БИЛЕТ № 14

1. Модель непротиворечивости - свободная непротиворечивости.

Репликация данных

Важным вопросом для распределенных систем является репликация данных. Данные обычно реплицируются для повышения надежности и увеличения производительности. Одна из основных проблем при этом — сохранение непротиворечивости реплик. Говоря менее формально, это означает, что если в одну из копий вносятся изменения, то нам необходимо обеспечить, чтобы эти изменения были внесены и в другие копии, иначе реплики больше не будут одинаковыми.

Вопросы, связанные с репликацией

1. как должны расходиться обновления по копиям
2. поддержка непротиворечивости
3. проблема кеширования

Основные проблемы репликации

Первая проблема - обеспечение взаимного исключения при доступе к объекту. Существует два решения:

- объект защищает себя сам (к примеру, доступ к объекту совершается при помощи синхронизированных методов в Java - каждому клиенту будет создано по потоку и виртуальная Java-машина не даст воспользоваться методом доступа к ресурсу обоим потокам в один момент времени)
- система защищает объект (ОС сервера создает некоторый адаптер, который делает доступным объект, только одному клиенту в один момент времени)

Вторая проблема - поддержка непротиворечивости реплик. Здесь снова выделяют два пути решения:

- решение основанное на осведомленности объекта о том, что он был реплицирован. По сути, обязанность поддерживать непротиворечивость реплик перекладывается на сам объект. То есть в системе нет централизованного механизма поддержки непротиворечивости репликаций. Преимущество в том, что объект может реализовывать некоторые специфичные для него методы поддержки непротиворечивости.
- обязанность поддержки непротиворечивости накладывается на систему управления распределенной системой. Это упрощает создание реплицируемых объектов, но если для поддержки непротиворечивости нужны некоторые специфичные для объекта методы, это создает трудности.

Модели непротиворечивости, ориентированные на данные

По традиции непротиворечивость всегда обсуждается в контексте операций чтения и записи над совместно используемыми данными, доступными в распределенной памяти (разделяемой) или в файловой системе (распределенной).

Модель непротиворечивости (consistency model), по существу, представляет собой контракт между процессами и хранилищем данных. Он гласит, что если процессы согласны соблюдать некоторые правила, хранилище соглашается работать правильно. То есть, если процесс соблюдает некоторые правила, он может быть уверен, что данные которые он читает являются актуальными. Чем сложнее правила - тем сложнее их соблюдать процессу, но тем с большей вероятностью прочитанные данные действительно являются актуальными.

- свободная

Слабая непротиворечивость имеет проблему следующего рода: когда осуществляется доступ к переменной синхронизации, хранилище данных не знает, то ли это происходит потому, что процесс закончил запись совместно используемых данных, то ли наоборот начал чтение данных. Соответственно, оно может предпринять действия, необходимые в обоих случаях, например, убедиться, что завершены (то есть распространены на все копии) все локально инициированные операции записи и что учтены все операции записи с других копий. Если хранилище должно распознавать разницу между входом в критическую область и выходом из нее, может потребоваться более эффективная реализация. Для предоставления этой информации необходимо два типа переменных или два типа операций синхронизации, а не один.

Свободная непротиворечивость (release consistency) предоставляет эти два типа. Операция захвата (acquire) используется для сообщения хранилищу данных о входе в критическую область, а операция освобождения (release) говорит о том, что критическая область была покинута. Эти операции могут быть реализованы одним из двух способов: во-первых, обычными операциями над специальными переменными; во-вторых, специальными операциями. В любом случае программист отвечает за вставку в программу соответствующего дополнительного кода, реализующего, например, вызов библиотечных процедур acquire и release или процедур enter_critical_region и leave_critical_region.

Основная идея: Это модификация слабой непротиворечивости, но теперь есть не одна процедура “синхронизировать”, а две процедуры “вход в критическую секцию” (надо убедится что у меня актуальные данные), “выход из критической секции” (надо разослать всем то, что я понаделал и узнать что понаделали другие). Так же вводятся барьеры.

2) Методы предотвращения тупиков. Добавить с нашего конспекта

Тупик – это ситуация, которая происходит, когда в группе процессов каждый получает монопольный доступ к некоторому ресурсу и каждому требуется еще один ресурс, принадлежащий другому процессу в группе.

