**26.11 лекция ОСы**

7:38

**Взаимоисключение в распределенных системах**

К централизованному алгоритму можно добавить, что в интернетах этот алгоритмвстречается как Алгоритм Забияки. Под забиякой понимается процесс-координатор. Если какой-то процесс из цепи взаимодействующих процессов обнаружил отсутвие координатора, он инициирует новые выборы координатора.

Второй тип алгоритма этораспределенный алгоритм.

Алгоритм формулируется следующимобразом: когда процесс хочет войти в критическую секцию,  он формирует сообщение. В этом сообщении он указывает:

    \* идентификатор нужной ему критической секции;

    \* свой номер (идентификатор);

    \* время формирования сообщения по своим локальным часам.

    Предполагается, что передача сообщения надёжна: получение каждого сообщения сопровождает подтверждение.

**Протокол**- соглашение о том, какие действия выполняют процессы при передаче и получении сообщения. Надёжный протокол всегда предполагает подтверждение получения сообщения.

    Это сообщение, сформированноепроцессом, процесс рассылает всем взаимодействующим процессам (то есть н-1 сообщение )

    Получив такое сообщение, процесс проверяет, в какой ситуации он сам находится по отношению к критическойсекции.

    Возможны 3 ситуации:

        1) получивший сообщение процесс не собирается входить в данную критическуюсекцию (и не находится в ней). В этом случае процесс посылает сообщение-ответ с разрешением войти в критический участок;

        2) процесс, получивший сообщение, находится в данной критической секции. В этом случае никакое сообщение не посылается;

        3) Процесс, получивший сообщение, сам формировал сообщение запрос на вход в данную критическую секцию, но еще не успел это сделать.

        (на доске: хочет войти в ту же крит секцию). Тогда процесс проверяет временную метку создания своего сообщения, сравнивает ее с временной меткой полученного сообщения. Если его сообщение было сформировано раньше, то никакого сообщения -ответа процесс не посылает. Если в результате таких рассылок процесс получает н-1 сообщение подтверждения входа в критическийучасток, то процесс может войти в крит уч. Если он не получает хотя бы одно сообщ то в крит уч процесс войти не может.

        В этом случае процесс должен послать н-1 сообщ-запрос и в ответ получить н-1 сообщение-разрешение на вход, иначе войти не может.

        20:36 <3

**Алгоритм Токен Ринг** (**Token Ring**)

        Смысл заключается в том, что процессы создают логическое кольцо - замкнутую цепочку.

        Каждый процесс в логическом кольце знает идентификатор следующегопроцесса в этом логическомкольце. по данному лог кольцу передается так называемый токен. Это специальное сообщение, которое содержит идентификатор конкретной критической секции. Этот токен переходит от n к n+1, циркулирует по лог кольцу. когда процесс получает токен, он анализирует, нужно ли ему войти в данную критическуюсекцию. Если нужно, получив токен, он входит в крит секц и токен удерживает. Выйдя из крит секц, процесс отправ токен следующему в цепи процессу. Пока процесс находится в крит секции он удерживает токен. Если ни один процессне заинтересован во вхождении в критическую секцию токена, такой токен быстро циркулирует по цепочке.

Самый надежный алгоритм: централизованный алгоритм. За счет возможности выбора нового координатора.

В случае распр алга:

Если какой-то из процессов перестал существовать, то  процесс, разославший n-1 сообщение-запрос, не сможет получитьn-1 сообщ-ответ: потеря работоспособности по конкретной крит секции.

В случае Токен Ринг: если какой-то проц перестает существовать, то цепь разрывается. Чтобы ее восстановить, надо предпринять соотв действ.

Если циркулирование токена по каким то причинам прервано, то есть только одно средство определения: таймаут.

27:26

Высокоуровневое средство взаимодействия процессов: неделимые транзакции.

Добавление слова неделимые можно считать излишним: любая транзакция должна быть неделимой, это свойство транзакции.

