbf{P_5}$ не ожидает никаких сообщений, поэтому он может отправить сообщение процессу $\mathbf{P_1}$, тем самым выводя его из состояния ожидания.

В результате звенья ($\mathbf{P_1}$, $\mathbf{P_5}$) и ($\mathbf{P_1}$, $\mathbf{P_2}$) удаляются.

Добавлена еще одна зависимость: процесс $\mathbf{P_5}$ ожидает сообщения от процесса $\mathbf{P_2}$, который ждет сообщения от процесса $\mathbf{P_3}$, ожидающего сообщения от процесса $\mathbf{P_1}$, который, в свою очередь, ждет сообщения от процесса $\mathbf{P_2}$.

В результате возникает взаимоблокировка.

1.2.3 Недоступность буферов сообщений

Одна из причин, по которой в системе с обменом сообщениями может возникнуть взаимоблокировка, связана с распределением буферов, предназначенных для хранения передаваемых сообщений.

Самым простым видом взаимоблокировки является **непосредственная блокировка хранения и передачи сообщений** (**store-and-forward deadlock**).

Она может возникнуть в том случае, когда узел пакетной коммутации использует **единый буферный пул**, буферы которого по мере необходимости предоставляются процессам передачи пакетов.

Ситуация, в которой все буферное пространство на узле **A** занято пакетами, предназначенными для узла **B**. То же справедливо и для узла **B**. Ни один из узлов не может принять ни одного пакета, поскольку их буферы заполнены. Таким образом, ни один из них не может передать или получить сообщение.

- Непосредственную блокировку хранения и передачи сообщений **можно предотвратить**, не позволяя всем буферам заполняться сообщениями, предназначенными для одного узла. Этого можно достичь, используя **раздельные буфера** фиксированного размера для каждой связи. Если сообщения, предназначенные для одного узла, не занимают все буферное пространство, взаимоблокировка не возникнет даже при использовании общего буферного пула.

Более сложная разновидность взаимоблокировки - **косвенная блокировка хранения и передачи сообщений.**

Она возникает, если образуется замкнутый цикл узлов, на каждом из которых очередь сообщений, передаваемых на соседний узел, до отказа

заполнена пакетами, предназначенными для узла, находящегося за соседним узлом.

Одним из простых способов предотвращения взаимоблокировки подобного типа является использование структурированного буферного пула.

Такие буферы имеют **иерархическую организацию**.

- Использование пула памяти на нулевом уровне не ограничивается никакими правилами; там может храниться любой входящий пакет.
- Буферы, которые находятся на уровнях с **первого** по ***N-й*** (где ***N*** - это максимальное количество промежуточных узлов при передаче пакета по сети), резервируются следующим образом:
- буферы на уровне ***k*** зарезервированы для пакетов, которые до этого подверглись не менее ***k*** транзитным пересылкам. Таким образом, при большой загрузке сети буфера заполнены от **нулевого** до ***N-го*** уровня.
- Если заполнены все уровни до ***k***-го включительно, то не принимается ни одно входящее сообщение, прошедшее менее *k* транзитных пересылок.

При соблюдении такой стратегии удается избежать как прямых, так и косвенных взаимоблокировок.

С проблемой взаимоблокировок можно столкнуться в коммуникационной архитектуре, обычно на сетевом уровне модели OSI.

Проблема такого же рода может возникнуть и в **распределенной операционной системе**, в которой межпроцессное взаимодействие происходит с помощью обмена сообщениями.

2 Репликация данных

Если система должна обеспечивать быстрый доступ к данным, эти данные часто *реплицируются,* то есть копируются на несколько узлов.

Таким образом снижается нагрузка на серверы и тем самым сокращается время ответа.

Помимо ускорения доступа репликация данных повышает **отказоустойчивость системы**: если один из серверов выйдет из строя, останутся другие.

Системам, в которых применяется репликация данных, свойственен ряд проблем, связанных с

- **доступностью серверов,**
- **коммуникационными задержками**
- **синхронизацией реплик.**

В определенных случаях **повышение доступности достигается за счет снижения степени согласованности данных.**

2.1 Согласование реплик данных

Два альтернативных варианта внесения изменений в реплицированные данные:

Слабая согласованность. Изменения, внесенные в одну из реплик, немедленно становятся видимы клиентам. Далее производится распространение этих изменений на другие реплики, в ходе которого их данные остаются несогласованными.

Строгая согласованность. Изменения становятся видимы клиентам только после их распространения на все реплики. Такую политику сложно реализовать, поскольку даже в случае доступности всех компьютеров, на которых находятся реплики данных, процесс их обновления **достаточно длителен**.

Еще одним типичным источником проблем является временный выход из строя отдельных сетевых соединений, при котором существует вероятность несогласованного обновления реплик в разделенных частях сети.