Самый простой метод борьбы с тупиками – метод страуса, следует воткнуть голову в песок и притвориться, что никаких проблем вообще не существует.

Тупик неизбежен, если выполняются 4 следующих условия:

1. **Условие взаимоисключения.** Процесс монопольно владеет ресурсом до своего завершения, т.е. ресурс неразделяем.
2. **Условие удержания и ожидания.** Процесс, владеющий ресурсом, может запрашивать новые ресурсы.
3. **Условие неперераспределяемости.** У процесса нельзя принудительным образом отнять ранее полученный ресурс и передать его другому процессу.
4. **Условие кругового ожидания.** Процесс ждет доступа к ресурсу, удерживающему следующим членом последовательности.

Способы борьбы с тупиками:

- предотвращение (необходимые методы, чтобы тупик был невозможен, обычно убирают одно из условий возникновения тупика);
- обход;

- обнаружение;

восстановление системы при обнаружении тупиков

Способы предотвращения тупиков:

I способ: исключается 2-е условие; процесс запрашивает все ресурсы сразу, но система будет работать не эффективно, кроме того, программист может неправильно предусмотреть количество необходимых ресурсов; может возникнуть ситуация бесконечного откладывания.

II способ: исключает 1-е и 3-е условие; если процесс пытается захватить новые ресурсы, а система не может их предоставить, то у этого процесса отбираются все ресурсы. Недостаток – можем потерять всю работу, которая делалась до этого, опять может возникнуть ситуация бесконечного откладывания. Поэтому, чтобы исключить потери, можно решать задачу модульно.

III способ: исключение четвертого условия; все ресурсы выстраиваются в заранее определенном порядке. Круговое ожидание исключения. Недостаток – нельзя перескакивать через ресурсы (изменять порядок).

IV способ: «*алгоритм банкира*». Рассматривает каждый запрос по мере поступления и проверяет, приведет ли его удовлетворение к безопасному состоянию. Предусматривает:

1) знание количества ресурсов каждого вида; 2) знание max количества ресурсов для каждого процесса;
3) знание, сколько ресурсов занимает каждый процесс; 4) в системе разрешено захватывать и освобождать ресурсы по одному; 5) понятие надежности/ненадежности: система находится в надежном состоянии, если при наличии свободных ресурсов хотя бы 1 процесс может завершиться до конца.

Обход тупика:

Тупики можно предотвратить структурно, построив систему таким образом, что тупиковая ситуация никогда не возникнет по построению (если позволить процессу использовать только один ресурс в любой момент времени, то необходимое для возникновения тупика условие циклического ожидания не возникнет). Тупиков также можно избежать, если перенумеровать все ресурсы и затем требовать от процессов создания запросов в строго возрастающем порядке.

Обнаружение тупика:

Если систему чувствует, что процессы не выходят из системы по квоте и эффективность падает, рисуется граф состояний процессов и ресурсов. Затем, в матричном представлении графа ищем любой процесс, который может быть завершен, и удаляем его. Если граф редуцировался до конца, – мы избежали тупика.

Восстановление системы после обнаружения тупика:

наиболее распространенный способ устранить тупик – завершить выполнение одного или более процессов, чтобы впоследствии использовать его ресурсы. Также в системе можно использовать контрольные точки и откат, нужно сохранять состояния процессов, чтоб иметь возможность восстановления предыдущих действий.

3) Система управления файлами. Решаемые задачи.

Доп. инфа Файловая система - это часть операционной системы, назначение которой состоит в том, чтобы обеспечить пользователю удобный интерфейс при работе с данными, хранящимися на диске, и обеспечить совместное использование файлов несколькими пользователями и процессами.

В широком смысле понятие "файловая система" включает:
совокупность всех файлов на диске,
наборы структур данных, используемых для управления файлами, такие, например, как каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске,
комплекс системных программных средств, реализующих управление файлами, в частности: создание, уничтожение, чтение, запись, именование, поиск и другие операции над файлами.

4) Методы защиты памяти в различных системах организации.