**Транзакцией**называется последоватльность операций над одним или несколькими объектами базы данных, такими как файлы, записи и тому подобное, которая переводит систему из одного целостного состояния в другое целостное состояние.

    Модель неделимой транзакции пришла из бизнеса. Переговорный процесс двух фирм о купле-продаже: в процессе переговоров условия договора могут меняться, при этом каждая из договаривающихся сторон может отказаться продолжать переговоры. Но после подписания контракта, сделка считается установленной (определённой) - ее надо выполнять.

    Такое свойство транзакции называется "все или ничего".

    Для того, чтобы транзакция могла выполняться, система (или ПО) должна предоставлять необходимый набор примитивов - команд управления транзакцией:

    \* begin\_transaction

    \* end\_transaction

    \* abort\_transaction (прерывание означаетнеобходимость восстановления исходных занчений)

    \* read, write transaction (позволяют читать или писать данные)

    begin и end  определяют границы транзакции.

**Свойства транзакций:**

        \* упорядоченность: гарантирует, что если две или более транзакции выполняются в одно и то же время, то конечный результат выглядит так, как если бы транзакции выполнялись в определенном порядке.

        \* неделимость: если транзакция находится в процессе выполнения, то промежуточные результаты выполнения транзакции не видны никакому другому процессу.

        \* постоянство: означает, что после фиксации транзакции никакой сбой не может отменить результатов ее выполнения.

**Существует два основных подхода к реализации механизма транзакций:**

    1) Для каждого процесса, участвующего в транзакции, создается **индивидуальноерабочее пространство**.

    (на доске: индивидуальное рабочее пространтсво). В этом пространстве находятся копии всех файлов (объектов), которые нужны процессу для выполняния транзакции. Пока транзакция не будет зафиксирована или не прервется, все изменения выполняются только над объектами в этом индивидуальномрабочем пространтсве.  Если транзакцияуспешно завершается, то все изменения копируются в исходные объекты. Если транзакция прерывается, то копии просто удаляются (исходные файлы не были изменены). Очевидный недостаток: большие накладные расходы, тк создаются доп копии файлов (объектов) в системе.

    2) **Список намерений** . В данном подходе модифицируются сами файлы/записи/объекты, но перед любым изменением выполняется запись в специальный файл, который называетсяжурнал регистрации. В этом журнале отмечается, какая транзакция делает изменения, какой файл/запись изменяется и в этот журнал записывается старое и новое значения изменяемого файла записи. Только после того, как транзакцияуспешно выполнена, указаные изменениясохраняются(?) в исходном файле. Если транзакция фиксируется (вып успешно), то об этом делается запись в журнале регистрации, но записи не удааляются. Если транз прерывается, то журнал регистрации используется для приведения файлов / записей в исходное состояние - такое действие называется откатом.

    В распределённых системах фиксациятранзакции может потребовать взаимодейстия нескольких процессов, которые выполняются на разных машинах. В этом случае каждая такая отдельная машина хранит какие-то переменные/файлы базы данных. Для достижения неделимости транзакции в распределенных системах, используется специальный протокол - протокол двухфазной фиксации транзакций.

    Это не единственный протокол для фиксации транзакции, но он считается наиболее надежным. Суть протокола: для обеспечения выполнения транзакции в распределенной системе один из взаимодействующих процессов выполняет функции координатора.

    картинка - протокол двухфазной фиксации: 49:08

    надписи: запись "приготовиться", послать сообщения "приготовиться", собрать собщения-ответы, запись "готов", послать сообщение "готов"

    Первая фаза  протокола связана с тем,что  координатор должен записать в журнал регистрации приглашение для других процессов участвующих в транзакции: например, "приготовиться".

    После того, как эта запись сделана в журнале регистрации, выполняется рассылка сообщений  процессам, участвующим в транзакции.

    Подчинённый процесс, получив сообщение "приготовится", в журнал регистрации выполняет запись о готовности завершить транзакцию, например, "готов".

