1. **Дать определение. Системное програмное обеспечение.**

Системное ПО, в состав которого входят операционная система, трансляторы языков и обслуживающие программы, управляет доступом к аппаратному обеспечению. СПО делится на обрабатывающие программы (производящие преобразование информации – компиляторы, редакторы связей, драйверы) и управляющие (упр. Процессами, заданиями, памятью, в/выв., данными (ФС))

2. **Дать характеристику и сравнить средства межпроцессного взаимодействия. Достоинства и недостатки**

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.

Очереди сообщений

Механизм очередей сообщений позволяет процессам взаимодействовать, обмениваясь данными. Данные передаются между процессами дискретными порциями, называемыми сообщениями. Процессы выполняют над сообщениями две основные операции - прием и отправку. Процессы, отправляющие или принимающие сообщение, могут приостанавливаться, если требуемую операцию невозможно выполнить немедленно. В частности, могут быть отложены попытки отправить сообщение в заполненную до отказа очередь, получить сообщение из пустой очереди и т.п. ("операции с блокировкой"). Если же указано, что приостанавливать процесс нельзя, "операции без блокировки" либо выполняются немедленно, либо завершаются неудачей.
Прежде чем процессы смогут обмениваться сообщениями, один из них должен создать очередь. Одновременно определяются первоначальные права на выполнение операций для различных процессов, в том числе соответствующих управляющих действий над очередями.

Семафоры

Согласно определению стандарта POSIX-2001, семафор - это минимальный примитив синхронизации, служащий основой для более сложных механизмов синхронизации, определенных в прикладной программе.
У семафора есть значение, которое представляется целым числом в диапазоне от 0 до 32767.
Семафоры создаются (функцией semget()) и обрабатываются (функцией semop()) наборами (массивами), причем операции над наборами с точки зрения приложений являются атомарными. В рамках групповых операций для любого семафора из набора можно сделать следующее: увеличить значение, уменьшить значение, дождаться обнуления.

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.

3. Когда можно считать, что процесс зафиксирован в системе.

Т.к., процесс – это траэктория процессора в адресном пространстве, то говорить о его фиксировании нельзя. По конспекту, ЗАДАЧА считается зафиксированой в системе (переход из задания в задачу), если ей выделены системнвые ресурсы (как минимум, PCB). ОС не может отклонить задачу

4. Роль PCB для управления процессами.

Для того чтобы операционная система могла выполнять операции над процессами, каждый процесс представляется в ней некоторой структурой данных. Эта структура содержит информацию, специфическую для данного процесса:
состояние, в котором находится процесс;
программный счетчик процесса или, другими словами, адрес команды, которая должна быть выполнена для него следующей;
содержимое регистров процессора;
данные, необходимые для планирования использования процессора и управления памятью (приоритет процесса, размер и расположение адресного пространства и т. д.);
учетные данные (идентификационный номер процесса, какой пользователь инициировал его работу, общее время использования процессора данным процессом и т. д.);
сведения об устройствах ввода-вывода, связанных с процессом (например, какие устройства закреплены за процессом, таблицу открытых файлов).
Соответственно, роль PCB состоит в том, чтобы в любой момент времени предоставлять ОС информацию о процессе для управления.

5. Виды загрузчиков. Их основные отличия

4 функции загрузчиков:
Настройка адресных констант
Выделение памяти
Связывание
Собственно загрузка
-Абсолютный
Выполняет только 4 функцию (по Симону). Как часть загрузчика может рассматриваться редактор связей (хотя он выполняется отдельно) Абс. загрузчик выделяет загружаемой программе память в пределах памяти, выделеной процессу
-Настраиваемый
До этапа выполнения программы в модулях описываются векторы переходов (внутри – и внешнемодульные), и константы, которые нужно переопределить. Загрузчик при просмотре этих записей определяет абсолютные адреса гастраиваемых величин.
-Непосредственно связывающий
Динамически выполняет все 4 функции. Выделяет память постранично, выполняет связывание программы динамически, по мере необходимости. Для этого к модулю прикрепляются 2 таблицы: ESD и RLD. (таблицы векторов переходов и адресных констант)

6. Особенности выделения диковой памяти в HPFS

Смотри вопрос 56.

7. Состав системы управления заданиями

8. Чем определяется насыщение системы прерываний.

Насыщение системы прерываний возможно при неправильной настройке приоритетов (зацикливание приоритетов), а также при чрезмерном увеличении устройств и процессов, вызывающих прерывания – система постоянно находится в режиме прерывания

9. Функции системы управления памятью.

Чтобы обеспечить эффективный контроль использования памяти, ОС должна выполнять следующие функции:
-отображение адресного пространства процесса на конкретные области физической памяти;
-распределение памяти между конкурирующими процессами;
-контроль доступа к адресным пространствам процессов;
-выгрузка процессов (целиком или частично) во внешнюю память, когда в оперативной памяти недостаточно места;
-учет свободной и занятой памяти.

10. Концепция виртуальной памяти.

Суть концепции виртуальной памяти заключается в следующем. Информация, с которой работает активный процесс, должна располагаться в оперативной памяти. В схемах виртуальной памяти у процесса создается иллюзия того, что вся необходимая ему информация имеется в основной памяти. Для этого, во-первых, занимаемая процессом память разбивается на несколько частей, например страниц. Во-вторых, логический адрес (логическая страница), к которому обращается процесс, динамически транслируется в физический адрес (физическую страницу). И наконец, в тех случаях, когда страница, к которой обращается процесс, не находится в физической памяти, нужно организовать ее подкачку с диска. Для контроля наличия страницы в памяти вводится специальный бит присутствия, входящий в состав атрибутов страницы в таблице страниц.

Таким образом, в наличии всех компонентов процесса в основной памяти необходимости нет. Важным следствием такой организации является то, что размер памяти, занимаемой процессом, может быть больше, чем размер оперативной памяти. Принцип локальности обеспечивает этой схеме нужную эффективность.

Возможность выполнения программы, находящейся в памяти лишь частично, имеет ряд вполне очевидных преимуществ.

Программа не ограничена объемом физической памяти. Упрощается разработка программ, поскольку можно задействовать большие виртуальные пространства, не заботясь о размере используемой памяти.

Поскольку появляется возможность частичного помещения программы (процесса) в память и гибкого перераспределения памяти между программами, можно разместить в памяти больше программ, что увеличивает загрузку процессора и пропускную способность системы.

Объем ввода-вывода для выгрузки части программы на диск может быть меньше, чем в варианте классического свопинга, в итоге каждая программа будет работать быстрее.

Таким образом, возможность обеспечения (при поддержке операционной системы) для программы «видимости» практически неограниченной (характерный размер для 32-разрядных архитектур 232 = 4 Гбайт) адресуемой пользовательской памяти (логическое адресное пространство) при наличии основной памяти существенно меньших размеров (физическое адресное пространство) – очень важный аспект.

Но введение виртуальной памяти позволяет решать другую, не менее важную задачу – обеспечение контроля доступа к отдельным сегментам памяти и, в частности, защиту пользовательских программ друг от друга и защиту ОС от пользовательских программ. Каждый процесс работает со своими виртуальными адресами, трансляцию которых в физические выполняет аппаратура компьютера. Таким образом, пользовательский процесс лишен возможности напрямую обратиться к страницам основной памяти, занятым информацией, относящейся к другим процессам.

11. Что такое просмотр команд вперед и какая характеристика прри этом улучшается.

См. Ткаченко, Луцкий. Улучшается время выборки-распаковки и выполнения команды

12. Когда используется 'свопинг'.

Когда при использовании виртуальной памяти все физические страницы заняты, и какой-то процесс требует выделения ему новой страницы или запрашивает страницу, отсутствующую в физической ОП. Тогда запускается механизм своппинга – записи выбраной по определенному алгоритму страницы на диск и замещения ее выделяемой страницей или затребованой процессом.

13. Классификация способа размещения данных в иерархической памяти.

Стратегии размещения информации в памяти

Те или иные стратегии размещения информации в памяти применяются для того, чтобы определить, в какое место основной памяти следует помещать поступающие программы и данные. В технической литературе чаще всего описываются три стратегии:

· Стратегия наиболее подходящего. Поступающее задание помещается в тот свободный участок основной памяти, в котором ему наиболее «тесно», так что остается минимально возможное неиспользуемое пространство. Для многих людей выбор наиболее подходящего кажется интуитивно самой рациональной стратегией.

· Стратегия первого подходящего. Поступающее задание помещается в первый встретившийся свободный участок основной памяти достаточного размера. Выбор первого подходящего по размеру свободного участка также кажется интуитивно рациональным, поскольку он позволяет быстро принять решение о размещении задания.

· Стратегия наименее подходящего. На первый взгляд подобный подход кажется весьма странным. Однако более тщательное рассмотрение показывает, что выбор наименее подходящего по размеру также имеет сильные интуитивные аргументы в свою пользу. Этот принцип говорит о том, что при помещении программы в основную память нужно занимать свободный участок, имеющий наиболее далекий размер, т. е. максимальный возможный свободный участок. Интуитивный аргумент в пользу такого подхода достаточно прост: после помещения программы в большой свободный участок остающийся свободный участок зачастую также оказывается большим и, таким образом, в нем можно разместить относительно большую новую программу.

14. Дать определение тупика.

Предположим, что несколько процессов конкурируют за обладание конечным числом ресурсов. Если запрашиваемый процессом ресурс недоступен, ОС переводит данный процесс в состояние ожидания. В случае когда требуемый ресурс удерживается другим ожидающим процессом, первый процесс не сможет сменить свое состояние. Такая ситуация называется тупиком (deadlock).

15. Назначение системы управления данными

Смотреть 81

16. В чем идея технологии MEMORY CHANNEL.

Данная технология предназначена для эффективной организации кластерных систем на базе модели разделяемой памяти. Передачи через Memory Channel программируются как доступ в память, а не как доступ к внешним устройствам. В каждом узле кластера память организуется по принципу виртуальной адресации: адрес состоит из номера страницы и собственно адреса внутри страницы. В каждом компьютере в ходе инициализации выделяется определенное количество физических страниц памяти, предназначенных для разделения с остальными узлами кластера. Данная концепция организации межпроцессорного взаимодействия ориентирована на значительное уменьшение задержек, вызваемых обычно операционной системой, обслуживающей прием и передачу сообщений. Доступ к удаленным областям памяти выполняется посредством обычных команд чтения (load) и записи (store), работающих как с обычными страницами виртуальной памяти без обращения к операционной системе. Межузловые обмены происходят при выполнении команд чтения или записи, адресующих память, расположенную на другом узле. Для формирования правильных запросов на каждом узле создаются специальные таблицы управления страницами памяти: одна для приема удаленных обращений к локальной памяти, другая для формирования собственных обращений к удаленной памяти. Memory Channel состоит из адаптеров, работающих на шине PCI, коммутатора и дуплексных линий длиной до 10 м. Максимальная пропускная способность составляет 132 Мбайт/с. Коммутаторы Memory Channel содержат серьезные ограничения, препятствующие масштабируемости сети. Протокол достаточно сложен, содержит развитые средства подтверждения доставки, что, по мнению разработчиков, должно гарантировать отсутствие потери пакетов.

17. Особенности операционных систем локальных вычислительных сетей.

Сетевая операционная система составляет основу любой вычислительной сети. Каждый компьютер в сети в значительной степени автономен, поэтому под сетевой операционной системой в широком смысле понимается совокупность операционных систем отдельных компьютеров, взаимодействующих с целью обмена сообщениями и разделения ресурсов по единым правилам - протоколам.
В узком смысле сетевая ОС - это операционная система отдельного компьютера, обеспечивающая ему возможность работать в сети.
В состав ОС входят:
Средства управления локальными ресурсами компьютера: функции распределения оперативной памяти между процессами, планирования и диспетчеризации процессов, управления процессорами в мультипроцессорных машинах, управления периферийными устройствами и другие функции управления ресурсами локальных ОС.
Средства предоставления собственных ресурсов и услуг в общее пользование - серверная часть ОС (сервер). Эти средства обеспечивают, например, блокировку файлов и записей, что необходимо для их совместного использования; ведение справочников имен сетевых ресурсов; обработку запросов удаленного доступа к собственной файловой системе и базе данных; управление очередями запросов удаленных пользователей к своим периферийным устройствам.
Средства запроса доступа к удаленным ресурсам и услугам и их использования - клиентская часть ОС (редиректор). Эта часть выполняет распознавание и перенаправление в сеть запросов к удаленным ресурсам от приложений и пользователей, при этом запрос поступает от приложения в локальной форме, а передается в сеть в другой форме, соответствующей требованиям сервера. Клиентская часть также осуществляет прием ответов от серверов и преобразование их в локальный формат, так что для приложения выполнение локальных и удаленных запросов неразличимо.
Коммуникационные средства ОС, с помощью которых происходит обмен сообщениями в сети. Эта часть обеспечивает адресацию и буферизацию сообщений, выбор маршрута передачи сообщения по сети, надежность передачи и т.п., то есть является средством транспортировки сообщений.

18. Особенность создания процесса при рекурсивном обращении.

Дело в том, что при рекурсивном создании дочернего процесса (фактически, бесконечном), быстро забивается таблица PCB и:

1. Не может быть выделено место под другие, нужные системе процессы
2. Из-за этого произойдет тупиковая ситуация: (какой-то задаче понадобится результат выполнения другого процесса, который она не может создать)

В ранних версиях UNIX можно было запустить пользовательскую программу, которая бы бесконечно создавала процессы, тем самым блокируя работу всей системы. Поэтому в новых версиях, стали вводить ограничения на количество дочерних процессов

19. Назвать механизмы неявной реализации когерентности.

Проблема когерентности памяти для мультипроцессоров и устройств ввода/вывода имеет много аспектов. Обычно в малых мультипроцессорах используется аппаратный механизм, называемый протоколом, позволяющий решить эту проблему. Такие протоколы называются протоколами когерентности кэш-памяти. Существуют два класса таких протоколов:
Протоколы на основе справочника (directory based). Информация о состоянии блока физической памяти содержится только в одном месте, называемом справочником (физически справочник может быть распределен по узлам системы).
Протоколы наблюдения (snooping). Каждый кэш, который содержит копию данных некоторого блока физической памяти, имеет также соответствующую копию служебной информации о его состоянии. Централизованная система записей отсутствует. Обычно кэши расположены на общей (разделяемой) шине и контроллеры всех кэшей наблюдают за шиной (просматривают ее) для определения того, не содержат ли они копию соответствующего блока.
Имеются две методики поддержания описанной выше когерентности. Один из методов заключается в том, чтобы гарантировать, что процессор должен получить исключительные права доступа к элементу данных перед выполнением записи в этот элемент данных. Этот тип протоколов называется протоколом записи с аннулированием (write ivalidate protocol), поскольку при выполнении записи он аннулирует другие копии. Это наиболее часто используемый протокол как в схемах на основе справочников, так и в схемах наблюдения. Исключительное право доступа гарантирует, что во время выполнения записи не существует никаких других копий

элемента данных, в которые можно писать или из которых можно читать: все другие кэшированные копии элемента данных аннулированы. Чтобы увидеть, как такой протокол обеспечивает когерентность, рассмотрим операцию записи, вслед за которой следует операция чтения другим процессором. Поскольку запись требует исключительного права доступа, любая копия, поддерживаемая читающим процессором должна быть аннулирована (в соответствии с названием протокола). Таким образом, когда возникает операция чтения, произойдет промах кэш-памяти, который вынуждает выполнить выборку новой копии данных. Для выполнения операции записи мы можем потребовать, чтобы процессор имел достоверную (valid) копию данных в своей кэш-памяти прежде, чем выполнять в нее запись. Таким образом, если оба процессора попытаются записать в один и тот же элемент данных одновременно, один из них выиграет состязание у второго (мы вскоре увидим, как принять решение, кто из них выиграет) и вызывает аннулирование его копии. Другой процессор для завершения своей операции записи должен сначала получить новую копию данных, которая теперь уже должна содержать обновленное значение.
Альтернативой протоколу записи с аннулированием является обновление всех копий элемента данных в случае записи в этот элемент данных. Этот тип протокола называется протоколом записи с обновлением (write update protocol) или протоколом записи с трансляцией (write broadcast protocol). Обычно в этом протоколе для снижения требований к полосе пропускания полезно отслеживать, является ли слово в кэш-памяти разделяемым объектом, или нет, а именно, содержится ли оно в других кэшах. Если нет, то нет никакой необходимости обновлять другой кэш или транслировать в него обновленные данные.
А теперь из конспекта:

Алгоритм MESI (в простонародии алгоритм с обратной записью)является одним из алгоритмов, обеспечивающих когерентность (согласованность) памяти.
Small intro:
Механизмы ралиизации когерентности могут быть явными и неявными для программиста.
Все архитектуры делятся на 4 категории:

1. Явное размещение и явный доступ (все операции над данными явны с точки зрения размещения и доступа. Команды Send() и Receive())
2. Неявное размещение и неявный доступ (доступ к данным прозрачен для программиста).Т.е. оперирование данными происходит на уровне команд читать/писать без явного указания адресов.
3. Неявное размещение (как в страничной организации) явный доступ. Используется разделяемое множество страниц на ВУ. При запросе страницы система автоматически обеспечивает согласование.
4. Явное размещение с указанием разделяемых(? Или раздела) страниц и неявный доступ с помощью команд Load() и Store().При этом используется технология **MEMORY CHANNEL**. Каждый компьютер имеет виртуальную память и разделяемые страницы, отображение этих страниц имеется во всех остальных компьютерах системы их удаление производится с использованием специальной команды. При этом используется специальная сетевая карта , обеспечивающая взаимодействие память-память.