Ответ:

Для защиты компьютера 360 компания IBM приняла следующее решение: она разделила память на блоки по 2 Кбайт и назначила каждому блоку 4-битовый защитный код. Регистр PSW (Program Status Word – слово состояния программы) содержал *4-битовый* ключ. Аппаратура IBM 360 перехватывала все попытки работающих процессов обратиться к любой части памяти, чей защитный код отличался от содержимого регистра слова состояния программы. Так как только операционная система могла изменять коды защиты и ключи, предотвращалось вмешательство пользовательских процессов в дела друг друга и в работу операционной системы.

Альтернативное решение сразу обеих проблем (защиты и перераспределения) заключается в оснащении машины двумя специальными аппаратными регистрами, называемыми **базовым** и **предельным регистрами.** При планировании процесса в базовый регистр загружается адрес начала раздела памяти, а в предельный регистр помещается длина раздела. К каждому автоматически формируемому адресу перед его передачей в память прибавляется содержимое базового регистра. Таким образом, если базовый регистр содержит величину 100 К, команда CALL 100 будет превращена в команду CALL 100K+100 без изменения самой команды. Кроме того, адреса проверяются по отношению к предельному регистру для гарантии, что они не используются для адресации памяти вне текущего раздела. Базовый и предельный регистры защищаются аппаратно, чтобы не допустить их изменений пользовательскими программами.

Неудобство этой схемы заключается в том, что требуется выполнять операции сложения и сравнения при каждом обращении к памяти. Операция сравнения может быть выполнена быстро, но сложение — это медленная операция, что обусловлено временем распространения сигнала переноса, за исключением тех случаев, когда употребляется специальная микросхема сложения.

Такая схема использовалась на первом суперкомпьютере в мире CDC 6600. В центральном процессоре Intel 8088 для первых IBM PC применялась упрощенная версия этой модели: были базовые регистры, но отсутствовали предельные. Сейчас такую схему можно встретить лишь в немногих компьютерах.

Доп.инфа:

При мультипрограммном режиме работы ЭВМ в ее памяти одновременно могут находиться несколько независимых программ. Поэтому необходимы специальные меры по предотвращению или ограничению обращений одной программы к областям памяти, используемым другими программами. Программы могут также содержать ошибки, которые, если этому не воспрепятствовать, приводят к искажению информации, принадлежащей другим программам. Последствия таких ошибок особенно опасны, если разрушению подвергнутся

программы операционной системы. Другими словами, надо исключить воздействие программы пользователя на работу программ других пользователей и программ операционной системы. Следует защищать и сами программы от находящихся в них возможных ошибок.
Таким образом, средства защиты памяти должны предотвращать [2]

- неразрешенное взаимодействие пользователей друг с другом,
- несанкционированный доступ пользователей к данным,
- повреждение программ и данных из-за ошибок в программах,
- намеренные попытки разрушить целостность системы,
- использование информации в памяти не в соответствии с ее функциональным назначением.

Чтобы воспрепятствовать разрушению одних программ другими, достаточно защитить область памяти данной программы от попыток записи в нее со стороны других программ, а в некоторых случаях и своей программы (защита от записи), при этом допускается обращение других программ к этой области памяти для считывания данных.
В других случаях, например при ограничениях на доступ к информации, хранящейся в системе, необходимо запрещать другим программам любое обращение к некоторой области памяти как на запись, так и на считывание. Такая защита от записи и считывания помогает в отладке программы, при этом осуществляется контроль каждого случая обращения за область памяти своей программы.
Для облегчения отладки программ желательно выявлять и такие характерные ошибки в программах, как попытки использования данных вместо команд или команд вместо данных в собственной программе, хотя эти ошибки могут и не разрушать информацию (несоответствие функционального использования информации).
Если нарушается защита памяти, исполнение программы приостанавливается и вырабатывается запрос прерывания по нарушению защиты памяти.
Защита от вторжения программ в чужие области памяти может быть организована различными методами. Но при любом подходе реализация защиты не должна заметно снижать производительность компьютера и требовать слишком больших аппаратурных затрат.
Методы защиты базируются на некоторых классических подходах, которые получили свое развитие в архитектуре современных ЭВМ. К таким методам можно отнести защиту отдельных ячеек, метод граничных регистров, метод ключей защиты [7].
Защита отдельных ячеек памяти организуется в ЭВМ, предназначенных для работы в системах управления, где необходимо обеспечить возможность отладки новых программ без нарушения функционирования находящихся в памяти рабочих программ, управляющих технологическим процессом. Это может быть достигнуто выделением в каждой ячейке памяти специального "разряда защиты". Установка этого разряда в "1" запрещает производить запись в данную ячейку, что обеспечивает сохранение рабочих программ. Недостаток такого подхода - большая избыточность в кодировании информации из-за излишне мелкого уровня защищаемого объекта (ячейка).
В системах с мультипрограммной обработкой целесообразно организовывать защиту на уровне блоков памяти, выделяемых программам, а не отдельных ячеек.
Метод граничных регистров (рис. 17.1) заключается во введении двух граничных регистров, указывающих верхнюю и нижнюю границы области памяти, куда программа имеет право доступа.