     После успешной записи в журнал регистрации, подчинённый процесс может послать сообщение координатору.

     Первая фаза заключается  в том, что координатор собирает  сообщения от других процессов - ответы. Затем втораяфаза.

     Вторая фаза возможна, только в том случае, если координатор получил сообщения от всех процессов о готовности завершить транзакцию.

     Если действие завершается с некоторой ошибкой, то в этом случае никаких действий подчиненный процесс совершать не будет и никакого сообщения "готов" не пошлет.

     Если координатор собирает сообщения от всех подчинённых процессов, то координатор переходит к фиксации (завершению) транзакции.

     Координатор делает запись в журнале регистрации "завершить". Только после успешной записи в журнале регистрации, корординатор посылает сообщение "завершить" подчиненным процессам.

     Получив соощение "завершить", подчиненный процесс записывает в журнал "завершить" (фиксировать) .

     Только после этого, посылается ответное сообщение. Процесс координатор снова должен собрать эти сообщения ответы. если он собрал сообщения от всех участников транзации процессов, то результат транзакции фиксируется. Вторая фаза является фазой фиксации результатов транзакции.

     В явном виде в распределенном организме присутствуют временные отметки. В системе важно, чтобы соблюдалось отношение "случилось до/случилось после".

     59:30

     Алгоритм Лампорта применяется для синхронизации временных ответок (для соблюдения отношения "случилось до/случилось после").

     Алгоритм Лампорта - не единственное решение, которое позволет согласовывать действия процессов, обеспечивать отношение "случилось до/случилось после"

     Другое решение - векторные часы

     61:20

**Векторные часы**

          Алгоритм векторные часы используется для обеспечения частично упорядоченных событий. Они широко используются в распределенных системах для обеспечения отношения причинности (случилось до/случислось после). В частности, векторные часы используется для обнаружения нарушения причинности.

     В отличие от временных меток Лампорта, векторные часы соодержат вектор отметок.

     Каждый раз, когда какой-то процесс выполяет какие-то действия (когда в процессе происходит какое-то событие) в векторе отметок фиксируется отметка.

     иллюстрация вектора отметок     64:55

     Взаимодействуют три процесса: A, B, C. В начальный момент времени вектор обнулен. Если в каком-то процессе происходит некоторое событие, например,отправка и прием сообщений (собсна другие события, которые происходят на локальных машинах, нас не интересуют). Скажем, процесс в какой-то момент времени сформировал сообщение и сделал в своем векторе отметку и проставил значение 1.

     Взаимодейтсвие в распределенных системах связано с временными задержками.

      Получение процессом сообщения так же является событием.

      Вектор, фиксированный процессом B - B:1. После этого процесс B формирует соообщение процессу A c отметкой B:2. Процесс А формирует ответное сообщение процессу B c отметкой А:2.

      (см ее комментарии к рисунку на записи, вроде в личный кабинет выложит)

      Важно различать причину и следствие.

      Пунктиром выделены действия, которые являются причиной. За пунктиром - следствия.

      Каждый раз, когда процесс получает сообщение, он увеличивает собственную логическую переменную - собственный логические часы в векторе - на 1. Каждое событие инкрементирует собственное значение счетчика событий процесса.

      Данные вектора могут использоваться для обеспечения частичного порядка.  Существуют соответствующие доказательства, что данный алгоритм - векторные часы - позволяет обеспечить свойство частичного порядка.

Алгоритм Лампорта можно считать отправной точкой развития этой темы.(взаимодействие процессов в распределенной системе)

77:50

Вызов удаленных процедур (RPC)

**☦**

Наличие такого высокоуровневого средства взаимодейтсвия как транзакции, это самый нижний уровень взаимодействия процессов в распределенных системах.