MESI – один из алгоритмов *неявной* реализации когерентности. Идея его такова : Память сосредоточена.Подключение к ней идет через шину. Все транзакции также реализуются только через шину, поэтому есть возможность их отслеживания. Каждая строка в *кэшах* имеет следующие состояния :

1. M – модифицирована (операция R/W разрешена только в этом модуле)
2. E – монопольно копирована (операция R/W разрешена во всех модулях)
3. S – множественно копирована (операция R/W разрешена во всех модулях , где есть копия страницы)
4. I – запрещена к использованию.

Периодически по шине отправляются циклы опроса состояния строк.
© Все тот же Вова, респект ему за это

20. В чем заключается проблема синхронизации при передаче данных от процесса к процессу

В том, что для выполнения передачи необходима готовность процесса принять данные – с вытекающими последствиями типа появления точек синхронизации, возщможности появления тупиков и т. Д. Для решения используются примитивы синхронизации

21. Как обрабатывается прерывание ввода/вывода.

1. Обращение к супервизору
2. Сохранение текущего PSW в старый PSW. Выполнение дешифрации прерывания
3. Сохранение нового PSW в текущий
4. Переход по SVC на обработчик прерывания
5. Подготовка операций ввода/вывода
6. В адресное слово канала запис. Адрес начала канальной программы – дальше Луцкий
7. Запуск канальной программы
8. Команда SIO
9. Передача ус-вом сигнала об усп./неусп. старте – в слове сост. Канала CSW.
10. Обработчик прерываний запрашывает ответ о успешном старте
11. Ввод-вывод
12. Обработчик анализирует CSW на сигнал о завершении ввода/вывода, записывает старый PSW в текущий

Все. Дальше – продолжение основной программы

22. Способы хранения свободного места во внешней памяти

Используются такие методы, как битовые карты (вопрос 26), списки блоков свободного пространства, комбинированные.

23. Дать определение - Страничная фрагментация.

Всюду пишут, что при страничной организации памяти как раз отсутствует фрагментация… По-моему, это случай, когда размер страницы слишком мал и/или множество прцессов требует свои страницы – тогда система постоянно переключает страницы, и эффективность е работы понижается.

24. Место расположение программы – драйвера

Так как драйвер является обрабатывающим системным ПО, он не входит в состав ядра но загружается вместе с загрузкой ОС

25. Примеры обрабатывающих программ.

компиляторы, редакторы связей, драйверы, текстовые редакторы….

26. Особенности хранения свободного места в HPFS

Для определения свободен сектор или занят HPFS использует битмапы в которых каждый бит соответствует одному сектору. Если бит содержит 1 то это означает, что сектор занят, иначе он свободен. Если бы на весь диск был бы только один битмап то для его подкачки приходилось бы перемещать головки чтения/записи в среднем через половину диска. Чтобы избежать этого HPFS разбивает диск на "полосы" (Bands) длиной по 8 мегабайт и хранит битмапы свободных секторов в начале или конце каждой полосы. При этом битмапы соседних полос располагаются рядом:

```
+--------- 16MB --------+    *** - Use/Free sector bitmap.
!                       !
+--!-----------+----------!--+-------------+-------------+
!***  Полоса 0 ! Полоса 1  ***!***  Полоса 2 ! Полоса 3  ***!
+--------------+-------------+-------------+-------------+
0MB            8MB           16MB           24MB           32MB
```

Из этого следует что расстояние между двумя битмапами равно 16MB. Размер полосы (8MB) может быть изменен в следующих версиях HPFS т.к. на него нет прямых завязок. HPFS определяет размер полосы при чтении управляющих блоков с диска во время выполнения операции FSHelperAttach.

Сейчас размер битмапа равен 2K. (8MB/512/8 = 2K).

Полоса находящаяся в центре диска используется для хранения каталогов. Эта полоса называется Directory Band. Однако если она будет полностью заполнена HPFS начнет располагать каталоги файлов в других полосах.

© Степанов

27. Трудности планирования параллельных процессов.

Добавляется планирование в пространстве; появляется проблема синхронизации процессов. Если в системе нет одинаковых устройств то очереди обрабатываются по линейной схеме => планирование только во времени. Иначе планирование во времени и в пространстве. Для планирования параллельных процессор используют семиуровневую модель планирования. Семиуровневая модель – схема системы планирования с учетом (Маштабируемость, Разделяемость, Параллельность) 1. Предварительное (входное) планирование исходного потока заявок (задача фильтрации) 2. Структурный анализ взаимосвязи входного потока заявок по ресурсам с определением общих ресурсов (Анализ) 3. Структурный анализ заявок и определение возможности распараллеливания (Задача распараллеливания) 4. Адаптация распределения работ соответственно особенностям ВС (Задача адаптирования) 5. Составление плана – расписания выполнения взаимосвязанных процедур. Оптимизация плана по времени решения и кол-ву ресурсов (Задача Оптимизации) 6. Планирование потока задач претендующих на захват времени процессора на каждый прцессор – задача распределения. 7. Выделение процессорного времени, активизация задач. Перераспределение работ в ВС, при отказе оборудования (задача распределения – перераспределения).

28. Условия возникновения тупика.

См. 14

29. Уровни планирования в однопроцессорной системе.

Имеем все три уровня планирования – верхний (отбрасывание невыполнимых заявок на входе), средний (в данном случае – временное планирование, имеем один процессор) и нижний

30. Способы организации кеш памяти.

Смотреть 197

31. Особенности распределенных операционных систем.

* *Прозрачность сети (transparency)*

Прозрачность сети требует, чтобы детали сети были скрыты от конечных пользователей. Так, пользователь истинно прозрачной распределенной системы должен находиться под впечатлением, что он использует одиночную мини — (супер) — ЭВМ. При этом перемещение файлов должно осуществляться без изменения их имен. Если это требование реализовано, то написание программ для распределенной системы не должно быть сложнее создания программ для сосредоточенной системы. Определим некоторые аспекты решения этой задачи [107]:

* имя любого ресурса системы не должно быть связано с его местоположением;
* имя должно быть уникальным в глобальном смысле, и не имеет значения в каком месте системы оно будет использовано;
* имя ресурса желательно связать с соответствующим применением этого ресурса, независимо от места его применения (рис.3.6.).

Таким образом, прозрачность сети действительно может эффективно решить проблему скрытости местоположения ресурсов от пользователя, однако для диспетчеров распределенных систем ее реализация сталкивается со многими конфликтными ситуациями.

* *Локальная автономность (local autonomy)*

Это естественная норма для пользователя (администратора) узла в распределенной системе заключается в возможности контроля и управления ресурсами. Однако здесь разработчики сталкиваются с проблемой обеспечения прозрачности всей сети и возникающими при этом конфликтными проблемами — все команды должны иметь одинаковый эффект во всех узлах сети, но в этом случае пользователь другого узла уже не может так же использовать данную команду (ресурс), а может вообще не иметь доступа к нему. Поэтому в данной ситуации от локальной автономности прийдется отказаться.

* *Оптимизация управления ресурсами*

В некоторых случаях пользователь может иметь желание точно знать о местоположении ресурсов (а иногда и управлять ими). Это может потребовать оптимизации всей системы, путем размещения часто используемых ресурсов в те места, где они были бы легко доступны. При этом значительно усложняются функции управления всей системой, а вместе с этим и РОС. Тогда необходимо будет либо вообще отказаться от механизма прозрачности, либо найти другое средство для реализации оптимизации в сети, либо совсем отказаться от такой реконфигурации системы.

* *Разнородность (heterogeneity)*

Трудно добиться полной прозрачности сети. Проблема возникает сразу в двух аспектах. Во-первых, скрытность аппаратного обеспечения не позволяет определить, какие ресурсы могут быть использованы в данном узле, особенно это касается неоднородной структуры, где узлы сильно разнятся между собой по своим аппаратным ресурсам, программному обеспечению и структурам данных. Во-вторых, более сложная проблема возникает в том случае, если узлы используют различные ОС [107]. Очень сложно выявить и учесть все это, учитывая требования к прозрачности сети. В DECnet, например, количество операций, используемых между узлами с гетерогенными свойствами, — это подмножество операций, которое употребляется в гомогенной структуре [64].

Свойства распределенных операционных систем:

- Отсутствие общей памяти приводит к тому, что нельзя определить общее состояние системы с помощью множества совместных переменных, а невозможность совместного обращения к памяти и различия в задержке передач сообщений приводят к тому, что при определении состояния какого-либо элемента системы из двух различных точек можно получить разные результаты, в зависимости от порядка предшествующих событий;
- Распределенная система распределяет выполняемые работы в узлах системы, исходя из соображений повышения пропускной способности всей системы;
- Распределенные системы имеют высокий уровень организации параллельных вычислений.

Два фактора увеличивают вероятность отказов отдельных элементов (по сравнению с централизованными системами):

1. Большое число элементов системы.
2. Надежность систем связи обычно меньше, чем надежность информационных систем.

Однако надежность всей системы в целом в распределенных операционных системах весьма высока за счет специфических свойств, позволяющих исключить или ослабить эффект отказа отдельных элементов системы (исправление вышедших из строя элементов, переконфигурация системы и т.д.).

32. Как и когда в системе определяется приоритет системных задач?

33. Чем определяется глубина системы прерывания.

глубина системы прерывания – это степень вложенности прерываний. Она определяется количеством старых PSW которые можно поместить в постоянную область памяти или в стеке.

34. Отличие загрузочного модуля от объектного. Участие системных программ в их создании.

Загрузочный модуль – исполняемый файл программы. Он содержит объектный код программы, точку входа в программу, а также некоторую другую полезную информацию (в т. ч. отладочную).

Объектный модуль – файл, содержащий объектный код нескольких функций/процедур программы, отладочную информацию, но не достаточных для самостоятельного выполнения.

В процессе компиляции исходных файлов программы, для каждого файла создается свой объектный модуль, содержащий объектный код описанных в файле процедур/функций. Если в исходнике определена какая-то функция, но не описана, то она считается «импортированной» (ее «экспортирует» другой модуль).

При помощи специальной утилиты (lib) можно соединить несколько объектных модулей в библиотеку.

При помощи «связчика» (linker) объектные модули и библиотеки соединяются в загрузочный модуль. При этом один из модулей должен содержать специфично для языка названную процедуру (для C – main()), которая будет вызвана после входа в программу. При этом все «импортированные» объектными модулями функции должны содержатся в связываемых модулях.

.

35. Чем определяется размер таблицы страниц.

Размер таблицы страниц определяется размерностью виртуального адресного пространства системы и размером одной страницы.

N = V / S, где N – количество страниц, V – размер виртуального адресного пространства, S – размер страницы.

Соответственно, чем больше количество страниц, тем больший объем занимает таблица страниц:

T = N*K, где T – размер таблицы страниц; K – размер одной записи таблицы страниц.

В системах с двухуровневой таблицей (386+) T можно значительно уменьшить за счет того, что если в рамках какой-либо из записей таблицы 1-го уровня мы не используем ни одной страницы, можем не создавать таблицу 2-го уровня.

36. Функции системы управления памятью. Смотри вопр. № 9

37. Состав системы управления заданиями. Смотри вопр. № 7

Система управления заданиями предназначена для управления прохождением задач на многопроцессорных вычислительных установках (в том числе кластерных). Она позволяет автоматически распределять вычислительные ресурсы между задачами, управлять порядком их запуска, временем работы, получать информацию о состоянии очередей.

38. Какой тип системы прерываний реализован в IBM PC.

Комплекс программно-аппаратных средств, обеспечивающих прохождение сигнала прерывания от устройства к части ядра отвечающей за обслуживание прерывания. Различают следующие классы прерываний: -от схем контролера; -внешние прерывания; -от портов вв/выв; -обращение супервизора; -программные; -прерыв по Кеш;

Реализована система двухуровневого прерыв –> сигнал прерыв // приоритет вызывающих программ

39. Характеристика связного и несвязного распределения памяти.

40. Перечислить и дать сравнительную характеристику примитивам синхронизации

Для синхронизации процессов используются примитивы. Некоторые из них (в основном, семафоры и мониторы) реализованы в ядре (для синхронизации ввода/вывода, переключения контекста (переход по прерыванию) и т.д.).

Примитивы синхронизации могут быть реализованы следующим образом:

1) Самые простые примитивы — флажки (примитивы взаимоисключения). Если флажок равен 0, т.е. условие ложно, то процесс не может войти в КУ, т.к. там уже находится другой процесс; если флажок равен 1 (условие истинно), то процесс входит в КУ, обнуляя флажок (если имеется очередь процессов к ОР, то в КУ входит наиболее приоритетный из них). На основе флажков реализованы алгоритмы синхронизации Деккера. Флажки — программная реализация решения задачи взаимного исключения. Это самый низкий уровень.

2) Команда test_and_set — аппаратное средство синхронизации (более высокий уровень). Это специальная команда, выполняющая вместе (неделимо) 3 действия:

- чтение значения переменой;
- запись ее значения в область сохранения;
- установка нового значения этой переменной.

2. Семафор — некоторая (защищенная) переменная, значение которой можно считывать и изменять только при помощи специальных операций P и V. Преимущество по сравнению с test_and_set — разделение действий на отдельные.

Операция P(S): проверяет значение семафора S; если S>0, то S:=S-1, иначе (5=0) ждать (по S).

Операция V(S): проверяет, есть ли процессы, приостановленные по данному семафору; если есть, то разрешить одному из них продолжить свое выполнение (войти в КУ); если нет, то S=S+1.

Операция P должна стоять до входа в КУ , а операция V — после выхода из КУ.

Недостатки:

1) громоздкость — в каждом процессе каждый КУ должен окаймляться операциями P и V;
2) невозможность решения целого ряда задач с помощью таких примитивов.

3.Монитор — примитив высокого уровня. Здесь "забор" ставится вокруг ОР, что удобнее (чем семафоры), т.к. ресурсов в несколько раз меньше, чем процессов (их КУ) и ставить "заборы" вокруг КУ слишком(80) громоздко. Можно сказать, что монитор — это некоторый программный модуль, в котором объединены ОР и подпрограммы для работы с ними. При обращении к ОР, т.е. соответствующей процедуре монитора для работы с ОР, осуществляется вход в монитор. В каждый конкретный момент времени может выполняться только одна процедура монитора — это и есть решение задачи взаимного исключения. Если процесс обратился кг процедуре монитора, а другой процесс уже находится внутри монитора, то обратившийся процесс блокируется и переводится во внешнюю очередь процессов — блокированных по входу в монитор. Процесс, вошедший в монитор (т.е. выполняющий его процедуру), также может быть блокирован им, если не выполняется некоторое *условие* X для выполнения процесса. Условие X в мониторе — это некоторая переменная типа condition. С ней связывается некоторая внутренняя очередь процессов, блокированных по условию X. Внутренняя очередь приоритетнее внешней. Для блокирования процесса по переменной X и помещения его во внутреннюю очередь по этой переменной используется операция монитора WAIT(X). При выполнении операции SIGNAL(X) из очереди, связанной с переменной X, извлекается и разблокируется первый процесс. Если внутренняя очередь пуста, то в монитор разрешается войти первому процессу из внешней очереди.

41. Основная особенность страничной организации памяти.

Страничная организация памяти — это такой способ управления памятью, при котором пространство адресов памяти разбивается на блоки фиксированной длины. При страничной организации все пространство оперативной памяти физически делится на отрезки равной длины, называемые *физическими страницами.* Программы и данные делятся на называемые *страницами* содержательные части того же размера, что и физические страницы оперативной памяти . Таким образом, одну страницу информации можно загрузить в одну физическую страницу. Для каждой текущей задачи создается **таблица страниц.**

Диспетчер памяти для каждой страницы формирует соответствующий дескриптор. Дескриптор содержит так называемый бит присутствия. Если он = 1, это означает, что данная страница сейчас размещена в ОП. Если он = 0, то страница расположена во внешней памяти. Защита страничной памяти основана на контроле уровня доступа к каждой странице.

для обеспечения целостности, секретности и взаимной изоляции выполняющихся программ должны быть предусмотрены различные режимы доступа к страницам, которые реализуются с помощью специальных индикаторов доступа в элементах таблиц страниц.

Следствием такого использования является значительный рост таблиц страниц каждого пользователя. Одно из решений проблемы сокращения длины таблиц основано на введении многоуровневой организации таблиц. Частным случаем многоуровневой организации таблиц является сегментация при страничной организации памяти.

Основное достоинство страничного способа распределения памяти - ***минимально возможная фрагментация (эффективное распределение памяти).*** Недостатки: 1) потери памяти на размещение таблиц страниц

2) потери процессорного времени на обработку таблиц страниц (диспетчер памяти).

3) Программы разбиваются на страницы случайно, без учета логических взаимосвязей, имеющихся в коде ⇒ межстраничные переходы осуществляются чаще, чем межсегментные + трудности в организации разделения программных модулей между выполняющими процессами

42. Стратегии борьбы с тупиками.