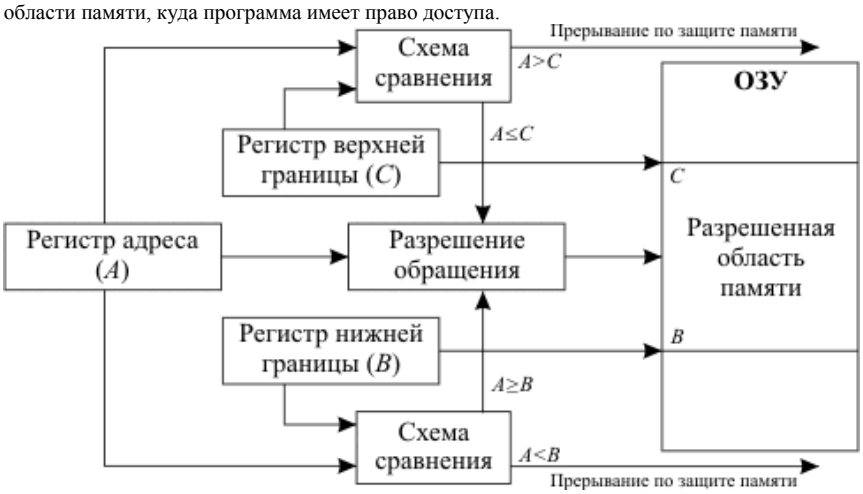

Рис. 17.1. Защита памяти методом граничных регистров

При каждом обращении к памяти проверяется, находится ли используемый адрес в установленных границах. При выходе за границы обращение к памяти не производится, а формируется запрос прерывания, передающий управление операционной системе. Содержание граничных регистров устанавливается операционной системой при загрузке программы в память.
Модификация этого метода заключается в том, что один регистр используется для указания адреса начала защищаемой области, а другой содержит длину этой области.
Метод граничных регистров, обладая несомненной простотой реализации, имеет и определенные недостатки. Основным из них является то, что этот метод поддерживает работу лишь с непрерывными областями памяти.
Метод ключей защиты, в отличие от предыдущего, позволяет реализовать доступ программы к областям памяти, организованным в виде отдельных модулей, не представляющих собой единый массив.
Память в логическом отношении делится на одинаковые блоки, например, страницы. Каждому блоку памяти ставится в соответствие код, называемый ключом защиты памяти, а каждой программе, принимающей участие в мультипрограммной обработке, присваивается код ключа программы. Доступ программы к данному блоку памяти для чтения и записи разрешен, если ключи совпадают (то есть данный блок памяти относится к данной программе) или один из них имеет код 0 (код 0 присваивается программам операционной системы и блокам памяти, к которым имеют доступ все программы: общие данные, совместно используемые подпрограммы и т. п.). Коды ключей защиты блоков памяти и ключей программ устанавливаются операционной системой.
В ключе защиты памяти предусматривается дополнительный разряд режима защиты. Защита действует только при попытке записи в блок, если в этом разряде стоит 0, и при любом обращении к блоку, если стоит 1. Коды ключей защиты памяти хранятся в специальной памяти ключей защиты, более быстродействующей, чем оперативная память.
Функционирование этого механизма защиты памяти поясняется схемой на рис. 17.2.