      RPC - remote procedure call

      RPC является важнейшим механизмом для приложений типа клиент-сервер. Он был разработан Sun Microsystems. Он представляет собой набор библиотечных функций, например:

          \* RPC->NIS(Network Information System)

          \* RPC->NFS (Network File System)

Его основы:

    RPC это механизм, с помощью которого один процесс активизирует другой процессна этой же или удаленной машине для выполнения какой-то работы (функции) от своего имени. Этот механизм был реализован в Unix таким образом, что он напоминает вызов обычной (локальной) функции. Вызов функции связан с передачей этой функции каких-то данных, которые называется фактическим параметрами. Вызванная функция обрабатывает эти данные и в итоге, формирует результат, который возвращается основной функции, которая вызвала данную локальную функцию.

    RPC внешне напоминает именно такие действия. Но вызываемая функциявыполняется (может выполняться) на совершенно другой машине. Это взаимодействие выполняется по модели клиент-сервер.

    Процесс, который вызывает функцию RPC является клиентским. Процесс, который выполняет функцию на этой же или на другой машине, является серверным.

    Наиболее ясно механизм RPC можно продемонстрировать на схеме: 86:38.

(клиентский процесс  RPC серверный процесс)

**Как устроено:**

[прикладная программа]

---------->

интерфейс ( определяется с помощью языка IDL )

------------->

(клиентский stab)

---------->

[библиотека RTL(?) RPC]

----->

(сокет BSD)

------------------------------------------->

/\*серверный процесс\*/

(сокет BSD)

---------->

[библиотека (?)RTL RPC]

----------->

(серверный stab)

------------>

интерфейс ( определяется с помощью языка IDL )

------------>

[прикладная программа сервера]

--\*Формируется ответ и посылается ответ обратно по той же схеме ( снизу - вверх ) \*--

Прикладная программа заинтересована в выполнении каких-то действий на удаленной машине. Прикладная программа, выполняемая клиентским процессом, выполняет вызов удаленной процедуры. При этом для этой программы это просто вызов на локальной машине. В системе существует интерфейс, то есть вызов, выполняемый приложением, преобразуется в вызов так называемого клиентского stab'a(фиктивный модуль клиента)(клиентская заглушка).

В результате вызова начинает выполняться этот stab. Происходит вызов функции стандартной библиотеки. Взаимодействие выполняется через сокеты.

Механизм сокетов был предложен и реализован в ОС Unix BSD

В результате вызова библиотечной функции, происходит обращение к местному сокету (сокету BSD).

Все эти действия выполняются в рамках вызова клиентской программы на локальной машине, на которой выполняется киентский процесс. В результате задействования сокетов, задействуются сетевые стредства (т. н. транспортный уровень). Осуществляется передача сообщения .

Все действия кторые выполняются на стороне клиента - их итогом является формирование сетевого пакета, который передается процессу-серверу.

Очевидно, что в распределённой системе такой пакет должен содержать адрес назначения.

На серверной стороне через сокет пакет поступает на машину, на которой выполняется процесс-сервер. Вызывается библиотека и её функции. На стороне сервера имеется серверный stab.

Интерфейс в таких системах определяется с помощью языка IDL. Через интерфейс уже прикладной программе сервера передается соответсвующий запрос на обслуживание. Программа-сервер выполяет данный запрос и формирует ответ, который через серверный stab , через сокет, через сетевые средства взаимодействия возвращается запросившему процессу. Очевидно, в обратном порядке.

С  RPC связано очень много понятий. Это низкоуровневый механизм взаимодействия в распределенных системах. RPC достигает прозрачности следующим образом: если вызываемая процедура/функци действительно является удаленной, в библиотеку помещается вместо локальной функции, другая версия этой функции, которая называется клиентским stabом (вызов подменяется другим вызовом).