Если коротко, тупик – это ситуация, из которой система не может выйти или ситуация, когда процесс ждет события, которое никогда не произойдет. Можно выделить 4 стратегии борьбы с тупиками:

1. Предотвращение тупика. Требуется создать условия, исключающие возможность возникновения тупиковых ситуаций (Возникновение тупика невозможно, если нарушается одно из четырех необходимых условий возникновения тупиковых ситуаций.). Однако этот метод часто приводит к нерациональному использованию ресурсов.

2. Обход тупика. Этот метод предусматривает менее жесткие ограничения, чем первый и обеспечивает лучшее использование ресурсов. Метод учитывает возможность возникновения тупиков, и при увеличении вероятности тупиковой ситуации принимаются меры по обходу тупика. На практике используется алгоритм "банкира". При использовании алгоритма "банкира" подразумевается, что:

1) системе заранее известно количество имеющихся ресурсов;

a. система знает, сколько ресурсов может максимально потребоваться процессу;
b. число пользователей (процессов), обращающихся к ресурсам, известно и постоянно;
c. система знает, сколько ресурсов занято в данный момент времени этими процессами (в общем и каждым из них);
d. процесс может потребовать ресурсов не больше, чем имеется в системе.

3. Обнаружение тупиков. Требуется установить сам факт возникновения тупиковой ситуации, причем, точно определить те процессы и ресурсы которые в нее включены.

Восстановление после тупиков. Необходимо устранить тупиковую ситуацию, чтобы система смогла продолжить работу, а процессы, попавшие а тупиковую ситуацию, смогли завершиться и освободить занимаемые ими ресурсы. Восстановление — это серьезная проблема, так что в большинстве систем оно осуществляется путем полного выведения из решения одного или более процессов, попавших в тупиковую ситуацию. Затем выведенные процессы обычно перезапускаются с самого начала, так что вся или почти вся проделанная процессами работа теряется.

43. Назначение программы системного вывода. Смотри вопр. № 182

44. Почему точек входа PCB меньше, чем количество задач. Смотри вопр. № 109 , 256

Это связано с тем, что структура ОС имеет либо оверлейную, либо динамически — последовательную структуру иерархического типа, и нет необходимости создавать для программ, которые никогда не будут находиться одновременно в оперативной памяти, отдельные PCB. При такой организации легко учитывать приоритеты системных процессов, выстроив их по приоритетам заранее при инициализации системы. Блоки управления проблемными (пользовательскими) процессами создаются в процессе активизации процессов динамически. Все PCB находятся в выделенной системной области памяти.

45. Дать пояснение кеш памяти с прямым отображением

Кэш-память - это быстродействующая память, расположенная между центральным процессором и основной памятью. Вместе с основной памятью она входит в иерархическую структуру и ее действие эквивалентно быстрому доступу к основной памяти. Чтобы добиться высокой эффективности в кэш-память из основной памяти должны попадать только самые нужные данные. Для этого существуют различные методы отображения кэш-памяти. Простейшим видом отображения является прямое отображение. Вся информация в памяти и в кэш-памяти организуется в виде логических блоков, которые называются кэш-строками. Данные передаются из основной памяти в кэш-память в виде кэш-строк. Кэш-строка обычно состоит из четырех слов. Слово состоит из 32 или 64 бит информации. Однако в кэш-памяти может находиться только часть кэш-строк, хранимых в основной памяти. Если обозначить число слов в кэш-памяти через 2^a, а число слов в основной памяти - через 2^b, то относительное количество данных в кэш-памяти можно определить как 2^a, деленное на 2^b, или 2^(a-b), где a < b. Данные в памяти разбиты на так называемые классы эквивалентности. Каждый класс эквивалентности состоит из 2^(b-a) элементов, или кэш-строк. Каждому классу эквивалентности в основной памяти соответствует индекс и кэш-строка в кэш-памяти. Просматривая кэш-память в поисках информации, процессор ищет нужный класс эквивалентности и индекс. Если этот класс эквивалентности содержит нужную кэш-строку, происходит кэш-попадание. Если нет, процессор увеличивает индекс на единицу, извлекает нужную кэш-строку из основной памяти и заменяет ими имеющийся в кэш-памяти индекс и класс эквивалентности.

Таким образом Кэш с прямым отображением (размещением) является самым простым типом буфера. В идейном отношении это наиболее простая форма отображения, которая обеспечивает очень быстрый доступ хотя бы потому, что каждая кэш-строка состоит из четырех слов. Адрес памяти однозначно определяет строку кэша, в которую будет помещен блок информации. При этом предполагается, что оперативная память разбита на блоки и каждому такому блоку в буфере отводится всего одна строка.

46. Виды связного распределения памяти.

На самых первых вычислительных машинах в каждый момент времени мог работать только один человек, и все ресурсы машины 1 оказывались в распоряжении этого пользователя. Предъявление счетов за использование машинного времени производилось по самому простому принципу — поскольку пользователю машина предоставлялась целиком, ему приходилось платить за все ее ресурсы независимо от того, занимала ли реально его программа эти ресурсы. В связи с этим обычные механизмы выписки счетов

ориентировались на чисто календарное время. Пользователю машина выделялась на определенное время, а затем предъявлялся счет на почасовую оплату этого времени. В некоторых современных вычислительных системах, работающих с разделением времени, для расчетов с пользователями применяются гораздо более сложные алгоритмы.

Первоначально каждый пользователь сам писал всю программу, необходимую для реализации конкретной прикладной задачи, включая подробные процедуры ввода-вывода в машинных кодах. Но очень скоро коды программ, необходимых для реализации базовых функций ввода-вывода, начали объединяться в так называемые системы управления вводом-выводом (IOCS). Благодаря этому у пользователей, которым требовались определенные операции ввода-вывода, отпала необходимость самим" непосредственно кодировать эти операции,— они получили возможность вызывать соответствующие подпрограммы ввода-вывода, эффективно выполняющие \ реальную работу. Тем самым было обеспечено существенное упрощение и ускорение процесса кодирования программ. Создание систем управления вводом-выводом можно, по-видимому, считать началом развития концепции современных операционных систем. Организация памяти в типичном случае связного распределения для одного пользователя показана на рис. 7.2.

Размер программ в обычном случае ограничивается емкостью имеющейся основной памяти, однако существует возможность выполнения программ, превышающих по размеру основную память, благодаря использованию так называемых оверлейных сегментов. Если какой-то конкретный модуль программы не работает в течение всего периода выполнения программы, то из внешней памяти можно выбрать

другой модуль и записать его в основную память на место уже не нужного модуля.

Оверлейный режим предоставляет программисту возможность как бы расширить ограниченные рамки основной памяти. Однако ручная реализация оверлейного режима требует продуманного и трудоемкого планирования. Программа со сложной оверлейной структурой может весьма трудно поддаваться модификации. Как мы увидим в последующих главах, проблема оверлейных сегментов, контролируемых самим программистом, отпадает благодаря появлению систем виртуальной памяти.

47. Дать определение. Область сохранения. Ее роль в организации процессов. Смотри вопр. № 180

Область сохранения - область в которую записывается информация о прерванном процессе, и откуда система берет данные при восстановлении процесса.

48. Особенности программного обеспечения распределенных систем обработки информации.

49. Основная особенность чистой страничной организации памяти.

Многие преимущества, присущие сегментации пространства адресов программы, имеют место и при страничной организации пространства адресов оперативной памяти. Как и в случае сегментации, реализованное с помощью аппаратного оборудования вычисление адресов делает сложную систему адресации понятной пользователю. Однако фиксированная длина страниц приводит к важным различиям между этими двумя методами. Поскольку размер страниц фиксирован, страница может оказаться недостаточно большой, чтобы в ней поместился целиком какой-то содержательный раздел программы. Поэтому задача установления внешних связей при страничной организации памяти не так проста, как в случае сегментации программы. Но при фиксированной длине страниц значительно упрощается распределение памяти. Поскольку все страницы одинаковой длины, ввести в оперативную память новую страницу легче, чем новый сегмент. Ее можно либо поместить в незанятую физическую страницу, либо вытеснить другую страницу, чтобы освободилось место для новой. В любом случае для того, чтобы освободить место для новой страницы, не требуется по-новому располагать остальные страницы в оперативной памяти.

50. Дать определение. Инициализация системы. Какая программа выполняет эти действия.

Смотри вопр. № 303

В виду того, что ядро ОС представляет собой совокупность взаимосвязанных модулей, имеющих связь друг с другом, процесс структурной организации модулей называется инициализацией. Инициализация системы - процесс создания ядра системы который включает не только перезапись программ, но и системных структур, обеспечивающих знания системы о ее параметрах. Программа - загрузчик.

51. Дать определение - Обрабатывающие программы ОС

Системное программное обеспечение включает в себя обрабатывающие программы и управляющие программы. Если управл. программы отвечают за управление процессами, заданиями, памятью, вв/выв и файловой системой, - то обраб. программы – это

программы, которые преобразуют информацию, а также программы выполнения стандартных (в рамках ОС) функций, обработки исключительных ситуаций.

52. Назначение цепочки ожидания.

Для синхронизации в PCB имеются четыре поля:

1-2. Поля для организации цепочки связи.

3-4. Поля для организации цепочки ожидания.

В цепочке ожидания, в поле 3 указывается адрес PCB вызываемого процесса, если вызываемый процесс занят. В поле 4 занятого процесса находится число процессов, которые ожидают данный.

Если процесс А пытается вызвать процесс В, а у процесса В в PCB занята цепочка связей, то есть он является вызываемым по отношению к другим процессам, тогда адрес процесса В записывается в цепочке ожидания PCB процесса А, а в поле счетчика ожидания PCB процесса В добавляется 1. Как только процесс В завершает выполнение своих функций, он передает управление вызывающему процессу следующим образом: В проверяет состояние своего счетчика ожидания, и, если счетчик больше 0, то среди PCB других процессов ищется первый (по приоритету или другим признакам) процесс, в поле 3 PCB которого стоит имя ожидаемого процесса, в данном случае В, тогда управление передается этому процессу.

53. Каким образом осуществляется синхронизация процессов с помощью PCB ?

В каждом PCB есть поле состояния процесса. Все блоки управления системными процессами располагаются в порядке убывания приоритетов и находятся в системной области памяти. Если приоритеты системных блоков можно определить заранее, то для проблемных процессов необходима таблица приоритетов проблемных программ. Каждый блок PCB имеет стандартную структуру, фиксированный размер, точку входа, содержит указанную выше информацию и дополнительную информацию для синхронизации процессов. Для синхронизации в PCB имеются четыре поля:

1-2. Поля для организации цепочки связи.

3-4. Поля для организации цепочки ожидания.

В цепочке связи указывается адрес PCB вызываемого (поле 1) и вызывающего (поле 2) процесса.

В цепочке ожидания, в поле 3 указывается адрес PCB вызываемого процесса, если вызываемый процесс запит. В поле 4 занятого процесса находится число процессов, которые ожидают (71) данный.

Если процесс А пытается вызвать процесс В, а у процесса В в PCB занята цепочка связей, то есть он является вызываемым по отношению к другим процессам, тогда адрес процесса В записывается в цепочке ожидания PCB процесса А, а в поле счетчика ожидания PCB процесса В добавляется 1. Как только процесс В завершает выполнение своих функций, он передает управление вызывающему процессу следующим образом; В проверяет состояние своего счетчика ожидания, и, если счетчик больше 0, то среди PCB других процессов ищется первый (по приоритету или другим признакам) процесс, в поле 3 PCB которого стоит имя ожидаемого процесса, в данном случае В, тогда управление передастся этому процессу.

54. Дать характеристику и сравнить применение семафоров и мониторов при синхронизации процессов. Смотри вопр. № 68

55. Недостатки распределения памяти MFT. Смотри вопр. № 259

Диск NTFS условно делится на две части. Первые 12% диска отводятся под так называемую MFT зону - пространство, в которое растет метафайл MFT (об этом ниже). Запись каких-либо данных в эту область невозможна. MFT-зона всегда держится пустой - это делается для того, чтобы самый главный, служебный файл (MFT) не фрагментировался при своем росте. Остальные 88% диска представляют собой обычное пространство для хранения файлов. Свободное место диска, однако, включает в себя всё физически свободное место - незаполненные куски MFT-зоны туда тоже включаются. Механизм использования MFT-зоны таков: когда файлы уже нельзя записывать в обычное пространство, MFT-зона просто сокращается (в текущих версиях операционных систем ровно в два раза), освобождая таким образом место для записи файлов. При освобождении места в обычной области MFT зона может снова расширится. При этом не исключена ситуация, когда в этой зоне остались и обычные файлы: метафайл MFT все-таки может фрагментироваться, хоть это и было бы нежелательно. Резюме:

Базисом NTFS является главная таблица файлов (Master File Table, MFT). MFT изначально резервирует под себя одну восьмую часть раздела (примерно 12%). 1) Если место на разделе заканчивается, MFT сокращается в два раза, освобождая для файлов пользователя свободное пространство. Процедура может повторяться несколько раз. При появлении незанятого места MFT снова резервирует под себя 12% от объема раздела, что приводит к нежелательному эффекту - фрагментации MFT. При этом эфф-сть работы с NTFS-диском падает. 2)Неудачное планирование резко уменьшает эфф-сть системы

56. Особенности хранения свободного места в HPFS. Смотри вопр. № 26 , 175 , 189 , 310

Главное отличие - базовые принципы размещения файлов на диске и принципы хранения информации о местоположении файлов. Благодаря этим принципам HPFS имеет высокую производительность и отказоустойчивость, является надежной файловой системой.

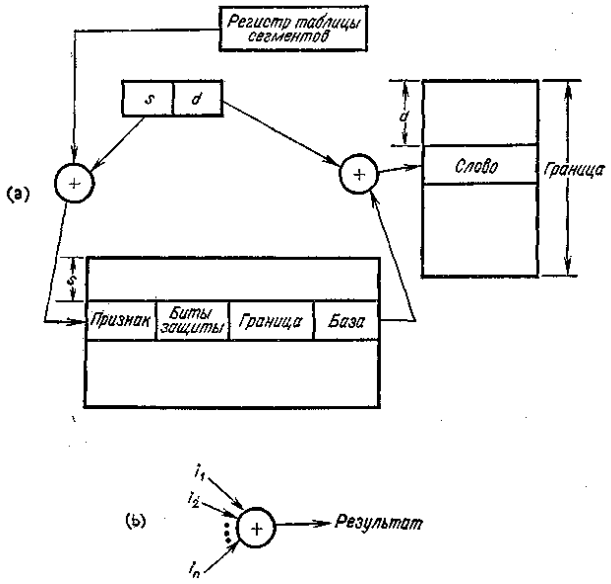
- Дисковое пространство в HPFS выделяется не кластерами (как в FAT), а *блоками.* HPFS распределяет пространство на диске не кластерами как в FAT, а физическими секторами по 512 байт, что не позволяет ее использовать на жестких дисках, имеющих другой размер сектора. Эти секторы принято называть блоками. Чтобы уменьшить фрагментацию диска, при распределении пространства под файл HPFS стремится, по возможности, размещать файлы в последовательных смежных секторах. В современной реализации размер блока взят равным одному сектору, но в принципе он мог бы быть и иного размера. (По сути дела, блок — это и есть кластер, только кластер всегда равен одному сектору). Размещение файлов в таких небольших блоках позволяет более эффективно использовать пространство диска, так как непроизводительные потери свободного места составляют в среднем всего (полсектора) 256 байт на каждый файл. Вспомним, что чем больше размер кластера, тем больше места на диске расходуется напрасно.

- Система HPFS стремится расположить файл в смежных блоках, или, если такой возможности нет, разместить его на диске таким образом, чтобы *экстенты* (фрагменты) файла физически были как можно ближе друг к другу. Такой подход существенно уменьшает время позиционирования головок записи/чтения жесткого диска и время ожидания (задержка между установкой головки чтения/записи на нужную дорожку). Напомним, что в FAT файлу просто выделяется первый свободный кластер.

Экстенты (extent) — фрагменты файла, располагающиеся в смежных секторах диска. Файл имеет по крайней мере один экстент, если он не фрагментирован, а в противном случае — несколько экстентов.

- Используется *метод* сбалансированных двоичных деревьев для хранения и поиска информации о местонахождении файлов (каталоги хранятся в центре диска, кроме того, предусмотрена автоматическая сортировка каталогов), что существенно повышает производительность HPFS (в сравнении с FAT).
- В HPFS предусмотрены специальные расширенные атрибуты файлов, позволяющие управлять доступом к файлам и каталогам.
- информация о местоположении файлов рассредоточена по всему диску, при этом записи каждого конкретного файла размещаются (по возможности) в смежных секторах и поблизости от данных об их местоположении;
- каталоги размещаются в середине дискового пространства;
- каталоги хранятся в виде бинарного сбалансированного дерева с записями, расположенными в алфавитном порядке.

Короче говоря, HPFS имеет мощные возможности и эффективно работает на дисках большой ёмкости. Однако эта файловая имеет и свои недостатки. Например, при повреждении начала раздела, где располагается информация, необходимая для начальной загрузки и указатель на корневой каталог, использование раздела HPFS станет невозможным и информация на нём будет безвозвратно утеряна. Кроме того, HPFS использует сектора размером 512 байт, которые не очень подходят для современных накопителей большой ёмкости.