Рис. 17.2. Защита памяти методом ключей защиты

При обращении к памяти группа старших разрядов адреса ОЗУ, соответствующая номеру блока, к которому производится обращение, используется как адрес для выборки из памяти ключей защиты кода ключа защиты, присвоенного операционной системой данному блоку. Схема анализа сравнивает ключ защиты блока памяти и ключ программы, находящийся в регистре слова состояния программы (ССП), и вырабатывает сигнал "Обращение разрешено" или сигнал "Прерывание по защите памяти". При этом учитываются значения режима обращения к ОЗУ (запись или считывание), указываемого триггером режима обращения ТгРО, и режима защиты, установленного в разряде режима обращения (РРО) ключа защиты памяти.

Средства защиты памяти в персональной ЭВМ

Защита памяти в персональной ЭВМ [4] делится на защиту при управлении памятью и защиту по привилегиям.

Средства защиты при управлении памятью осуществляют проверку

- превышения эффективным адресом длины сегмента,
- прав доступа к сегменту на запись или только на чтение,
- функционального назначения сегмента.

Первый механизм базируется на методе граничных регистров. При этом начальные адреса того или иного сегмента программы устанавливаются операционной системой. Для каждого сегмента фиксируется его длина. При попытке обращения по относительному адресу, превышающему длину сегмента, вырабатывается сигнал нарушения защиты памяти.

При проверке функционального назначения сегмента определяются операции, которые можно проводить над находящимися в нем данными. Так, сегмент, представляющий собой стек программы, должен допускать обращение как на запись, так и на чтение информации. К сегменту, содержащему программу, можно обращаться только на исполнение. Любое обращение на запись или чтение данных из этого сегмента будет воспринято как ошибочное. Здесь наблюдается некоторый отход от принципов Неймана в построении ЭВМ, в которых утверждается, что любая информация, находящаяся в ЗУ, функционально не разделяется на программу и данные, а ее идентификация проводится лишь на стадии применения данной информации. Очевидно, что такое развитие вызвано во многом появлением мультипрограммирования и необходимостью более внимательного рассмотрения вопросов защиты информации.

Защита по привилегиям фиксирует более тонкие ошибки, связанные, в основном, с разграничением прав доступа к той или иной информации.

В какой-то степени защиту по привилегиям можно сравнить с классическим методом ключей защиты памяти. Различным объектам (программам, сегментам памяти, запросам на обращение к памяти и к внешним устройствам), которые должны быть распознаны процессором, присваивается идентификатор, называемый уровнем привилегий. Процессор постоянно контролирует, имеет ли текущая программа достаточные привилегии, чтобы

- выполнять некоторые команды,
- выполнять команды ввода-вывода на том или ином внешнем устройстве,
- обращаться к данным других программ,
- вызывать другие программы.

На аппаратном уровне в процессоре различаются 4 уровня привилегий. Наиболее привилегированными являются программы на уровне 0. Число программ, которые могут выполняться на более высоком уровне привилегий, уменьшается от уровня 3 к уровню 0. Программы уровня 0 действуют как ядро операционной системы. Поэтому уровни привилегий обычно изображаются в виде четырех колец защиты (рис. 17.3).

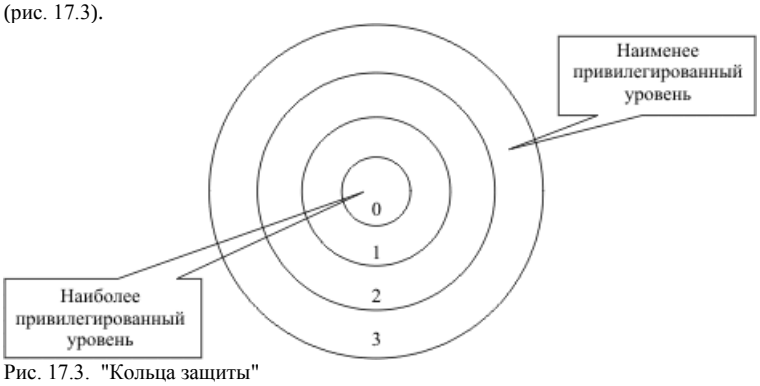

Рис. 17.3. "Кольца защиты"

Как правило, распределение программ по кольцам защиты имеет следующий вид:

уровень 0 - ядро ОС, обеспечивающее инициализацию работы, управление доступом к памяти, защиту и ряд других жизненно важных функций, нарушение которых полностью выводит из строя процессор;

уровень 1 - основная часть программ ОС (утилиты);

уровень 2 - служебные программы ОС (драйверы, СУБД, специализированные подсистемы программирования и др.);

уровень 3 - прикладные программы пользователя.