Система среднего масштаба может поддерживать так называемый «**горячий резерв**» - один или несколько резервных серверов с постоянно обновляемыми копиями данных, на которые в любой момент можно переключиться без потери состояния.

2.2 Сбор кворума для обеспечения строгой согласованности

Поскольку полный набор реплик доступен не всегда, можно производить обновление, когда в нем готово участвовать большинство реплик системы.

Необходимое их число называется ***кворумом записи***.

Также определяется ***кворум чтения***, наличие которого гарантирует, что как минимум одна из реплик содержит последнюю версию данных.

2.2.1 Сбор кворума в локальной системе

Если в системе имеется ***n*** реплик, кворум записи WQ и кворум чтения RQ определяются так:

WQ > *n* -2

RQ + WQ > *n*

Например, если ***n = 7***, для записи нужно собрать кворум из **пяти** реплик, а для чтения - из **трех**.

В процессе чтения проверяется время последнего обновления всех трех реплик и используется самая актуальная из них.

Системное программное обеспечение старается поддерживать в актуальном состоянии все имеющиеся реплики.

Каждая реплика поддерживается отдельным процессом, общее число которых также составляет ***n***.

Для фиксации обновления всех доступных реплик должен применяться протокол, обеспечивающий **атомарность этого действия**.

В процессе сбора кворума размер группы может меняться.

2.2.2 Согласование реплик в крупномасштабных системах

Сбор кворума в случае, когда требуется строгая согласованность реплик, требует взаимодействия с каждой из них.

В крупномасштабных системах кворум может быть столь велик, что его сбор займет много времени.

Для его ускорения реплики организуются иерархически

Выделяется набор первичных серверов, образующих верхний уровень иерархии, реплики которых должны быть строго согласованными.

Политика системы может определить, что все обновления должны производиться путем обращения к одному специально выделенному для такой цели первичному серверу, который отвечает за сбор кворума из первичных серверов.

Далее первичные серверы распространяют обновления на нижние уровни иерархии.

Процесс, которому необходимо прочитать самые последние данные, обращается к одному из **первичных серверов**.

Если же приоритет отдается скорости доступа, процесс будет контактировать с близко расположенным **сервером нижнего уровня**, но при этом остается вероятность получения устаревших данных.

3 Миграция процессов

Перенос (миграция) процесса состоит в передаче с одного узла сети на другой информации о состоянии процесса, достаточной для того, чтобы выполнение процесса можно было продолжить на новом узле.

3.1 Необходимость переноса

- **Распределение нагрузки**. Перенося процессы с более загруженных систем на менее загруженные, можно выравнивать загрузку систем, что будет способствовать повышению общей производительности. Эмпирические данные свидетельствуют о том, что таким образом можно достичь существенного роста производительности. Однако алгоритмы распределения нагрузки должны быть тщательно разработаны. Чем интенсивнее происходит обмен информацией в распределенной системе, тем хуже становится ее производительность.
- **Производительность обмена информацией**. Процессы, интенсивно обменивающиеся информацией, можно перенести на один узел, чтобы снизить стоимость передачи данных и длительность взаимодействия.

Кроме того, если процесс выполняет анализ данных, которые находятся в файле или наборе файлов, размер которого превышает размер процесса, возможно. лучше перенести процесс к данным, а не наоборот.

- **Работоспособность**. Может возникнуть необходимость перенести выполняющийся в течение длительного времени процесс, чтобы избежать запланированного простоя или отказов, о возможности возникновения которых стало известно заранее. Если от операционной системы поступило подобное извещение, процесс, которому нужно продолжать свою работу, может либо перейти на другую систему, либо убедиться, что позже его можно будет перезапустить на этой же системе.
- **Использование особых возможностей**. Процесс может переместиться с целью использования уникальных аппаратных или программных возможностей конкретного узла.

3.2 Механизмы переноса процессов

Разрабатывая средство переноса процессов, следует обращать внимание на ряд вопросов, в число которых входят следующие:

- **Кто должен стать инициатором переноса?**
- **Какая "часть" процесса должна быть перенесена?**
- **Что должно произойти с ресурсами переносимого процесса?**

3.2.1 Инициация переноса

Выбор инициатора переноса зависит от предназначения переноса процессов.

Если целью является **перераспределение нагрузки**, то решение о необходимости переноса процесса должно приниматься **модулем** операционной системы, **управляющим загрузкой** систем. Этот модуль отвечает за вытеснение переносимых процессов или передачу им сигналов. Чтобы определить, куда нужно перенести процесс, модуль должен обмениваться информацией с подобными ему модулями из других систем и иметь возможность отслеживать загрузку этих систем.