57. Схема формирования исполнительного адреса при сегментной организации памяти.

В системе с сегментацией всякий адрес представляет собой пару *Is, d],* где *s* — имя сегмента, *ad* — смещение. Каждому заданию или процессу соответствует всегда присутствующая в памяти *таблица сегментов,* в которой каждому сегменту данного задания соответствует одна запись. С помощью этой таблицы система отображает программные адреса в истинные адреса оперативной памяти. Адрес таблицы хранится в аппаратном регистре, называемом *регистром таблицы сегментов.* В s-й *) строке таблицы находятся

1) *признак* (или *бит присутствия),* показывающий, присутствует ли s-й сегмент в данный момент в памяти;
2) *базовый адрес* s-го сегмента;
3) *граница,* указывающая количество ячеек, занимаемых данным сегментом;
4) биты защиты (не обязательные), используемые для контроля способа доступа.

Чтобы добраться до слова [s, *d*], надо *с* помощью регистра таблицы сегментов обратиться к таблице сегментов (рис. 4.4). В s-й строке таблицы указан адрес сегмента s в памяти. Его *d-я* ячейка содержит искомое слово [s, *d].* Следовательно, чтобы получить слово [s, *d],* потребуется два обращения к памяти: одно к таблице сегментов и одно к слову внутри сегмента. Поскольку сегменты бывают различной длины, не существует фиксированного верхнего предела для *d.* Поэтому для того, чтобы помешать заданию обращаться за пределы сегмента, необходимо знать значение границы.

Прежде чем система сможет вычислить адрес, аппаратным путем проверяется признак присутствия сегмента в оперативной памяти.. Если сегмент присутствует, то адрес в памяти можно вычислить автоматически, как описано в предыдущем абзаце. Если сегмент отсутствует в оперативной памяти, то происходит так называемое «прерывание из-за отсутствия сегмента», т. е. вырабатывается аппаратное прерывание, которое включает супервизорную программу обработки таких ситуаций. Эта программа отыскивает нужный сегмент во вспомогательной памяти и вводит его в оперативную. Если в оперативной памяти нет места для нового сегмента, то система освобождает его, выгнав один из находящихся в оперативной памяти сегментов на периферийное устройство.

58. Назначение - Система управления задачами. Смотри вопр. № 97 , 157 , 238

59. Почему необходимо решать задачу когерентности См. также №280

При наличии локальной, то есть принадлежащей данному процессору кэш-памяти, следующая необходимая ему команда с большой вероятностью будет находиться в кэш-памяти. В результате этого уменьшается количество обращений к шине и быстродействие системы возрастает. Вместе с тем возникает проблема *кэш-когерентности*. Эта проблема заключается в том, что если, скажем, двум процессорам для выполнения различных операций понадобилось значение V, это значение будет храниться в виде двух копий в кэшпамяти обоих процессоров. Один из процессоров может изменить это значение в результате выполнения своей команды и оно будет передано в оперативную память компьютера. Но в кэшпамяти второго процессора все еще хранится старое значение! Следовательно, необходимо обеспечить своевременное обновление данных в кэш-памяти всех процессоров компьютера.

60. Каким образом система отслеживает попадание в КЕШ

Кэш-память SRAM используется для хранения данных, которые непосредственно требуются процессору. Статистический анализ работы современных компьютеров показывает, что около 90% всех данных, которые запрашивает процессор, обычно находится в кэш-памяти; эта величина называется коэффициентом кэш-попадания. Если процессор находит нужную ему информацию в кэш-памяти, это называется кэш-попаданием, иначе ему приходится искать нужную информацию в DRAM; такая ситуация называется кэш-промахом.

Поскольку кэш-память SRAM работает быстрее, чем память DRAM, коэффициент кэш-попадания пропорционален эффективному времени ожидания всей системы памяти. Если обозначить коэфициент кэш-попаданий через H, а время ожидания кэш-памяти и DRAM - через Tc и Tm соответственно, то эффективное (среднее) время ожидания системы памяти, включающей в себя кэш-память, определяется как : `Teff= H * Tc + (1-H)*Tm`
Увеличение скорости, связанное с наличием кэш-памяти, определяется как отношение Tm к Teff и эскпоненциально увеличивается с возрастанием H: `S=Tm/Teff = 1/(1-H(1-Tc/Tm))`

P.S.: надеюсь у тебя хватит ума не писать формулы??

61. Как определяется адрес следующей исполняемой команды.

Адрес следующей команды лежит в PSW (EIP в 386+). Если все идет ОК, то адрес следующей команды – либо адрес текущей команды + размер команды, либо адрес условного или безусловного перехода. Если же происходит прерывание, то адрес следующей команды берется из таблицы векторов прерываний. Когда обработка прерывания закончена – адрес следующей команды берется из сохраненного PSW.

62. Какая программа выполняет запись информации в область сохранения и какая информация туда записывается.

Область сохранения - область в которую записывается информация о прерванном процессе, и откуда система берет данные при восстановлении процесса. Размер области сохранения зависит от того количества данных о процессе, которые необходимы системе для возобновления процесса.

Запись в область сохранения осуществляет планировщик (вопрос 63). При решении о переходе в другой процесс/поток, планировщик выполняет переключение контекста, т. е. сохраняет текущее состояние процесса/потока в область сохранения и считывает состояние процесса/потока, в который осуществляется переход.

Аппаратно предусмотренной областью сохранения в системах для проц. 386+ выступает TSS (Task state register, смотри вопрос 80), хотя можно программно реализовать любую другую. Большим недостатком TSS является, например, отсутствие записи регистров сопроцессора.

63. Назначение программы - главный планировщик. Смотри вопр. № 242
Как только система определяет, что у нее есть ресурсы (наиболее важно наличие ОП), операционная система загружает планировщик как транзитную программу, которая анализирует наличие остальных ресурсов, инициализируя очередь процессов к процессору. Модуль "планировщик" распределяет между процессами время центрального процессора. Он планирует очередность выполнения процессов до тех пор, пока они добровольно не освободят центральный процессор, дождавшись выделения ресурса, или до тех пор, пока ядро системы не выгрузит их после того, как их время выполнения превысит заранее определенный квант времени. Планировщик выбирает на выполнение готовый к запуску процесс с наивысшим приоритетом; выполнение предыдущего процесса (приостановленного) будет продолжено тогда, когда его приоритет будет наивысшим среди приоритетов всех готовых к запуску процессов.
Другая версия: *ПВ* (высокого уровня)предназначен для *предварительного планирования* входного потока заявок, которые поступают в систему и претендуют на захват ресурсов вычислительной системы. Его иногда называют планировщиком доступа, т.к. он определяет, каким заданиям можно предоставить доступ к системе. Из всех этих заданий ПВ создает очередь О1 (рис. 4.17.) по некоторым критериям: относительные приоритеты, сроки запуска и завершения, время выполнения, интенсивность ввода/вывода данных, потребность памяти.

64. К какому приоритетному уровню относится страничное прерывание.

Страничное прерывание – внутреннее прерывание процессора

65. Недостатки распределения памяти MVT.
Достоинства:
1. Ликвидация фрагментации(В отличии от MFT не выделяет лишнего места)

Недостатки:
1. Ограничение размером физической памяти
2. Затраты на перекомпоновку

66. Каким образом можно изменить приоритет прерываний в контроллере прерываний IBM PC

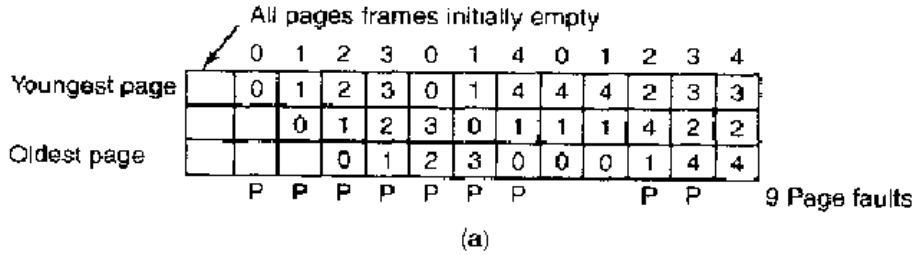
Контроллер прерываний можно запрограммировать для работы в нескольких режимах:

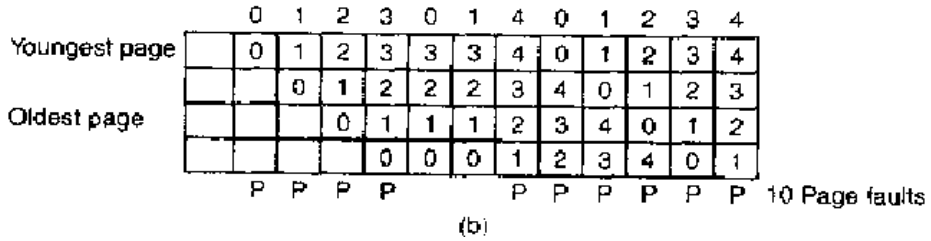
1. Режим фиксированных приоритетов *(Fixed Priority, Fully Nested Mode).* В этом режиме запросы прерываний имеют жесткие приоритеты от 0 (высший) до 7 (низший) и обрабатываются в соответствии с этими приоритетами. Если запрос с меньшим приоритетом возникает во время обработки запроса с более высоким приоритетом, то он игнорируется. BIOS при включении питания и при перезагрузке программирует контроллер прерываний на работу именно в этом режиме. В дальнейшем, режим при необходимости, можно изменить программным способом.
2. Режим с автоматическим сдвигом приоритетов *(Automatic Rotation).* В этом режиме после обработки прерывания данному уровню присваивается минимальный приоритет, а все остальные приоритеты изменяются циклически таким образом, что запросы следующего за только что обработанным уровнем имеют наивысший, а предыдущего — наинизший приоритет. Например, после обработки запроса уровня 6 приоритеты устанавливаются следующим образом: уровень 7 имеет приоритет 0, уровень 8 —

приоритет 1 и т.д., уровень 5 — приоритет 6 и уровень 6 — приоритет 7. Таким образом, в этом режиме всем запросам дается возможность получить обработку.

3. Режим с программно-управляемым сдвигом приоритетов *(Specific Rotation).* Приоритеты можно изменять так же, как и в режиме автоматического сдвига, однако для этого необходимо подать специальную команду, в которой указывается номер уровня, которому нужно присвоить максимальный приоритет.
4. Режим автоматического завершения обработки прерывания *(Automatic End of Interrupt).* В этом режиме, в отличие от остальных, обработчик прерывания *не должен* при своем завершении посылать контроллеру специальный **сигнал завершения обработки аппаратного прерывания** *(End Of Interrupt, EOI),* который разрешает обработку прерываний с текущим и более низкими приоритетами. В данном режиме эти прерывания разрешаются в момент начала обработки прерывания. Режим используется редко, т.к. все обработчики прерываний должны быть **повторно входимыми (реентерабельными)** в случае, если до завершения обработки возникнет запрос на прерывание от того же источника.
5. Режим специальной маски *(Special Mask Mode).* В данном режиме можно замаскировать отдельные уровни прерываний, изменив приоритетное упорядочение обработки запросов. После отмены этого режима восстанавливается прежний порядок обработки прерываний.
6. Режим опроса *(Polling Mode).* В этом режиме обработка прерываний не производится автоматически, сигналы запроса прерываний лишь фиксируются во внутренних регистрах контроллера. Процедуру обработки прерывания необходимо вызывать "вручную", в соответствии с хранящимся в контроллере номером уровня с максимальным приоритетом, по которому имеется запрос.

67. В чем заключается аномалия алгоритма FIFO при смене страниц – аномалия Билэди





Больше page faults при 4 страницах чем при 3!

На первый взгляд кажется очевидным, что с увеличением количества страничных кадров, выделяемых процессу, этот процесс будет выполняться с меньшим количеством прерываний по отсутствию нужной страницы в памяти.

Однако, как установили Билейди, Нельсон и Шедлер (Be69b), в стратегии FIFO определенные последовательности обращений к страницам приводят в действительности к увеличению количества прерываний по отсутствию страницы при увеличении количества страничных кадров, выделяемых процессу. Это явление носит название «аномалии FIFO».

Первая таблица рисунка показывает, как для этой последовательности обращений страницы будут вталкиваться и выталкиваться при реализации стратегии FIFO и выделении данному процессу трех страничных кадров. Вторая таблица показывает, каким образом этот процесс будет работать при тех же самых обстоятельствах, при выделении ему четырех страничных кадров. Слева от каждой таблицы мы отмечаем, будет ли соответствующее новое обращение к странице вызывать прерывание по отсутствию страницы или нет. В действительности оказывается, что при выполнении процесса с четырьмя страничными кадрами количество прерываний по отсутствию страницы на одно больше, чем в случае трех страничных кадров, — факт, явно противоречащий интуиции.

Аномалию FIFO следует, по-видимому, считать скорее курьезом, чем фактором, требующим серьезного к себе отношения. Быть может, истинное значение этого явления для студента, изучающего операционные системы, состоит в том, что оно служит дополнительным указанием на исключительную сложность операционных систем как объектов, где интуитивный подход иногда неприемлем.

68. Дать характеристику и сравнить применение семафоров и мониторов при синхронизации процессов

1. **Семафор** — некоторая (защищенная) переменная, значение которой можно считывать и изменять только при помощи специальных операций P и V. Преимущество по сравнению с test_and_set — разделение действий на отдельные.

Операция **P(S)**: проверяет значение семафора S; если S>0, то S:=S-1, иначе (S=0) ждать (по S) .

Операция **V(S)**: проверяет, есть ли процессы, приостановленные по данному семафору; если есть, то разрешить одному из них продолжить свое выполнение (войти в КУ); если нет, то S:=S+1.

Операция P должна стоять до входа в КУ , а операция V — после выхода из КУ.

Недостатки:

1) громоздкость — в каждом процессе каждый КУ должен окаймляться операциями P и V;
2) невозможность решения целого ряда задач с помощью таких примитивов.

4. **Монитор** — примитив высокого уровня. Здесь "забор" ставится вокруг ОР, что удобнее (чем семафоры), т.к. ресурсов в несколько раз меньше, чем процессов (их КУ) и ставить "заборы" вокруг КУ слишком громоздко. Можно сказать, что монитор — это некоторый программный модуль, в котором объединены ОР и подпрограммы для работы с ними. При обращении к ОР, т.е. соответствующей процедуре монитора для работы с ОР, осуществляется вход в монитор. В каждый конкретный момент времени может выполняться только одна процедура монитора — это и есть решение задачи взаимного исключения. Если процесс обратился к процедуре монитора, а другой процесс уже находится внутри монитора, то обратившийся процесс блокируется и переводится во внешнюю очередь процессов — блокированных по входу в монитор. Процесс, вошедший в монитор (т.е. выполняющий его процедуру), также может быть блокирован им, если не выполняется некоторое *условие* X для выполнения процесса. Условие X в мониторе — это некоторая переменная типа condition. С ней связывается некоторая внутренняя очередь процессов, блокированных по условию X. Внутренняя очередь приоритетнее внешней. Для блокирования процесса по переменной X и помещения его во внутреннюю очередь по этой переменной используется операция монитора WAIT(X). При выполнении операции SIGNAL(X) из очереди, связанной с переменной X, извлекается и разблокируется первый процесс. Если внутренняя очередь пуста, то в монитор разрешается войти первому процессу из внешней очереди.

69. Дать определение распределенная операционная система.

Распределенная система обработки данных (РСОД) или распределенная вычислительная система (РВС) — это среда, в которой компоненты системы или ресурсы: процессоры, память, принтеры, графические станции, программы, данные и т.д., связанны вместе посредством сети, которая позволяет пользователям представлять ВС как единую вычислительную среду и иметь доступ к ее ресурсам [52]. Распределенная ВС имеет некоторые особенности, характерные только для нее:

* О*граниченность*. (failure). Она связана с тем, что количество узлов в сети ограничено и они являются независимыми компонентами сети.
* *Идентификация.* (naming). Каждый из ресурсов сети должен однозначно идентифицироваться.
* *Распределенное управление* (distributing control). Каждый из узлов должен иметь программно-аппаратные возможности выполнения функций управления сетью. Однако повышенные требования к надежности распределенной системы вызывают необходимость избегать применения алгоритмов управления, выделяющих привилегированные по сравнению с другими узлы в сети.
* *Гетерогенность* (heterogeneity). Узлы сети могут быть разнородны, с различной длиной слов и байтовой организацией. Этот аспект может касаться и программного обеспечения, как прикладного так и системного, применяемого для каждого из узлов.

При написании прикладных программ пользователям не требуется знать особенности аппаратного обеспечения сети. Пользователь распределенной вычислительной системы не обязательно должен иметь представление о деталях системы, с которой он работает. Все необходимые пользователю действия при взаимодействии с системными ресурсами сети обеспечиваются для него посредством операционной системы.

Рассмотрим более подробно тот раздел программного обеспечения, который, собственно, и решает задачи, связанные с обеспечением сокрытия вышеупомянутых особенностей системы и облегчением работы пользователя в распределенной вычислительной среде, а также обеспечением эффективного функционирования РСОД. В дальнейшем для определения этого системного программного обеспечения используется термин "распределенная операционная система" (РОС) (distributed operating system).

70. Задачи решаемые при обходе тупиков.

Алгоритм "банкира".