Конкретная операционная система не обязательно должна поддерживать все четыре уровня привилегий. Так, ОС UNIX работает с двумя кольцами защиты: супервизор (уровень 0) и пользователь (уровни 1,2,3). Операционная система OS/2 поддерживает три уровня: код ОС

работает в кольце 0, специальные процедуры для обращения к устройствам ввода-вывода действуют в кольце 1, а прикладные программы выполняются в кольце 3.
Простую незащищенную систему можно целиком реализовать в одном кольце 0 (в других кольцах это сделать невозможно, так как некоторые команды доступны только на этом уровне).
Уровень привилегий сегмента определяется полем DPL уровня привилегий его дескриптора. Уровень привилегий запроса к сегменту определяется уровнем привилегий RPL, закодированном в селекторе. Обращение к сегменту разрешается только тогда, когда уровень привилегий сегмента не выше уровня запроса. Обращение к программам, находящимся на более высоком уровне привилегий (например, к утилитам операционной системы из программ пользователя), возможно с помощью специальных аппаратных механизмов - шлюзов вызова.
При страничном преобразовании адреса применяется простой двухуровневый механизм защиты: пользователь (уровень 3) / супервизор (уровни 0,1,2), указываемый в поле U/S соответствующей таблицы страниц.
При сегментно-страничном преобразовании адреса сначала проверяются привилегии при доступе к сегменту, а затем - при доступе к странице. Это дает возможность установить более высокую степень защиты отдельных страниц сегмента.

---

5) Виды межпроцессного взаимодействия, их сравнения.

Конспект: К средствам межпроцессорного взаимодействия относятся:
1)сигналы
2)каналы
3)именованные каналы (ФИФО)
4)семафоры (мониторы, мютексы, секвенторы)
5)разделяемые файлы
6)сообщения
7)сокеты

передача сообщений. Этот метод межпроцессного взаимодействия использует два примитива: send и receive, которые скорее являются системными вызовами, чем структурными компонентами языка (что отличает их от мониторов и делает похожим на семафоры). Поэтому их легко можно поместить в библиотечные процедуры, например

```
send(destination, &message);
receive(source, &message);
```

Первый запрос посылает сообщение заданному адресату, а второй получает сообщение от указанного источника (или от любого источника, если это не имеет значения). Если сообщения нет, второй запрос блокируется до поступления сообщения либо немедленно возвращает код ошибки.

Каналы
Именованные каналы – любой процесс может послать информацию или забрать канал.
Неименованные каналы – могут быть созданы только между родственными процессами.
Канал имеет вход, выход и буфер. Недостаток канала: если в канале есть информация (сообщения), то мы можем считывать, пока не освободим канал; объем считываемой информации определяется пользователем.
Если описатель пишет в канал, а он заполнен, то блокируется (это не критическая блокировка). Также есть проблемы при освобождении (попытка считать пустой буфер) из-за заполнения буферов.
Очереди сообщений или сообщение считаются более информационно емкими в межпроцессорном взаимодействии и являются составляющей частью ОС и разделенным операционным ресурсом.
В системе существует или может существовать несколько очередей сообщений. Поэтому, каждая очередь именованная. Можно создавать мультиплексирование сообщений.
Система поддерживает специальные структуры для каждой очереди в виде связного списка и списко-указатель на само сообщение. Надо знать в очереди: адрес, размер и кому предназначено.
Каналы
Средства локального межпроцессного взаимодействия реализуют высокопроизводительную, детерминированную передачу данных между процессами в пределах одной системы.
К числу наиболее простых и в то же время самых употребительных средств межпроцессного взаимодействия принадлежат каналы, представляемые файлами соответствующего типа. Стандарт POSIX-2001 различает именованные и безымянные каналы. Напомним, что первые создаются функцией mkfifo() и одноименной служебной программой, а вторые - функцией pipe(). Именованным каналам соответствуют элементы файловой системы, ко вторым можно обращаться только посредством файловых дескрипторов. В остальном эти разновидности каналов эквивалентны.
Взаимодействие между процессами через канал может быть установлено следующим образом: один из процессов создает канал и передает другому соответствующий открытый файловый дескриптор. После этого процессы обмениваются данными через канал при помощи функций read() и write().
Сигналы
Как и каналы, сигналы являются внешне простым и весьма употребительным средством локального межпроцессного взаимодействия, но связанные с ними идеи существенно сложнее, а понятия - многочисленнее.
Согласно стандарту POSIX-2001, под сигналом понимается механизм, с помощью которого процесс или поток управления уведомляют о некотором событии, произошедшем в системе, или подвергают воздействию этого события. Примерами подобных событий могут служить аппаратные исключительные ситуации и специфические действия процессов. Термин "сигнал" используется также для обозначения самого события.
Говорят, что сигнал генерируется (или посылается) для процесса (потока управления), когда происходит вызвавшее его событие (например, выявлен аппаратный сбой, отработал таймер, пользователь ввел с терминала специфическую последовательность символов, другой процесс обратился к функции kill() и т.п.). Иногда по одному событию генерируются