Клиентский стаб вызывает последовательность действий, вызывает в процессе выполн. В процессе выполненияклиентского стаба выполняется ситемный вызов. Система переходит в режим ядра. В режиме ядра выполняются совершенноспецифические для RPC действия, связанные с формированием сообщения. После того, как сообщение сформировано, это сообщение по сети передается другой машине. Воспринимается через сокет в режиме ядра, распаковывается в режиме ядра, передается серверному стабу, затем передается прикладной программе.

Иллюстрация (86:38) была очень общая.

101:32

Процесс клиента

--------------------------------

Вызов клиентского стаба

------------------------------------

V

Действия, выполняемые клиентским стабом:

\* Подготовка буфера сообщения

\* Упорядочивание параметров в буфере

\*  Добавление в собщение полей заголовков

\*  Системный вызов, в результате, которого происходит переход в режим ядра ивыполняется переключение контекста

-------------------------------------

\* В ядре для того, чтобы обработатьсообщение, оно должно быть скопированов буфер ядра.  (В режиме пользователя свой буфер, в режиме ядра - свой)

\* Определение адреса назначения

\* Адрес помещается в заголовок сообщения

\* Инициализируется сетевой интерфейс

\* Включается таймер

-----------------------------------------------

Сообщение исходное, которое для процесса-клиента не является сообщением.

Он вызывает функцию, затем уже начинает работать система и соответствующая библиотека и формирует пакет.

В чем суть пакета : в нем указываетсясодержание (письмо) в конверте, а на конверте адрес, куда надо доставить.

Здесь прямо аналогичные действия.

(\*записывает снизу вверх)

Пакет поступает на машину, на которой выполняется процесс-сервер.

\* Получение такого пакета приводит к прерыванию - необходимо обработать получение пакета. Процесс, который выполнялся прерывается.

\* Проверяется правильность пакета.

\* Подтверждается правильность пакета.

\* Определяется стаб, которому предназначен данный пакет.

\* Выполняется переключение контекста в режим пользователя.

---------------<стаб сервера>---------------

\*  в пользовательском режиме выполняется распаковка параметров

\* помещение параметров в стек

\*  Вызов программы сервера

Этот стаб берет на себя задачу на дальнейшее продвижение запроса и его преобразования.

И на стороне сервера выполняется необходимая работа. (Выполнение требуемой работы)

Все рассмотренное - статическая адресация. Как показано в последовательности определяется адрес сервера и по конкретному адресу высылается соответствующий пакет, который распаковывается. Но мы понимаем, что система это живой организм, в котором происходят разные события в частности запрашиваемый процесс может отсутствовать. Поэтому для того, чтобы избежать подобных проблем связанных с отсутствиемадресата в распределенных системах используется динамическое связывание.

117:37

Динамическое связывание (Binding)

Если говорить языком, свойственным данному типу взаимодействия, то формирование данных клиентом состоит из двух процессов:

    1) маршалинга

    2) сериализации

    Маршалинг - это перекодировка и упаковка данных по 2 сообщения

    Сериализаций - преобразование сообщения в последовательность байтов.

    На стороне сервера выполняются обратные действия:

        1) Десериализация

        2) Демаршалинг

        Связывание может быть статическим и динамическим. Динамическое называется binding.

        Для того, чтобы динамическое связывание было возможным, необходима специализированная служба, которая определяет фактически адрес сервера (ответствена за локализацию сервера). Для этого должен быть создан дополнительный программный слой, который называется сервером имен и каталогов.

    Binder - сервер имен и каталогов.

127:00 - ложный конец

        фсё?

       Good job, team

        Огромное спасибо фанатам тайпинга

        <3

Вы думали, я вас не переиграю?

        вд || процессов.

129:00

IPC - Inter Process Communication

Средства взаимодействия параллельных процессов. (Для новой лабы по unix)

Мы рассматриваем средства IPC System V.

К ним принято относить:

    \* сигналы

    \* семафоры

    \* программные каналы  (именованные  и неименованные)

    \* очереди сообщений

    \* сегменты разделяемой памяти

    \* ввод-вывод с отображением в память ( в современных ОС)