При использовании алгоритма "банкира" подразумевается, что :

1) системе заранее известно количество имеющихся ресурсов;
2) система знает, сколько ресурсов может максимально потребоваться процессу;
3) число пользователей (процессов), обращающихся к ресурсам, известно и постоянно;
4) система знает, сколько ресурсов занято в данный момент времени этими процессами (в общем и каждым из них);
5) процесс может потребовать ресурсов не больше, чем имеется в системе.

В алгоритме "банкира" введено понятие надежного состояния. Текущее состояние вычислительной машины называется **надежным**, если ОС может обеспечить всем текущим пользователям (процессам) завершение их заданий в течение конечного времени. Иначе состояние считается **ненадежным**. Надежное состояние — это состояние, при котором общая ситуация с ресурсами такова, что все пользователи имеют возможность со временем завершить свою работу. Ненадежное состояние может со временем привести к тупику.

Система удовлетворяет только те запросы, при которых ее состояние остается надежным, т.е. при запросе ресурса система определяет сможет ли она завершить хотя бы один процесс, после этого выделения.

Пример решения задачи выделения ресурсов по алгоритму "банкира".

Имеем 6 ресурсов и 6 процессов (рис .2.17.). В таблице показано состояние занятости ресурсов на данный момент (1 свободный ресурс).

Процесс	Выделенное число ресурсов	Требуемое число ресурсов
A	2	3
B	1	3
C	0	2
D	1	2
E	1	4
F	0	5

Рис. 2.17

Эта таблица отражает надежное состояние, т.к. существует такая последовательность выделений — освобождений ресурсов, при которой все процессы за конечное время будут завершены.

Недостатки алгоритма банкира:

1) Если количество ресурсов большое, то становится достаточно сложно вычислить надежное состояние.
2) Достаточно сложно спланировать, какое количество ресурсов может понадобиться системе.
3) Число работающих пользователей (процессов) остается постоянным.
4) Пользователи должны заранее указывать максимальную потребность в ресурсах, однако, потребность может определяться динамически по мере выполнения процесса.
5) Пользователь всегда заказывает ресурсов больше, чем ему может потребоваться.
6) Алгоритм требует, чтобы процессы возвращали выделенные им ресурсы в течение некоторого конечного времени. Однако, для систем реального времени требуются более конкретные гарантии. Т.е. в системе должно быть задано максимальное время захвата всех ресурсов процессом (через это время ресурсы должны быть освобождены).

Графический метод обхода тупика (рис. 2.18).

Имеются 2 процесса P1 и P2 и каждому из них для выполнения требуются по 2 ресурса: Д — диск , П — процессор. Если P1 захватил процессор , а P2 — диск, то возникает тупиковая ситуация. Однако, если P1 захватил П и Д и завершил свое выполнение, то P2 захватывает эти же ресурсы и заканчивает выполнение — беступиковое взаимодействие.

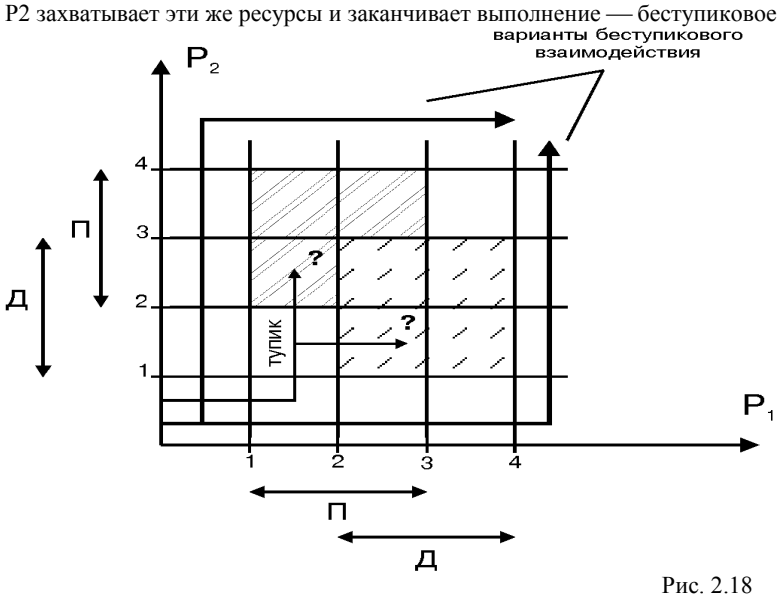


Рис. 2.18

71. Способы хранения места расположения файлов на внешнем носителе

- это учет соответствия блоков(кластеров) диска файлам. Основ. Методы:
  - непрерывнве файлы(CD-ROM), Преимущества-1легко реализовать 2 высок. производительность. Недостатки – 1 фрагментация
  - связные списки Недост -1 сильные потери производит. при произвольном доступе 2 нечетность длинны блока
  - связные списки при пом. таблиц в памяти (FAT) – устраняет недост. пред. метода , но для этого вся таблица должна быть в памяти(недост)
  - i-node (*nix - системы)

72. Способы повышения эффективности работы файловых систем

Производительность ФС:

-кєширование(использовать буфера в ОП) –для бістрого поиска в кєше используется хєш таблица
-опережающее чтение блока
-снижение времени перемещения блока головок
-файловая система с журнальной структурой LFS (очень мутная)

73. **Дать определение. Параллельная обработка.**

Под режимом *параллельной обработки* данных нескольких задач понимается такой многопрограммный режим, в котором переход от одной задачи к другой происходит через достаточно короткие промежутки времени (кванты), сравнимые с тактом машины, чтобы создать у пользователя впечатление одновременности исполнения нескольких программ (режим кажущегося совмещения).

Основной целью данного режима является улучшение обслуживания пользователей, выражающееся в том, что у последних создается впечатление отсутствия очереди, так как решение их задач не прерывается на длительные отрезки времени. Кроме того, если пользователю предоставляются некоторые средства прямого доступа (хотя бы для вывода информации), то это впечатление еще более усиливается выдачей результатов по мере их получения. Параллельная обработка основана на значительном отличии времени реакции пользователя и машины вследствии высокой скорости работы последней.

74. Смысл алгоритма MESI.(см 291, сам я не нашел)

75. Связь модулей по управлению. Какие операции выполняются и какими программами.

76. Чем определяется размер таблицы страниц при виртуальной организации памяти.

От количества записей – по записи на виртуальную страницу . В зависимости от размера памяти и рамера страницы получим их количество.

Но есть еще инвертированные таблицы страниц – в них по записи на один блок реальной памяти(ОП) а не виртуальной.

77. Какая программа создает PCB?

Выполнение функций ОС, связанных с управлением процессами, осуществляется с помощью **блока управления процессом** (PCB). **Вход в процесс** (фиксация системой процесса) — это создание его блока управления (PCB), а **выход из процесса** — это его уничтожение, т. е. уничтожение его блока управления.

Таким образом для каждого активизированного процесса система создает PCB, в котором в сжатом виде содержится информация о процессе, используемая при управлении. PCB — это системная структура данных, содержащая определенные сведения о процессе и имеющая следующие поля:

78. Применение сигналов при синхронизации процессов

Для каждого сигнала в системе предусмотрена обработка по умолчанию, если процесс не указал другого действия. Возможны следующие действия при принятии сигнала:

- Завершить выполнение процесса
- Игнорировать сигнал
- Остановить процесс
- Продолжить

Процесс может установить свой собственный обработчик на обработку своего сигнала. Процесс может задержать или заблокировать обработку сигнала, однако система защищает себя и иногда некоторые сигналы нельзя заблокировать.

Любая обработка сигнала подразумевает, что процесс активен.

Существует опасность задержки сигнала между отправкой и доставкой при большой загрузке ВС. Сигнал может быть доставлен только в том случае, если он выбран планировщиком.

Доставка сигнала произойдет, когда ядро от имени процесса вызовет специальную системную функцию, которая проверит, существуют ли сигналы, адресованные процессу. Эти функции вызываются в следующих случаях:

- Непосредственно перед возвратом процесса из режима ядра в режим задачи при обработке системного вызова
- Перед переходом процесса в состояние сна, если он имеет приоритет, допускающий прерывание сигнала
- Сразу же после пробуждения с приоритетом, допускающим прерывание сигнала
- Если процедура обнаруживает ожидающий доставку сигнал, ядро вызывает функцию доставки сигнала, выполняющую действие по умолчанию либо функцию обработки сигнала. Функция возвращает процесс в режим задачи, переходит на обработку и возвращает контекст.

При переходе процесса в состояние сна определены 2 категории событий, связанных с наличием сигнала:

- При наличии сигнала допускается прерывание этого сигнала
  Доставка сигнала будет проверена ядром непосредственно перед переходом в сон. Если генерация во время сна, то ядро разбудит его и прерванный системный вызов будет завершен с ошибкой
- Не допускается

79. Недостаток распределения памяти перемещаемыми разделами

- необходимость отслеживания перемещения страниц (сложно)
- фрагментация

80. Структура дескриптора TSS.

Задача и сегмент состояния задачи

До сих пор мы говорили о параллельной работе программ. На самом деле каждая отдельная программа может состоять из нескольких частей, работающих параллельно. Например, текстовый процессор может содержать программные модули, которые параллельно с редактированием текста выполняют нумерацию страниц, печать текста или автоматическое сохранение его на диске. Каждую такую часть программы мы будем называть задачей.

Исходя из этой терминологии в мультизадачной среде одновременно выполняется много задач, принадлежащих разным программам. Причём количество задач больше или равно количеству выполняющихся программ.

Как правило, квантование времени процессора выполняется на уровне задач, а не на уровне программ. По прерыванию таймера процессор переключается от одной задачи к другой, и таким образом осуществляется параллельное выполнение программ.

Для хранения контекста неактивной в настоящей момент задачи процессор i80286 использует специальную область памяти, называемую сегментом состояния задачи TSS (Task State Segment). Формат TSS представлен на рис. 14.

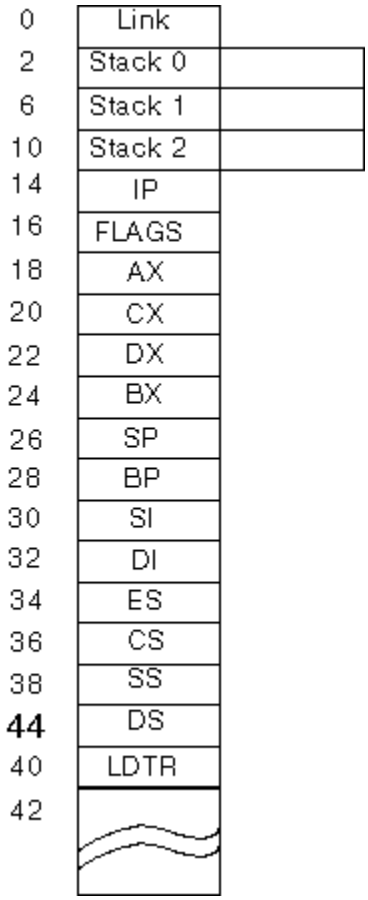


Рис. 14. Формат сегмента состояния задачи TSS.

Сегмент TSS адресуется процессором при помощи 16-битного регистра TR (Task Register), содержащего селектор дескриптора TSS, находящегося в глобальной таблице дескрипторов GDT (рис. 15).

Рис. 15. Дескриптор сегмента состояния задачи TSS.

Поле доступа содержит бит B - бит занятости. Если задача активна, этот бит устанавливается процессором в 1.

Операционная система для каждой задачи создаёт свой TSS. Перед тем как переключиться на выполнение новой задачи, процессор сохраняет контекст старой задачи в её сегменте TSS.

Что же конкретно записывается в TSS при переключении задачи?

Записывается содержимое регистров общего назначения AX, BX, CX, DX, регистров SP, BP, SI, DI, сегментных регистров ES, CS, SS, DS, содержимое указателя команд IP и регистра флажков FLAGS. Кроме того, сохраняется содержимое регистра LDTR, определяющего локальное адресное пространство задачи.

Дополнительно при переключении задачи в область TSS со смещением 44 операционная система может записать любую информацию, которая относится к данной задаче. Эта область процессором не считывается и никак не модифицируется.

Поле Link представляет собой поле обратной связи и используется для организации вложенных вызовов задач. Это поле мы рассмотрим в следующем разделе.

Поля Stack 0, Stack 1, Stack 2 хранят логические адреса (селектор:смещение) отдельных для каждого кольца защиты стеков. Эти поля используются при межсегментных вызовах через вентили вызова.

Для обеспечения защиты данных процессор назначает отдельные стеки для каждого кольца защиты. Когда задача вызывает подпрограмму из другого кольца через вентиль вызова, процессор вначале загружает указатель стека SS:SP адресом нового стека, взятого из соответствующего поля TSS.

Затем в новый стек копируется содержимое регистров SS:SP задачи (т.е. адрес вершины старого стека задачи). После этого в новый стек копируются параметры, количество которых задано в вентиле вызова и адрес возврата.

Таким образом, при вызове привилегированного модуля через вентиль вызова менее привилегированная программа не может передать в стеке больше параметров, чем это определено операционной системой для данного модуля.

Включение адресов стеков в TSS позволяет разделить стеки задач и обеспечивает их автоматическое переключение при переключении задач.

81. Назначение - Система управления данными

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.

82. Как обрабатывается прерывание по обращению к супервизору.

Прерывания по обращению к супервизору. Вызываются при выполнении процессором **команды обращения к супервизору** (вызов функции операционной системы). Обычно такая команда инициируется выполняемым процессом при необходимости получения дополнительных ресурсов либо при взаимодействии с устройствами ввода/вывода.

83. Чем ограничен размер виртуальной памяти.

ОП + область подкачки .

84. Назначение - Система управления внешними устройствами

Одной из главных функций ОС является управление всеми устройствами ввода-вывода компьютера. ОС должна передавать устройствам команды, перехватывать прерывания и обрабатывать ошибки; она также должна обеспечивать интерфейс между устройствами и остальной частью системы. В целях развития интерфейс должен быть одинаковым для всех типов устройств (независимость от устройств).

85. Сколько входных очередей заданий формирует система ?

Процессы при поступлении в систему находятся в подготовленном состоянии и накапливаются во входных очередях заданий. Следует различать процессы, поступающие в систему и требующие безусловного выполнения в соответствии с одной из дисциплин обслуживания, и процессы, находящиеся в подготовленном состоянии после загрузки операционной системы. Процессы первого вида, как правило, принадлежат пользователям, а второго вида — ОС. Такие процессы находятся в подготовленном состоянии до тех пор, пока для выполнения функций системы они не будут активизированы. Пользовательские процессы активизируются или делается

попытка их активизации при наличии в системе ресурсов. Попытка активизации процесса производится системой при наличии в системе оперативной памяти, достаточной для размещении программы, подчиненной процессу.

Поэтому в системе может находиться одновременно довольно много подготовленных и готовых процессов, для которых организуются очереди (одна для готовых и одна для подготовленных процессов). Дисциплины управления ими могут быть различными и выбор дисциплины обслуживания определяется требованиями, предъявляемыми к системе планирования.

86. В чем причина появления несогласованности файловой системы

Сбой в системе прежде чем кэши будут сброшены на диск. Особенно важно если это служебная инфа(i-node, зап о свободном месте…)

87. Как решается задача согласованности данных.

Прогой(scandisk,fsck). Используется избыточность ФС – копии.

88. Чем определяется количество PCB ?

Таким образом для каждого активизированного процесса система создает PCB, в котором в сжатом виде содержится информация о процессе, используемая при управлении

Блоки управления системных процессов создаются при загрузке системы. Это необходимо, чтобы система выполняла свои функции достаточно быстро, и время реакции ОС было минимальным. Однако, количество блоков управления системными процессами меньше, чем количество самих системных процессов. Это связано с тем, что структура ОС имеет либо оверлейную, либо динамически — последовательную структуру иерархического типа, и нет необходимости создавать для программ, которые никогда не будут находиться одновременно в оперативной памяти, отдельные PCB.

89. Когда процесс переходит из активного состояния в готовое.

Если активный процесс не выполнился за выделенный ему квант времени, то он переходит снова в готовое состояние.

90. Связь модулей по данным. Виды связей.

Существует 2 вида связей между модулями:

-- по данным

-- по управлению

Связь по данным может быть через общие регистры или через адрес списка параметров. Связь по данным – данные через общий регистр или через системный регистр передается адрес списка параметров.

91. Концепция рабочего множества (зоны).

Многие сист отслеж рабочий набор процесса(поскольку процессы характеризуются локальностью обращения) и следят, что бы он был ОП.

92. Функции настраивающего загрузчика. Какая программа готовит для него информацию.

Настройка адресов при загрузке (прибавление констаты). Готовит компоновщик(Linker), указывая что надо настроить.

93. Где находится адрес программы начальной загрузки.

Далее BIOS определяет наличие OS DOS. Если обнаружена системная дискета, то BIOS выполняет прерывание INT 19h для доступа к первому сектору диска, который содержит блок начальной загрузки (Master boot), Считывается 0 дорожка 1 сектор цилиндра 0, определяется активный раздел диска (для Hard Drive), считывается первый сектор активного раздела, в начале которого находится адрес программы начальной загрузки операционной системы, она считывается и ей передается управление.

94. Какие основные типы прерываний используются при выполнении операции ввода-вывода.

