БИЛЕТ № 14

1) Особенности генерации, инсталяции, инициализации ОС.

При покупке вычислительной системы приобретается и программное обеспечение для выполнения тех функций, ради которых эта ВС и покупалась. Как правило, в это ПО включены все необходимые программы для полной поддержки всех соответствующих частей ВС. Таким образом, покупается исходный вариант операционной системы или *дистрибутив.* В него входят различные варианты драйверов для различных аппаратных компонент, а также программы для обеспечения различных режимов работы ВС. Это драйверы CD-ROMa, сетевых Ethernet карт, SCSI-устройств, Sound Blastera, принтера и т.д. Причем в состав дистрибутива могут входить варианты драйверов для поддержки различных локальных шин (VESA, PCI) и других аппаратных платформ. Кроме этого, дистрибутив имеет все необходимые программы, поддерживающие работу вычислительной системы в различных режимах работы, например, однопрограммном или многопрограммном, многопроцессорном, сетевом.

Для нормального функционирования вычислительной системы в реальных условиях эксплуатации совокупность программ всего исходного ПО не нужна и из всего их множества выбирается некое подмножество программ, обеспечивающих работу вычислительной установки в установленных пользователем режимах и на имеющемся оборудовании. Процесс отбора из дистрибутива тех программных модулей, которые будут использоваться называется *генерацией* операционной системы. Процесс генерации выполняется либо с использованием специального "языка генерации", либо с помощью организации диалога пользователя с системой в результате которого определяются желания и требования пользователя к функционированию системы, а система в соответствии с этим определяет совокупность необходимых программ, поддерживающих эти требования.

Место, где находится резиденция системы, называется " резидентным" томом, а сама система "резиденцией". Из всех программных модулей в резидентном томе часть используется по мере необходимости, а некоторые из них обеспечивают базовое функционирование ВС и составляют *резидентные* программы, находящиеся в системной области оперативной памяти. Совокупность программ, обеспечивающих функционирование ВС и находящихся в системной области оперативной памяти, составляет ядро супервизора.

Некоторые программы, связанные с выполнением функций ОС, но не находящиеся постоянно в оперативной памяти называются *транзитными*. Эти программы вызываются в память по мере необходимости. Кроме этого, пользователь может по своему желанию определить некоторые программы как резидентные. Все остальные программы являются транзитами, вызываются динамически и могут затирать друг друга.

При функционировании вычислительной системы пользователь может менять свои представления о ее функционировании. Кроме этого может меняться и состав оборудования. Процесс локальной настройки ОС по желанию пользователя называется *инсталляцией*. Признаком, по которому можно отличить инсталляцию от генерации является то, что, если при настройке операционной системы не используется дистрибутив — это инсталляция, если требуется дистрибутив — генерация.

При загрузке ОС необходимо выполнить связывание, размещение всех частей, входящих в ядро супервизора, т.е. размещение резидентных программ на своих местах, формирование специальных системных структур данных в области данных операционной системы, формирование постоянной области, активизацию процессов для начала работы операционной системы. Этот процесс называется *инициализацией* операционной системы.

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:

Доп.инфа:

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.

---

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.