Если же целью является использование каких-то **особых ресурсов**, то при возникновении необходимости процесс может мигрировать **самостоятельно**. В этом случае процесс должен быть осведомлен о существовании распределенной системы; в предыдущем случае работа функции переноса процесса и существование нескольких систем могут остаться для процесса незаметными.

3.2.2 Что переносится

Необходимо перенести образ процесса, состоящий, по крайней мере, из управляющего блока. Кроме того, нужно обновить все связи, предназначенные для передачи сигналов и сообщений, между данным процессом и другими процессами.

Ответственность за перенос управляющего блока процесса и обновление отображения связей возлагается на **операционную систему**. Перенос процесса остается незамеченным как самим процессом, так и его партнерами по обмену информацией.

Перенос управляющего блока процесса осуществляется просто. Сложность с точки зрения производительности представляет **перенос адресного пространства процесса и открытых им файлов**.

3.2.3 Перенос адресного пространства процесса

- **Интенсивная (полная)**. Во время переноса процесса перемещается все адресное пространство. Этот подход, несомненно, является самым понятным. В старой системе не остается никаких следов процесса. Однако если адресное пространство очень большое и процессу вряд ли понадобится большая его часть, то можно избежать его полного перемещения, тем самым на порядок сократив расходы по переносу процесса.
- **Предварительное копирование**. Пока процесс продолжает выполняться на старом узле, его адресное пространство копируется на новый узел. Страницы, которые были изменены на старой системе во время

предварительного копирования, нужно скопировать еще раз. Такая стратегия сокращает время простоя процесса, возникающего при его переносе.

- **Интенсивная (выборочная)**. Осуществляется перенос только тех страниц адресного пространства, которые находятся в основной памяти и были изменены. Все остальные блоки виртуального адресного пространства переносятся только по мере необходимости. Это сводит к минимуму передачу данных. Однако при соблюдении такой стратегии нужно, чтобы машина, с которой переносится процесс, продолжала принимать участие в его работе, поддерживая элементы таблицы страниц и/или сегментов, а также осуществляя удаленное предоставление страниц.
- **Копирование по запросу**. Эта стратегия является разновидностью предыдущей. В данном случае страницы предоставляются только тогда, когда на них поступает запрос. При этом начальные затраты на перенос процесса являются минимальными (в пределах от десятых до сотых долей микросекунды).
- **Очистка**. Страницы процесса удаляются из основной памяти старой машины и переносятся на диск. После этого доступ к ним осуществляется непосредственно с диска старого узла, без участия его основной памяти. Такая стратегия избавляет от необходимости держать в основной памяти старого узла какие-либо страницы переносимого процесса, освобождая занимаемый им блок памяти для других процессов.

Во многих случаях заранее неизвестно, понадобится ли большая часть отсутствующего в основной памяти адресного пространства. Однако если процессы разбиты на потоки и если основной единицей переноса является не процесс, а поток, то наиболее привлекательно выглядит стратегия, основанная на удаленной подкачке страниц.

Этой стратегии почти всегда отдается предпочтение, потому что оставшиеся потоки процесса остаются на старой системе и им также требуется доступ к адресному пространству этого процесса.

3.3 Перенос открытых файлов.

Если первоначально файл находится в той же системе, что и переносимый процесс, и если файл заблокирован переносимым процессом для своего исключительного использования, то имеет смысл перенести файл вместе с процессом.

При этом возникает опасность, состоящая в том, что перенос процесса может быть лишь временным, и в течение пребывания процесса на новой системе файл ему может не понадобиться.

Поэтому, возможно, стоит переносить весь файл только после того, как переносимый процесс обратится к нему.

Если файл используется совместно с другими распределенными процессами, то следует поддерживать распределенный доступ к этому файлу без перемещения.

Если разрешено использовать кэш, то возникают дополнительные сложности.

Например, если процесс открыл файл для записи, затем разветвился, а затем один из его дочерних процессов оказался перенесенным на другую машину, то этот файл теперь должен быть открыт для записи с двух различных узлов.

3.4 Вытесняющие и невытесняющие переносы

Вытесняющий перенос процессов включает передачу частично выполненного процесса (или, по крайней мере, процесса, создание которого уже завершено).

Невытесняющий перенос процессов — более простая функция, работающая только с процессами, выполнение которых еще не началось, благодаря чему переносить информацию о состоянии процесса не требуется.

В обоих случаях необходимо передать на удаленный узел информацию о среде, в которой будет выполняться процесс. Сюда могут входить сведения о

текущем каталоге пользователя, унаследованных процессом привилегиях и унаследованных ресурсах, таких, как дескрипторы файлов.

Невытесняющий перенос процесса может быть полезен при балансировке нагрузки. Эта схема обладает тем преимуществом, что позволяет избежать накладных затрат на полный перенос процесса. Недостатком этой схемы является то, что она не может адекватно реагировать на внезапные изменения в распределении нагрузки.