Прерывания по вводу/выводу. Инициируются аппаратурой ввода/вывода при изменении состояния каналов ввода/вывода, а также при завершении операций ввода/вывода.

Прерывания по обращению к супервизору. Вызываются при выполнении процессором **команды обращения к супервизору** (вызов функции операционной системы). Обычно такая команда инициируется выполняемым процессом при необходимости получения дополнительных ресурсов либо при взаимодействии с устройствами ввода/вывода.

95. Назначение регистра IDTR.

Таблица прерываний защищённого режима

Обработка прерываний и исключений в защищённом режиме по аналогии с реальным режимом базируется на таблице прерываний. Но таблица прерываний защищённого режима является таблицей дескрипторов, которая содержит так называемые вентили прерываний, вентили исключений и вентили задач.

Таблица прерываний защищённого режима называется дескрипторной таблицей прерываний IDT (Interrupt Descriptor Table). Также как и таблицы GDT и LDT, таблица IDT содержит 8-байтовые дескрипторы. Причём это системные дескрипторы - вентили прерываний, исключений и задач. Поле TYPE вентиля прерывания содержит значение 6, а вентиля исключения - значение 7.

Формат элементов дескрипторной таблицы прерываний IDT показан на рис. 12.

Рис. 12. Формат элементов дескрипторной таблицы прерываний IDT.

Где располагается дескрипторная таблица прерываний IDT?

Её расположение определяется содержимым 5-байтового внутреннего регистра процессора IDTR. Формат регистра IDTR полностью аналогичен формату регистра GDTR, для его загрузки используется команда LIDT. Так же, как регистр GDTR содержит 24-битовый физический адрес таблицы GDT и её предел, так и регистр IDTR содержит 24-битовый физический адрес дескрипторной таблицы прерываний IDT и её предел.

Регистр IDTR обычно загружают перед переходом в защищённый режим. Разумеется, это можно сделать и потом, находясь в защищённом режиме. Однако для этого программа должна работать в привилегированном нулевом кольце.

96. Структура таблицы страниц в виртуальной памяти.

Для каждого процесса ОС создает **таблицу страниц** — информационную структуру, содержащую записи обо всех виртуальных страницах процесса.

Запись таблицы, называемая **дескриптором страницы** и включает следующую информацию:

- номер физической страницы, в которую загружена данная виртуальная страница;
- признак присутствия, устанавливается в единицу, если виртуальная страница находится в оперативной памяти;
- признак модификации страницы, который устанавливается в единицу всякий раз, когда производится запись по адресу, относящемуся к данной странице;
- признак обращения к странице, называемый также битом доступа, который устанавливается в единицу при каждом обращении по адресу, относящемуся к данной странице.

Признаки присутствия, модификации и обращения в большинстве современных процессоров устанавливаются аппаратно при операциях с памятью. Информация из таблиц страниц используется для решения вопроса о необходимости перемещения той или иной страницы между памятью и диском, а также для преобразования виртуального адреса в физический.

Виртуальный адрес состоит из виртуального номера страницы и смещения. Номер записи в таблице страниц соответствует номеру виртуальной страницы. Размер записи колеблется от системы к системе, но чаще всего он составляет 32 бит. Из этой записи в таблице страниц находится номер кадра для данной виртуальной страницы, затем прибавляется смещение и формируется физический адрес. Помимо этого запись в таблице страниц содержит информацию об атрибутах страницы. Это биты присутствия и защиты (например, 0 – read/write, 1 – read only...). Также могут быть указаны: бит модификации, который устанавливается, если содержимое страницы модифицировано, и позволяет контролировать необходимость перезаписи страницы на диск; бит ссылки, который помогает выделить малоиспользуемые страницы; бит, разрешающий кэширование, и другие управляющие биты. Заметим, что адреса страниц на диске не являются частью таблицы страниц.

Подсчитаем примерный размер таблицы страниц. В 32-битном адресном пространстве при размере страницы 4 Кбайт (Intel) получаем $2^{32}/2^{12}=2^{20}$, то есть приблизительно миллион страниц, а в 64-битном и того более. Таким образом, таблица должна иметь примерно миллион строк (entry), причем

запись в строке состоит из нескольких байтов. Заметим, что каждый процесс нуждается в своей таблице страниц.

97. Назначение - Система управления задачами

Смотри вопрос 58

98. Дать пояснение кеш памяти с прямым отображением

Смотри вопрос 45

99. Виды сигналов при синхронизации процессов

100. Назвать способы реализации защиты памяти.

Защиту от влияния одних программ на другие называют защитой памяти. Средства защиты памяти обеспечивают проверку адреса при каждом обращении к памяти. К простейшим способам защиты, нашедшим широкое применение, относятся способы защиты по граничным адресам, основанные на использовании граничных или базово-граничных регистров. В обоих случаях программе выделяется область памяти, которая задается двумя предельными величинами: значением *B* начального адреса области (базовой области) и предельной длиной *L* программы, или значениями S1 и S2 граничных адресов выделенной области. Эти значения заносятся в соответствующие регистры в момент инициирования программы. При каждом обращении к выделенной области оперативной памяти производится сравнение генерируемых программой адресов с установленными пределами. Если рассматриваемый адрес находится за установленными пределами, то возникает прерывание по нарушению защиты памяти и управление передается специальной программе, обрабатывающей нарушение защиты памяти.

Такая система защиты может считаться вполне приемлемой при условии, что программа и данные находятся в единой области памяти. Если же программа и данные будут находиться в различных областях памяти, то в этом случае для реализации защиты памяти потребуется два базово-граничных регистра или две пары граничных регистров. При этом один из базово-граничных регистров или одна из пар граничных регистров будет использоваться для контроля корректности адреса операнда, а второй базово-граничный регистр или вторая пара операндов — для контроля корректности адреса, находящегося в счетчике команд.

Дальнейшая сегментация программ и данных и размещение отдельных сегментов в различных областях оперативной памяти приводит к необходимости использования множества базово-граничных регистров или пар граничных регистров, число которых должно быть равно числу сегментов максимально-сегментированной программы. При этом возникает необходимость выделения номера сегмента из адреса.

При таком распределении с каждой математической страницей отождествляется специальный признак, который указывает на возможность или запрет использования соответствующей страницы при выполнении той или иной программы. Эти признаки хранятся в специальном регистре, который называют регистром защиты математических страниц. Состояние этого регистра в любой момент времени определяет доступность математических страниц. При этом защита памяти на уровне математических страниц существенно упрощается. Однако при записи программ в физических адресах механизм защиты памяти на уровне математических страниц уже не работает. В этой связи наряду с защитой памяти в поле математических адресов используют и защиту физических адресов.

При защите физических адресов в условиях страничного распределения памяти наиболее часто применяется защита памяти с использованием ключей, т. е. так называемая защита по ключам. В соответствии с принципом защиты по ключам каждой странице оперативной памяти присваивается ключ памяти. Все множество ключей хранится в так называемой памяти ключей, которая защищена от доступа рабочих программ. Запись в память ключей может производиться только супервизором. Каждой программе, размещенной в определенной странице оперативной памяти, присваивается соответствующий ключ защиты (ключ программы). Ключ защиты указывается в слове состояния программы (ССП). В процессе выполнения программы из кода адреса выделяется номер страницы, который рассматривается как входной адрес памяти ключей. Выбираемый по данному адресу ключ памяти сравнивается с ключом защиты данной программы,который является составной частью слова состояния программы. При установлении логической равнозначности обращение к оперативной памяти разрешается. В противном случае осуществляется прерывание по защите памяти. Исключением является случай, когда код ключа защиты равен нулю, что определяет свободный доступ к любой странице оперативной памяти. Код ключа защиты супервизора всегда равен нулю. В ЕС ЭВМ при защите физических адресов оперативная память рассматривается разделенной на блоки объемом в 2048 байтов (т. е. каждый блок составляет половину страницы). Память ключей состоит из 8-разрядных записей, каждые из которых включают в себя 4-разрядный ключ защиты, разряд выборки, разряд обращения, разряд изменения и разряд четности.

Если разряд выборки установлен в единичное состояние, то блок памяти с соответствующим ключом памяти защищен как от выборки, так и от записи.

Разряд обращения устанавливается в единицу при любом обращении к данному блоку. Просмотр состояния этих разрядов позволяет определить неиспользуемые страницы с целью их удаления из оперативной памяти. При этом для определения неиспользуемой страницы необходимо просмотреть разряды обращений двух блоков памяти, составляющих данную страницу.

Разряд изменения устанавливается в единицу только тогда, когда в страницу вносятся изменения. Если страница не изменяется, во внешней памяти имеется ее копия и ее следует вывести из оперативной памяти (она является кандидатом на удаление), то нет необходимости выполнять специальную программу для вывода ее во внешнюю память, а соответствующие блоки оперативной памяти можно сразу использовать. Если же страница претерпела изменения, то необходимо переписать данную

страницу во внешнюю память и только после этого использовать освободившийся блок оперативной памяти.

Защита **страничной** памяти основана на контроле уровня доступа к каждой странице, возможны следующие уровни доступа:
1. только чтение
2. и чтение и запись
3. только выполнение

В этом случае каждая страница снабжается трехбитным кодом уровня доступа. При трансформации логического адреса в физический сравнивается значение кода разрешенного уровня доступа с фактически требуемым. При их несовпадении работа программы прерывается.

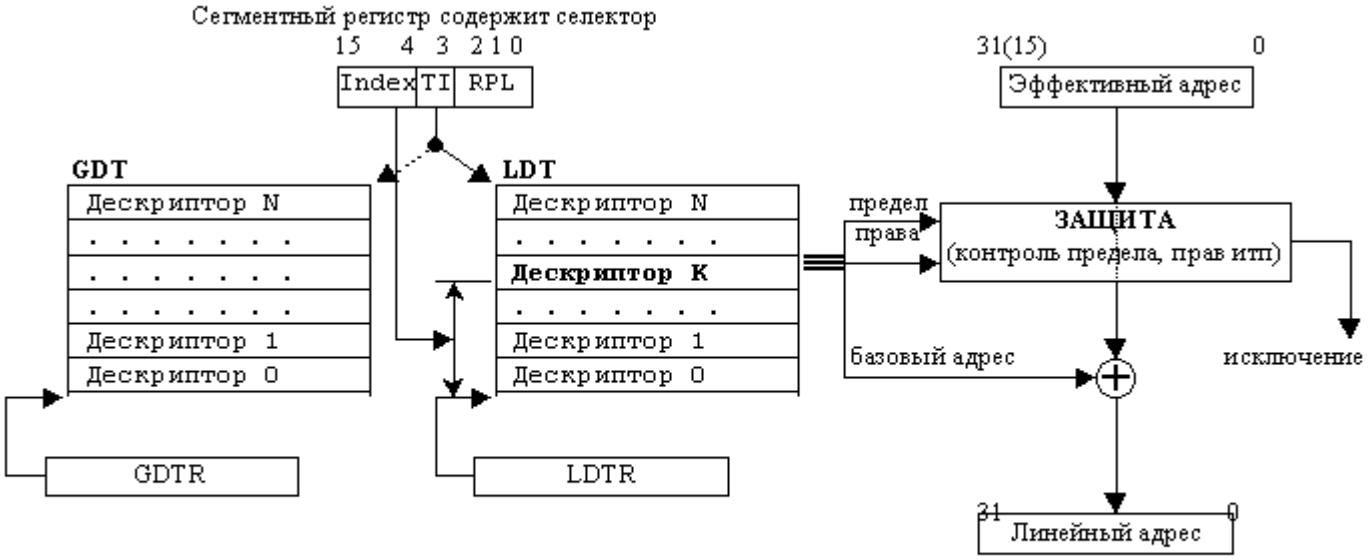
Защита сегментной памяти основана на контроле уровня доступа к каждому сегменту. Например, сегменты, содержащие только код, могут быть помещены как доступные только для чтения или выполнения, а сегменты, содержащие данные, помечают как доступные для чтения и записи.
Каждый сегмент описывается дескриптором сегмента.
ОС строит для каждого исполняемого процесса соответствующую таблицу дескрипторов сегментов и при размещении каждого из сегментов в ОП или внешней памяти в дескрипторе отмечает его текущее местоположение (бит присутствия).
Дескриптор содержит поле адреса, с которого сегмент начинается и поле длины сегмента. Благодаря этому можно осуществлять контроль
1) размещения сегментов без наложения друг на друга
2) обращается ли код исполняющейся задачи за пределы текущего сегмента.

В дескрипторе содержатся также данные о правах доступа к сегменту (запрет на модификацию, можно ли его предоставлять другой задаче), защита.

101. Содержимое каких полей GDT используются при смене страниц

Дескриптор - это 8-байтная единица описательной информации, распознаваемая устройством управления памятью в защищенном режиме, хранящаяся в дескрипторной таблице. Дескриптор сегмента содержит базовый адрес описываемого сегмента, предел сегмента и права доступа к сегменту. Дескрипторы являются основой защиты и мультизадачности. В защищенном режиме сегменты могут начинаться с любого линейного адреса и иметь любой предел вплоть до 4Гбайт. Существуют две обязательных дескрипторных таблицы - глобальная (GDT) и дескрипторная таблица прерывания (IDT),- а также множество (до 8192) локальных дескрипторных таблиц (LDT), из которых в один момент времени процессору доступна только одна. Дескрипторы сегментов могут находиться в GDT или LDT. Расположение дескрипторных таблиц определяется регистрами процессора GDTR, IDTR, LDTR. Регистры GDTR и IDTR - 6-байтные, они содержат 32 бита линейного базового адреса дескрипторной таблицы и 16 бит предела таблицы. Программно доступная часть регистра LDTR - 16 бит, которые являются селектором LDT. Дескрипторы LDT находятся в GDT. Однако чтобы не обращаться каждый раз к GDT в процессоре имеется теневая (программно недоступная) часть регистра LDTR, в которую процессор помещает дескриптор LDT при каждой перегрузке селектора в регистре LDTR.

Значения, помещаемые в сегментные регистры, называются селекторами. Селектор содержит индекс дескриптора в дескрипторной таблице, бит определяющий, к какой дескрипторной таблице производится обращение (LDT или GDT), а также запрашиваемые права доступа к сегменту. Таким образом, селектор выбирает дескрипторную таблицу, выбирает дескриптор из таблицы, а по дескриптору определяется положение сегмента в линейном пространстве памяти. Однако обращение к дескрипторным таблицам происходит только при загрузке селектора в сегментный регистр. При этом процессор помещает дескриптор в теневую (программно недоступную) часть сегментного регистра. При формировании линейного адреса дескриптор сегмента процессору уже известен.



TI (Table Indicator) – выбирает дескрипторную таблицу (TI=0 : GDT; TI=1 : LDT)
RPL (Requested Privilege Level) – запрашиваемый уровень привилегий

102. В чем особенность решения задач планирования для кластерных ВС.

103. Смысл реализации кеш памяти с обратной записью.

В процессе работы содержимое кэш-памяти постоянно обновляется. Вытеснение данных означает либо просто объявление свободной некоторой области кэш-памяти (сброс бита действительности) менялись либо в дополнение к этому копирование данных в основную память, если они модифицировались.

Наличие в компьютере двух копий данных – в основной памяти и в кэше – порождает проблему согласования данных.

Существует два подхода к решению этой проблемы:

- сквозная запись (write through). При запросе к основной памяти (в том числе при записи) просматривается кэш. Если данные по запрашиваемому адресу отсутствуют, запись выполняется только в основную память. Если данные находятся в кэше, запись делается и в кэш ив память.
- обратная запись (write back). Выполняется просмотр кэша, если данных там нет, то запись делается в основную память. В противном случае запись делается только в кэш. При этом устанавливается признак модификации. При вытеснении данных из кэша эти данные будут переписаны в основную память.

104. Условие перехода процесса из активного состояния в заблокированное.

Процесс, которому выделяется время процессора, переводиться в **активное** состояние или состояние выполнения.

Во время выполнения процесса ему могут понадобиться дополнительные ресурсы, но для их получения необходимо некоторое время. Т.е. процесс должен ожидать наступления некоторого события или окончания выполнения конкретной операции (например, ввода/вывода для получения/выдачи данных). Такой процесс переводится в **заблокированное** состояние.

105. Доставка и обработка сигналов при синхронизации процессов

Для каждого сигнала в системе предусмотрена обработка по умолчанию, если процесс не указал другого действия. Возможны следующие действия при принятии сигнала:

- Завершить выполнение процесса
- Игнорировать сигнал
- Остановить процесс
- Продолжить

Процесс может установить свой собственный обработчик на обработку своего сигнала. Процесс может задержать или заблокировать обработку сигнала, однако система защищает себя и иногда некоторые сигналы нельзя заблокировать.

Любая обработка сигнала подразумевает, что процесс активен.

Существует опасность задержки сигнала между отправкой и доставкой при большой загрузке ВС. Сигнал может быть доставлен только в том случае, если он выбран планировщиком.

Доставка сигнала произойдет, когда ядро от имени процесса вызовет специальную системную функцию, которая проверит, существуют ли сигналы, адресованные процессу. Эти функции вызываются в следующих случаях:

- Непосредственно перед возвратом процесса из режима ядра в режим задачи при обработке системного вызова
- Перед переходом процесса в состояние сна, если он имеет приоритет, допускающий прерывание сигнала
- Сразу же после пробуждения с приоритетом, допускающим прерывание сигнала
- Если процедура обнаруживает ожидающий доставку сигнал, ядро вызывает функцию доставки сигнала, выполняющую действие по умолчанию либо функцию обработки сигнала. Функция возвращает процесс в режим задачи, переходит на обработку и возвращает контекст.