сигналы для нескольких процессов (например, для группы процессов, ассоциированных с некоторым управляющим терминалом).
В момент генерации сигнала определяется, посылается ли он процессу или конкретному потоку управления в процессе. Сигналы, сгенерированные в результате действий, приписываемых отдельному потоку управления (таких, например, как возникновение аппаратной исключительной ситуации), посылаются этому потоку. Сигналы, генерация которых ассоциирована с идентификатором процесса или группы процессов, а также с асинхронным событием (к примеру, пользовательский ввод с терминала) посылаются процессу.
В каждом процессе определены действия, предпринимаемые в ответ на все предусмотренные системой сигналы. Говорят, что сигнал доставлен процессу, когда взято для выполнения действие, соответствующее данным процессу и сигналу. сигнал принят процессом, когда он выбран и возвращен одной из функций sigwait().

В интервале от генерации до доставки или принятия сигнал называется ждущим. Обычно он невидим для приложений, однако доставку сигнала потоку управления можно блокировать. Если действие, ассоциированное с заблокированным сигналом, отлично от игнорирования, он будет ждать разблокирования.
У каждого потока управления есть маска сигналов, определяющая набор блокируемых сигналов. Обычно она достается в наследство от родительского потока.

Сокеты – это средства межпроцессорного взаимодействия между процессорами разных вычислительных систем

Для обозначения коммуникационного узла, обеспечивающего прием и передачу данных для объекта (процесса), был предложен специальный объект — *сокет* (socket). Сокеты создаются в рамках определенного коммуникационного домена, подобно тому, как файлы создаются в рамках файловой системы. Сокеты имеют соответствующий интерфейс доступа в файловой системе, и так же как обычные файлы, адресуются некоторым целым числом — дескриптором. Однако в отличие от обычных файлов, сокеты представляют собой виртуальный объект, который существует, пока на него ссылается хотя бы один из процессов.

- *Сокет датаграмм* (datagram socket), через который осуществляется теоретически ненадежная, несвязная передача пакетов.
- *Сокет потока* (stream socket), через который осуществляется надежная передача потока байтов без сохранения границ сообщений. Этот тип сокетов поддерживает передачу экстренных данных.
- *Сокет пакетов* (packet socket), через который осуществляется надежная последовательная передача данных без дублирования с предварительным установлением связи. При этом сохраняются границы сообщений.
- *Сокет низкого уровня (raw* socket), через который осуществляется непосредственный доступ к коммуникационному протоколу.

Наконец, для того чтобы независимые процессы имели возможность взаимодействовать друг с другом, для сокетов должно быть определено *пространство имен.* Имя сокета имеет смысл только в рамках коммуникационного домена, в котором он создан.