При переходе процесса в состояние сна определены 2 категории событий, связанных с наличием сигнала:

- При наличии сигнала допускается прерывание этого сигнала
  Доставка сигнала будет проверена ядром непосредственно перед переходом в сон. Если генерация во время сна, то ядро разбудит его и прерванный системный вызов будет завершен с ошибкой
- Не допускается

106. Характеристика распределенной операционной системы.

Распределенная система обработки данных (РСОД) или распределенная вычислительная система (РВС) — это среда, в которой компоненты системы или ресурсы: процессоры, память, принтеры, графические станции, программы, данные и т.д., связанны вместе посредством сети, которая позволяет пользователям представлять ВС как единую вычислительную среду и иметь доступ к ее ресурсам. Распределенная ВС имеет некоторые особенности, характерные только для нее:

* О*граниченность*. (failure). Она связана с тем, что количество узлов в сети ограничено и они являются независимыми компонентами сети.
* *Идентификация.* (naming). Каждый из ресурсов сети должен однозначно идентифицироваться.
* *Распределенное управление* (distributing control). Каждый из узлов должен иметь программно-аппаратные возможности выполнения функций управления сетью. Однако повышенные требования к надежности распределенной системы вызывают необходимость избегать применения алгоритмов управления, выделяющих привилегированные по сравнению с другими узлы в сети.
* *Гетерогенность* (heterogeneity). Узлы сети могут быть разнородны, с различной длиной слов и байтовой организацией. Этот аспект может касаться и программного обеспечения, как прикладного так и системного, применяемого для каждого из узлов.

При написании прикладных программ пользователям не требуется знать особенности аппаратного обеспечения сети. Пользователь распределенной вычислительной системы не обязательно должен иметь представление о деталях системы, с которой он работает. Все необходимые пользователю действия при взаимодействии с системными ресурсами сети обеспечиваются для него посредством операционной системы.

107. Тот раздел системного программного обеспечения, который, собственно, и решает задачи, связанные с обеспечением сокрытия вышеупомянутых особенностей системы и облегчением работы пользователя в распределенной вычислительной среде, а также обеспечением эффективного функционирования РСОД и называется "распределенная операционная система" (РОС) (distributed operating system).

Принципы построения распределенных ОС (прозрачность, гибкость, надежность, эффективность, масштабируемость)

(1) Прозрачность (для пользователя и программы).

Прозрачность расположения	Пользователь не должен знать, где расположены ресурсы
Прозрачность миграции	Ресурсы могут перемещаться без изменения их имен
Прозрачность размножения	Пользователь не должен знать, сколько копий существует
Прозрачность конкуренции	Множество пользователей разделяет ресурсы автоматически
Прозрачность параллелизма	Работа может выполняться параллельно без участия пользователя

(2) Гибкость (не все еще ясно - потребуется менять решения).

Использование монолитного ядра ОС или микроядра.

(3) Надежность.

- Доступность, устойчивость к ошибкам (fault tolerance).
- Секретность.

(4) Производительность.
Грануллированность. Мелкозернистый и крупнозернистый параллелизм (fine-grained parallelism, coarse-grained parallelism). Устойчивость к ошибкам требует дополнительных накладных расходов.
(5) Масштабируемость.
Плохие решения:

- централизованные компоненты (один почтовый-сервер);
- централизованные таблицы (один телефонный справочник);
- централизованные алгоритмы (маршрутизатор на основе полной информации).

Только децентрализованные алгоритмы со следующими чертами:

- ни одна машина не имеет полной информации о состоянии системы;
- машины принимают решения на основе только локальной информации;
- выход из строя одной машины не должен приводить к отказу алгоритма;
- не должно быть неявного предположения о существовании глобальных часов.

108. Назвать состояния строки в кеш памяти.
В современных микропроцессорах, используемых для построения мультипроцессорных систем, идентичность данных в кэшах ВМ (когерентность кэшей) поддерживается с помощью межмодульных пересылок. Существует несколько способов реализации когерентности, применяемых в зависимости от типа используемой коммуникационной среды и сосредоточенности или физической распределенности памяти между процессорными модулями.

1. Сосредоточенная память
Рассмотрим реализацию одного из алгоритмов поддержки когерентности кэшей, известного как MESI (Modified, Exclusive, Shared, Invalid).
Каждая строка кэш-памяти ВМ может находиться в одном из следующих состояний:
M - строка модифицирована (доступна по чтению и записи только в этом ВМ, потому что модифицирована командой записи по сравнению со строкой основной памяти);
E - строка монопольно копированная (доступна по чтению и записи в этом ВМ и в основной памяти);
S - строка множественно копированная или разделяемая (доступна по чтению и записи в этом ВМ, в основной памяти и в кэш-памятях других ВМ, в которых содержится ее копия);
I - строка, невозможная к использованию (строка не доступна ни по чтению, ни по записи).
Состояние строки используется, во-первых, для определения процессором ВМ возможности локального, без выхода на шину, доступа к данным в кэш-памяти, а, во-вторых, - для управления механизмом когерентности.
Для управления режимом работы механизма поддержки когерентности используется бит WT, состояние 1 которого задает режим сквозной (write-through) записи, а состояние 0 - режим обратной (write-back) записи в кэш-память.
Промах чтения в кэш-памяти заставляет вызвать строку из основной памяти и сопоставить ей состояние E или S. Кэш-память заполняется только при промахах чтения. При промахе записи транзакция записи помещается в буфер и посылается в основную память при предоставлении шины.
Для поддержки когерентности строк кэш-памяти при операциях ввода/вывода и обращениях в основную память других процессоров на шине генерируются специальные циклы опроса состояния кэш-памятей. Эти циклы опрашивают кэш-памяти на предмет хранения в них строки, которой принадлежит адрес, используемый в операции, инициировавшей циклы опроса состояния. Возможен режим принудительного перевода строки в состояние I, который задается сигналом INV.

2. Физически распределенная память
На практике применяют сложные алгоритмы, например, DASH, который заключается следующем. Каждый модуль памяти имеет для каждой строки, резидентной в модуле, список модулей, в кэшах которых размещены копии строк.
С каждой строкой в резидентном для нее модуле связаны три ее возможных глобальных состояния:
1) "некэшированная", если копия строки не находится в кэше какого-либо другого модуля, кроме, возможно, резидентного для этой строки;
2) "удаленно-разделенная", если копии строки размещены в кэшах других модулей;
3) "удаленно-измененная", если строка изменена операцией записи
в каком-либо модуле.
Кроме этого, каждая строка кэша находится в одном из трех локальных состояний:
1) "невозможная к использованию";
2) "разделяемая", если.есть неизмененная копия, которая, возможно, размещается также в других кэшах;
3) "измененная", если копия изменена операцией записи. Каждый процессор может читать из своего кэша, если состояние читаемой строки "разделяемая" или "измененная". Если строка отсутствует в кэше или находится в состоянии "невозможная к использованию", то посылается запрос "промах чтения", который направляется в модуль, резидентный для требуемой строки.

Если глобальное состояние строки в резидентном модуле "некэшированная" или "удаленно-разделенная", то копия строки посылается в запросивший модуль и в список модулей, содержащих копии рассматриваемой строки, вносится модуль, запросивший копию.
Если состояние строки "удаленно-измененная", то запрос "промах чтения" перенаправляется в модуль, содержащий измененную строку. Этот модуль пересылает требуемую строку в запросивший модуль и в модуль, резидентный для этой строки, и устанавливает в резидентном модуле для этой строки состояние "удаленно-распределенная".
Если процессор выполняет операцию записи и состояние строки, в которую производится запись "измененная", то запись выполняется и вычисления продолжаются. Если состояние строки "невозможная к использованию" или "разделяемая", то модуль посылает в резидентный для строки модуль запрос на захват в исключительное использование этой строки и приостанавливает выполнение записи до получения подтверждений, что все остальные модули, разделяющие с ним рассматриваемую строку, перевели ее копии в состояние "невозможная к использованию".
Если глобальное состояние строки в резидентном модуле "некэшированная", то строка отсылается запросившему модулю, и этот модуль продолжает приостановленные вычисления.
Если глобальное состояние строки "удаленно-разделенная", то резидентный модуль рассылает по списку всем модулям, имеющим копию строки, запрос на переход этих строк в состояние "невозможная к использованию". По получении этого запроса каждый из модулей изменяет состояние своей копии строки на "невозможная к использованию" и посылает подтверждение исполнения в модуль, инициировавший операцию записи. При этом в приостановленном модуле строка после исполнения записи переходит в состояние "удаленно-измененная".
Предпринимаются попытки повысить эффективность реализации алгоритма когерентности, в частности, за счет учета специфики параллельных программ, в которых используются асинхронно одни и те же данные на каждом временном интервале исключительно одним процессором с последующим переходом обработки к другому процессору. Такого рода ситуации случаются, например, при определении условий окончания итераций. В этом случае возможна более эффективная схема передачи строки из кэша одного процессора в кэш другого процессора.

109. Как можно изменить приоритеты прерываний в контроллере прерываний IBM PC.

Смотри вопрос 66

110. Почему точек входа PCB меньше, чем количество задач.

Смотри вопрос 44

111. В чем отличительная особенность формирования исполнительного адреса при страничной организации памяти.

В самом простом и наиболее распространенном случае страничной организации памяти (или paging) как логическое адресное пространство, так и физическое представляются состоящими из наборов блоков или страниц одинакового размера. При этом образуются логические страницы (page), а соответствующие единицы в физической памяти называют страничными кадрами (page frames). Страницы (и страничные кадры) имеют фиксированную длину, обычно являющуюся степенью числа 2, и не могут перекрываться. Каждый кадр содержит одну страницу данных. При такой организации внешняя фрагментация отсутствует, а потери из-за внутренней фрагментации, поскольку процесс занимает целое число страни ц, ограничены частью последней страницы процесса.

Логический адрес в страничной системе – упорядоченная пара (p,d), где p – номер страницы в виртуальной памяти, а d – смещение в рамках страницы p, на которой размещается адресуемый элемент. Заметим, что разбиение адресного пространства на страницы осуществляется вычислительной системой незаметно для программиста. Поэтому адрес является двумерным лишь с точки зрения операционной системы, а с точки зрения программиста адресное пространство процесса остается линейным.

Описываемая схема позволяет загрузить процесс, даже если нет непрерывной области кадров, достаточной для размещения процесса целиком. Но одного базового регистра для осуществления трансляции адреса в данной схеме недостаточно. Система отображения логических адресов в физические сводится к системе отображения логических страниц в физические и представляет собой таблицу страниц, которая хранится в оперативной памяти. Иногда говорят, что таблица страниц – это кусочно-линейная функция отображения, заданная в табличном виде.

Интерпретация логического адреса показана на рис. 8.7. Если выполняемый процесс обращается к логическому адресу v = (p,d), механизм отображения ищет номер страницы p в таблице страниц и определяет, что эта страница находится в страничном кадре p', формируя реальный адрес из p' и d.

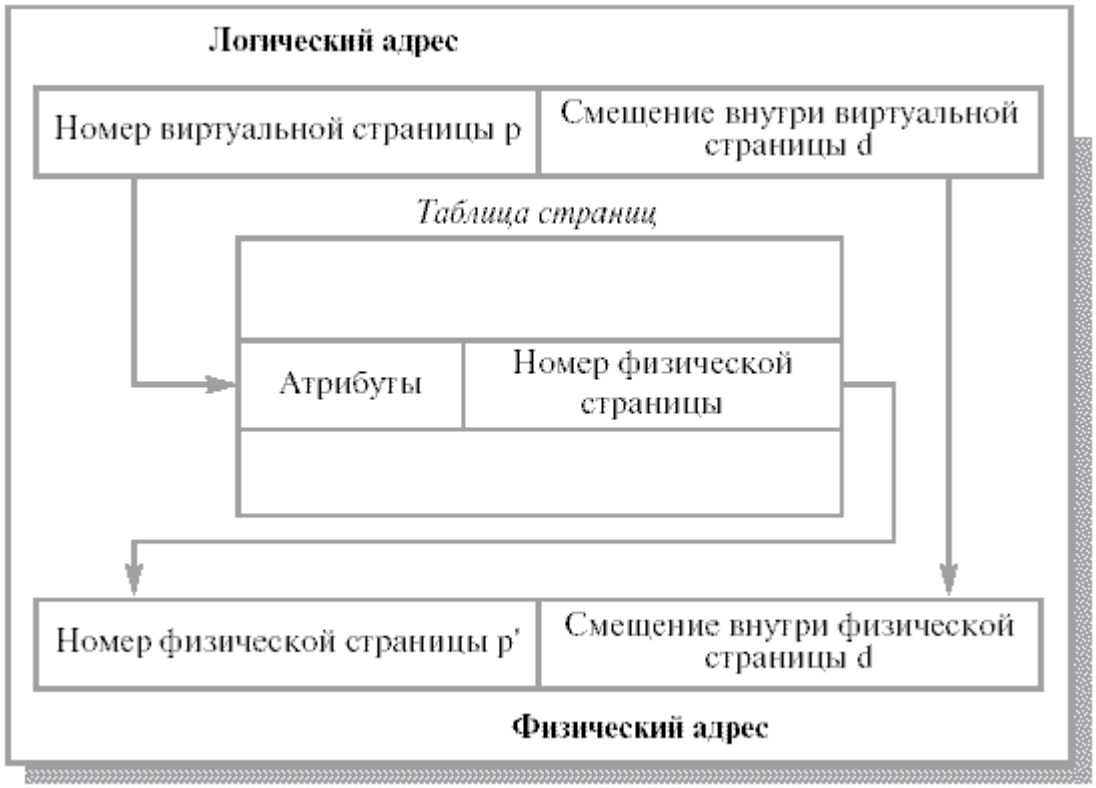


Рис. 8.7. Связь логического и физического адресов при страничной организации памяти

Таблица страниц (page table) адресуется при помощи специального регистра процессора и позволяет определить номер кадра по логическому адресу. Помимо этой основной задачи, при помощи атрибутов, записанных в строке таблицы страниц, можно организовать контроль доступа к конкретной странице и ее защиту.

Отметим еще раз различие точек зрения пользователя и системы на используемую память. С точки зрения пользователя, его память – единое непрерывное пространство, содержащее только одну программу. Реальное отображение скрыто от пользователя и контролируется ОС. Заметим, что процессу пользователя чужая память недоступна. Он не имеет возможности адресовать память за пределами своей таблицы страниц, которая включает только его собственные страницы.

Для управления физической памятью ОС поддерживает структуру таблицы кадров. Она имеет одну запись на каждый физический кадр, показывающий его состояние.

Отображение адресов должно быть осуществлено корректно даже в сложных случаях и обычно реализуется аппаратно. Для ссылки на таблицу процессов используется специальный регистр. При переключении процессов необходимо найти таблицу страниц нового процесса, указатель на которую входит в контекст процесса.

112. К какому приоритетному уровню относится кеш прерывание?

113. Назначение программы – планировщик

Задача распределения времени ЦП в системах с разделением времени решается специальной частью операционной системы, называемой планировщиком.
Модуль "планировщик" распределяет между процессами время центрального процессора. Он планирует очередность выполнения процессов до тех пор, пока они добровольно не освободят центральный процессор, дождавшись выделения ресурса, или до тех пор, пока ядро системы не выгрузит их после того, как их время выполнения превысит заранее определенный квант времени. Планировщик выбирает на выполнение готовый к запуску процесс с наивысшим приоритетом; выполнение предыдущего процесса (приостановленного) будет продолжено тогда, когда его приоритет будет наивысшим среди приоритетов всех готовых к запуску процессов.

114. Особенности размещения файлов в HPFS

С точки зрения размещения файлы, каталоги и их расширенные атрибуты (если они не помещаются в FNode) рассматриваются HPFS как наборы экстентов. Экстент — это кусок файла лежащий в последовательных секторах. Каждый экстент описывается двумя числами: номером первого сектора и длиной (в секторах). Два последовательных экстента всегда объединяются HPFS в один. Минимальный

размер экстента — один сектор. Так как расстояние между соседними битмапами свободных секторов равно 16MB, то и размер максимального экстента равен 16MB. Если файл состоит из восьми или менее экстентов, то его описание целиком хранится в FNode.

Если файл состоит более чем из восьми экстентов, то его описание может занимать несколько секторов расположенных поближе к файлу, при этом эти сектора содержат не список, а прошитое сбалансированное дерево экстентов (B+-Tree). Дерево построено так, что его разбалансировка никогда не превышает 1/3 по объему, и оно не отличается от оптимального более чем на один уровень. Корень дерева находится в FNode, причем может содержать до 12 элементов. Каждый дополнительный сектор представляющий собой ветку дерева содержит до 60 элементов, а лист — 40 элементов. Таким образом, если файл состоит из экстентов по одному сектору (этого никогда не будет!) и имеет размер 2GB, для его описания потребуется дерево следующей структуры: 12*60*60*60*40=53MB листьев и 1.7MB веток. Для случайном доступа к любой части файла при этом потребуется (в худшем случае) 5 операций чтения управляющих структур.