Примитивы сокетов для TCP протокола:

- SOCKET – создать точку коммуникации
- BIND – назначить сокету локальный адрес
- LISTEN – обозначить готовность к установке соединения
- ACCEPT – заблокировать вызывающую сторону до прибытия запроса на соединение
- CONNECT – попытка установить соединение
- SEND, WRITETO, SENDTO – послать сообщение сокету в зависимости от типа сокета и сообщения
- RECEIVE, READ, RECIEVEFROM – принять сообщение

CLOSE – разорвать соединение

Коммуникационный канал создаётся при создании сокета между источником и получателем и имеет такие характеристики:

- Коммуникационный протокол
- Локальный адрес источника
- Локальный адрес процесса источника
- Удалённый адрес получателя

Удалённый адрес процесса получателя
Семафор должен быть доступен всем процессам. По этому он должен находиться в адресном пространстве супервизора или (ядро ОС)
И операции с ним производятся в режиме ядра. Помимо собственных задач семафора в структуре семафора храниться идентификатор процесса выполневшего последнюю операцию с семафором, число процессов ожидающих увеличения значения семфора, число процессов, ожид, когда значение семафора будет =0.

Семафоры

В 1965 году Дейкстра (E. W. Dijkstra) предложил использовать целую переменную для подсчета сигналов запуска, сохраненных на будущее [96]. Им был предложен новый тип переменных, так называемые **семафоры**, значение которых может быть нулем (в случае отсутствия сохраненных сигналов активизации) или некоторым положительным числом, соответствующим количеству отложенных активизирующих сигналов.

Дейкстра предложил две операции, down и up (обобщения sleep и wakeup). Операция down сравнивает значение семафора с нулем. Если значение семафора больше нуля, операция down уменьшает его (то есть расходует один из сохраненных сигналов активации) и просто возвращает управление. Если значение семафора равно нулю, процедура down не возвращает управление процессу, а процесс переводится в состояние ожидания. Все операции проверки значения семафора, его изменения и перевода процесса в состояние ожидания выполняются как единое и неделимое **элементарное действие**. Тем самым гарантируется, что после начала операции ни один процесс не получит доступа к семафору до окончания или блокирования операции. Элементарность операции чрезвычайно важна для разрешения проблемы синхронизации и предотвращения состояния состязания.

Операция up увеличивает значение семафора. Если с этим семафором связаны один или несколько ожидающих процессов, которые не могут завершить более раннюю операцию down, один из них выбирается системой (например, случайным образом) и ему разрешается завершить свою операцию down. Таким образом, после операции up, примененной к семафору, связанному с несколькими ожидающими процессами, значение семафора так и останется равным 0, но число ожидающих процессов уменьшится на единицу. Операция увеличения значения семафора и активизации процесса тоже неделима. Ни один процесс не может быть блокирован во время выполнения операции up, как ни один процесс не мог быть блокирован во время выполнения операции wakeup в предыдущей модели.

В оригинале Дейкстра использовал вместо down и up обозначения P и V соответственно. Мы не будем в дальнейшем использовать оригинальные обозначения, поскольку тем, кто не знает датского языка, эти обозначения ничего не говорят (да и тем, кто знает язык, говорят немного). Впервые обозначения down и up появились в языке Algol 68.

Разделяемые сегменты памяти

В стандарте POSIX-2001 разделяемый объект памяти определяется как объект, представляющий собой память, которая может быть параллельно отображена в адресное пространство более чем одного процесса.

Таким образом, процессы могут иметь общие области виртуальной памяти и разделять содержащиеся в них данные. Единицей разделяемой памяти являются сегменты. Разделение памяти обеспечивает наиболее быстрый обмен данными между процессами.

Работа с разделяемой памятью начинается с того, что один из взаимодействующих процессов посредством функции shmget() создает разделяемый сегмент, специфицируя первоначальные права доступа к нему и его размер в байтах. Чтобы получить доступ к разделяемому сегменту, его нужно присоединить (для этого служит функция shmat()), т. е. разместить сегмент в виртуальном пространстве процесса. После присоединения, в соответствии с правами доступа, процессы могут читать данные из сегмента и записывать их (быть может, синхронизируя свои действия с помощью семафоров). Когда разделяемый сегмент становится ненужным, его следует отсоединить с помощью функции shmdt(). Аппарат разделяемых сегментов предоставляет нескольким процессам возможность одновременного доступа к общей области памяти. Обеспечивая корректность доступа, процессы тем или иным способом должны синхронизировать свои действия. В качестве средства синхронизации удобно использовать семафор.