115. Реальные файлы состоят из 1-3 экстентов.

116. В чем сложность для операционной системы в организации многопрограммного режима работы.

Система работает в *многопрограммном* режиме, если в ней находится несколько задач в разной стадии исполнения, и каждая из них может быть прервана другой с последующим возвратом.

Многопрограммный режим в однопроцессорной системе — это организация вычислительного процесса с планированием во времени. При этом операционная система обеспечивает выполнение активизированных процессов на одном и том же оборудовании, разделяя (синхронизируя) их работу во времени. В функции такой операционной системы также входит защита влияния одного процесса на другой. Основные заботы о синхронизации взаимодействующих процессов возложены на программиста.

Многопрограммный режим в многопроцессорной системе — это планирование во времени и в пространстве. В этом режиме операционная система, выполняя одну из своих основных функций (обеспечение эффективности работы ресурсов), должна распределять активизированные процессы по процессорам (в пространстве) и синхронизировать их работу во времени. В том случае, если количество задач больше количества процессоров, задача планирования, диспетчеризации усложняется. В связи со сложностью решения задач распределения процессов по процессорам, их решение выполняется или до активизации процессов в многопроцессорной системе (статическое планирование), или решаются простыми способами, не дающими оптимального решения.
В настоящее время применение систем массового распараллеливания и распределенных систем обработки информации, когда вычислительная система может в большинстве случаев выделить столько вычислительных ресурсов, сколько требуется, временную координату планирования можно исключить, что значительно облегчает задачи операционной системы по планированию и организации вычислительных процессов.

117. Понятие окна рабочего множества.

Процессы начинают работать, не имея в памяти необходимых страниц. В результате при выполнении первой же машинной инструкции возникает page fault, требующий подкачки порции кода. Следующий page fault происходит при локализации глобальных переменных и еще один - при выделении памяти для стека. После того как процесс собрал большую часть необходимых ему страниц, page faults возникают редко.

Таким образом, существует набор страниц (P_1, P_2, ... P_n), активно использующихся вместе, который позволяет процессу в момент времени t в течение некоторого периода T производительно работать, избегая большого количества page faults. Этот набор страниц называется **рабочим множеством** W(t,T) (**working set**) процесса. Число страниц в рабочем множестве определяется параметром T, является неубывающей функцией T и относительно невелико. Иногда T называют размером окна рабочего множества, через которое ведется наблюдение за процессом (см. рис. 10.3).

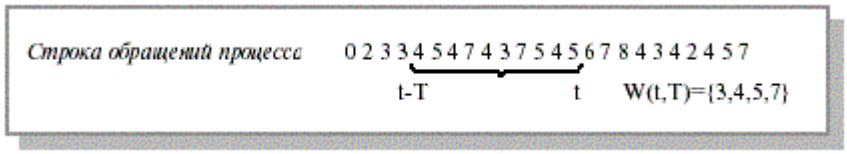


Рис. 10.3. Пример рабочего множества процесса

Легко написать тестовую программу, которая систематически работает с большим диапазоном адресов, но, к счастью, большинство реальных процессов не ведут себя подобным образом, а проявляют свойство локальности. В течение любой фазы вычислений процесс работает с небольшим количеством страниц.

Когда процесс выполняется, он двигается от одного рабочего множества к другому. Программа обычно состоит из нескольких рабочих множеств, которые могут перекрываться. Например, когда вызвана процедура, она определяет новое рабочее множество, состоящее из страниц, содержащих инструкции

процедуры, ее локальные и глобальные переменные. После ее завершения процесс покидает это рабочее множество, но может вернуться к нему при новом вызове процедуры. Таким образом, рабочее множество определяется кодом и данными программы. Если процессу выделять меньше кадров, чем ему требуется для поддержки рабочего множества, он будет находиться в состоянии трешинга.

Принцип локальности ссылок препятствует частым изменениям рабочих наборов процессов. Формально это можно выразить следующим образом. Если в период времени (t-T, t) программа обращалась к страницам W(t,T), то при надлежащем выборе T с большой вероятностью эта программа будет обращаться к тем же страницам в период времени (t, t+T). Другими словами, принцип локальности утверждает, что если не слишком далеко заглядывать в будущее, то можно достаточно точно его прогнозировать исходя из прошлого. Понятно, что с течением времени рабочий набор процесса может изменяться (как по составу страниц, так и по их числу).

Наиболее важное свойство рабочего множества - его размер. ОС должна выделить каждому процессу достаточное число кадров, чтобы поместилось его рабочее множество. Если кадры еще остались, то может быть инициирован другой процесс. Если рабочие множества процессов не помещаются в память и начинается трешинг, то один из процессов можно выгрузить на диск.

118. Решение о размещении процессов в памяти должно, следовательно, базироваться на размере его рабочего множества. Для впервые инициируемых процессов это решение может быть принято эвристически. Во время работы процесса система должна уметь определять: расширяет процесс свое рабочее множество или перемещается на новое рабочее множество. Если в состав атрибутов страницы включить время последнего использования t_i (для страницы с номером i), то принадлежность i-й страницы к рабочему набору, определяемому параметром t в момент времени t будет выражаться неравенством: $t-T < t_i < t$.

119. Особенность SCI.

Масштабируемый когерентный интерфейс – Scalable Coherent Interface (SCI), является ANSI/IEEE стандартом, определяющим невероятно гибкий высокоскоростной протокол передачи данных для соединений «точка-точка». Хотя первоначально SCI задумывался для использования в новых, более быстрых объединительных платах со скоростью передачи данных более 1Гбита/сек, его создатели заглянули далеко вперед, реализовав в нем превосходную масштабируемую транспортную среду, низкую латентность и когерентность кэша памяти. Интерфейс SCI разработан для обеспечения высокоэффективной передачи данных и совместного использования памяти между процессорами, материнскими платами или объединительными платами аппаратного блока.

SCI – это не LAN, а сетевая технология для многопроцессорных систем. По своей сути, SCI – это ведущая технология для построения масштабируемых мультипроцессорных систем, так как она позволяет избежать узких мест, присутствующих у традиционных шин и сетевых технологий.

120. Гибкость – в зависимости от требований к производительности SCI может быть внедрен как 16-битное параллельное соединение с использованием топологий кольцо или коммутируемая звезда (switched-star). Малая латентность – параллельные соединения SCI обеспечивают соединения с низкой латентностью между платами и/или системами, использующими стандартные интерфейсы объединительной платы, такие как VME, PCI или CPCI, и даже использующими конфигурации второго уровня. Доставка данных – еще одна неотъемлемая характеристика протокола SCI это «гарантированная доставка данных», обеспечивающая экстремально устойчивые и надежные решения.

Современные продукты, основанные на технологии SCI, обеспечивают одновременные соединения приема и передачи данных со скоростью 667 Мбайт/сек или 5.33 Гбита/сек, с задержкой сообщения между узлами менее 1.4 микросекунды. В будущем, продукты, основанные на SCI, будут достигать скорости двусторонних соединений в 1.333 Гбайт/сек, и емкости кольца для случайного трафика в 1 Гбайт/сек.

121. Смысл блокирования и разблокирования записей.

При открытии файла выделяется буфер, равный размеру кластера.
При загрузке ОС создается таблица дескрипторов файлов(какому процессу принадлежит файл, буфер файла). Каждому буферу подчинен указатель на текущую позицию в файле.
Разблокирование – считывание кластера целиком, из него выбирается логическая запись.
Блокирование – при любой записи в файл запись производится в буфер до тех пор, пока он не будет заполнен, тогда весь кластер пишется на диск. Когда закрываем файл, то независимо от заполненности буфера, он скидывается в файл.

122. Как обрабатывается прерывание от схем контроля.

Каждому уровню прерываний в оперативной памяти соответствуют две фиксированные области длиной в двойное слово. В этих областях размещаются старые и новые ССП, а текущие ССП в период выполнения соответствующей программы хранятся в специальном регистре ССП.
Рассматриваемые области памяти и регистр ССП при реализации прерываний используются следующим образом. При появлении какого-либо демаскированного запроса прерывания i-го уровня и отсутствии запросов прерывания других уровней в разрядах текущего ССП с 16-го по 31-й будет зафиксирован код прерывания, определяющий причину прерывания. После этого текущее ССП, фиксируемое в регистре ССП, передается в область оперативной памяти, соответствующую старому ССП i-го уровня прерываний, а из

области оперативной памяти, соответствующей новому ССП i-го уровня, в регистр ССП будет передано новое ССП. Это новое ССП и будет управлять выполнением прерывающей программы, обслуживающей возникшее прерывание.
Поскольку приоритеты между запросами прерывания и прерывающими программами устанавливаются таким образом, что прерывания, связанные с контролем, всегда имеют наивысший приоритет, то при любом сбое машины будет иметь место следующая последовательность действий. В разрядах текущего ССП с 16-го по 31-й фиксируется код прерывания от схем контроля ЭВМ, после чего данное ССП переписывается с регистра ССП в область оперативной памяти, начинающуюся с шестнадцатеричного адреса 30. Эта область памяти длиной в двойное слово соответствует области старого ССП для прерываний от схем контроля ЭВМ. Затем новое ССП выбирается из области памяти, начинающейся с шестнадцатеричного адреса 70, и передается в регистр ССП. Выбранное ССП далее рассматривается как текущее ССП и под его управлением начинает выполняться диагностическая программа.
После завершения выполнения программы реакции управление передается программе возврата на прерванную программу в «точке» ее прерывания.

Возникают при обнаружении сбоев в работе аппаратуры, например, при несовпадении четности в микросхемах памяти.

Сигналы внутренних прерываний возникают при нарушениях в работе самого микропроцессора либо при ошибках в выполняемых командах и формируются внутри схемы микропроцессора при возникновении одной из ситуаций.

Внутренние прерывания, происходят **синхронно** выполнению программы при появлении аварийной ситуации в ходе исполнения некоторой инструкции программы. Примерами являются деление на нуль, ошибки защиты памяти, обращения по несуществующему адресу. Прерывания возникают внутри выполнения команды.

Внутренние прерывания обрабатываются специальными модулями ядра.

Обобщенно последовательность действий аппаратных и программных средств по обработке прерывания можно описать следующим образом:

1. При возникновении условия прерывания происходит первичное аппаратное распознавание типа прерывания. Если прерывания данного типа в настоящий момент запрещены (приоритетной схемой или механизмом маскирования), то процессор продолжает поддерживать естественный ход выполнения команд.
2. В противном случае в зависимости от поступившей в процессор информации происходит автоматический вызов процедуры обработки прерывания, адрес которой находится в специальной таблице ОС, размещаемой либо в регистрах процессора, либо в определенном месте оперативной памяти.
3. Автоматически сохраняется некоторая часть контекста прерванного потока, которая позволит ядру возобновить исполнение потока процесса после обработки прерывания. В это подмножество включаются значения счетчика команд, слова состояния машины, хранящего признаки основных режимов работы процессора (пример слова — регистр EFLAGS в Intel Pentium), а также нескольких регистров общего назначения, которые требуются программе обработки прерывания. Может быть сохранен и полный контекст процесса, если ОС обслуживает данное прерывание со сменой процесса.
4. Одновременно с загрузкой адреса процедуры обработки прерываний в счетчик команд может автоматически выполняться загрузка нового значения слова состояния машины, которое определяет режимы работы процессора при обработке прерывания, в том числе работу в привилегированном режиме.
5. Временно запрещаются прерывания данного типа, чтобы не образовалась очередь вложенных друг в друга потоков одной и той же процедуры. Детали выполнения этой операции зависят от особенностей аппаратной платформы, например может использоваться механизм маскирования прерываний..
6. После того как прерывание обработано ядром ОС, прерванный контекст восстанавливается, и работа потока возобновляется с прерванного места.

Часть контекста восстанавливается аппаратно по команде возврата из прерываний (например, адрес следующей команды и слово состояния машины), а часть — программным способом, с помощью явных команд извлечения данных из стека. При возврате из прерывания блокировка повторных прерываний данного типа снимается.

123. Условие перехода процесса из заблокированного состояния в готовое.

Если активный процесс не выполнился за выделенный ему квант времени, то он переходит снова в готовое состояние.

Процесс, который требует дополнительных ресурсов, кроме времени процессора, либо выполнения некоторой операции другим процессом, переводится в заблокированное состояние и ожидает наступления события, которое он потребовал. Как только произошло ожидаемое событие, процесс переводится в готовое состояние.

124. Какая характеристика улучшается в ВС с частично - упорядоченной системой прерывания.

Для реализации механизма прерываний необходима аппаратная схема фиксации сигнала запроса на прерывание. Такая схема обычно содержит регистр, на котором фиксируется наличие сигналов во входных линиях **запросов на прерывания**. Объединенные схемой "ИЛИ", сигналы с разрядов регистра формируют общий сигнал о наличии запроса на прерывание. Затем все разряды регистра опрашиваются в порядке приоритетов входных линий. При этом используются 2 варианта реализации опроса:

Полноупорядоченная схема. Регистры опрашиваются по порядку приоритетов, от высшего к низшему. Это простая схема, однако при возникновении прерываний с низким приоритетом необходимо проверить *все* регистры запросов на прерывания с более высоким приоритетом.

Частично упорядоченная схема. Все прерывания делятся на **классы** и вводится 2-уровневая система приоритетов: первый уровень — среди различных классов, второй — внутри каждого класса. При этом, сначала по полноупорядоченной схеме определяется класс прерывания, на которое поступил запрос, а затем, также по полноупорядоченной схеме внутри установленного класса, определяется само прерывание. Это ускоряет поиск прерывания с низким приоритетом, однако усложняет процедуру поиска.

125. Как реагирует ВС на общий сигнал прерывания

Прерывание происходит в произвольной точке потока команд программы, которую программист не может прогнозировать.

Прерывания имеют некоторое сходство с процедурой в том, что в обоих случаях выполняется некоторая подпрограмма, обрабатывающая специальную ситуацию, а затем продолжается выполнение основной ветви программы.

В зависимости от источника, прерывания делятся на три больших класса:

- внешние;
- внутренние;
- программные;

Внешние прерывания возникают в результате действий пользователя или в результате поступления сигналов от аппаратных устройств. Данный класс прерываний является **ассинхронным** по отношению к потоку инструкций прерываемой программы.

Аппаратура процессора работает так, что асинхронные прерывания возникают между выполнением двух соседних инструкций, при этом система после обработки прерывания продолжает выполнение процесса, уже начиная со следующей инструкции.

Внутренние прерывания, происходят **синхронно** выполнению программы при появлении аварийной ситуации в ходе исполнения некоторой инструкции программы. Примерами являются деление на нуль, ошибки защиты памяти, обращения по несуществующему адресу. Прерывания возникают внутри выполнения команды.

Программные прерывания отличаются от предыдущих двух классов тем, что они по своей сути не являются «истинными» прерываниями. Программное прерывание возникает при выполнении особой команды процессора, выполнение которой имитирует прерывание, то есть переход на новую последовательность инструкций.

Прерываниям приписывается приоритет, с помощью которого они ранжируются по степени важности и срочности. О прерываниях, имеющих одинаковое значение приоритета, говорят, что они относятся к **одному уровню приоритета** прерываний.

Процедуры, вызываемые по прерываниям, обычно называют **обработчиками прерываний**, или **процедурами обслуживания прерываний**.

Аппаратные прерывания обрабатываются драйверами соответствующих внешних устройств, внутренние прерывания — специальными модулями ядра, а программные прерывания — процедурами ОС, обслуживающими системные вызовы.

Кроме этих модулей в ОС может находиться так называемый диспетчер прерываний, который координирует работу отдельных обработчиков прерываний.

Обобщенно последовательность действий аппаратных и программных средств по обработке прерывания можно описать следующим образом:

- При возникновении сигнала (для аппаратных прерываний) или условия (для внутренних прерываний) прерывания происходит первичное аппаратное распознавание типа прерывания. Если прерывания данного типа в настоящий момент запрещены (приоритетной схемой или механизмом маскирования), то процессор продолжает поддерживать естественный ход выполнения команд.
- В противном случае в зависимости от поступившей в процессор информации происходит автоматический вызов процедуры обработки прерывания, адрес которой находится в специальной таблице ОС, размещаемой либо в регистрах процессора, либо в определенном месте оперативной памяти.
- Автоматически сохраняется некоторая часть контекста прерванного потока, которая позволит ядру возобновить исполнение потока процесса после обработки прерывания. В это подмножество включаются значения счетчика команд, слова состояния машины, хранящего признаки основных режимов работы процессора (пример слова — регистр EFLAGS в Intel Pentium), а также нескольких регистров общего назначения, которые требуются программе

обработки прерывания. Может быть сохранен и полный контекст процесса, если ОС обслуживает данное прерывание со сменой процесса.

- Одновременно с загрузкой адреса процедуры обработки прерываний в счетчик команд может автоматически выполняться загрузка нового значения слова состояния машины, которое определяет режимы работы процессора при обработке прерывания, в том числе работу в привилегированном режиме.
- Временно запрещаются прерывания данного типа, чтобы не образовалась очередь вложенных друг в друга потоков одной и той же процедуры. Детали выполнения этой операции зависят от особенностей аппаратной платформы, например может использоваться механизм маскирования прерываний..
- После того как прерывание обработано ядром ОС, прерванный контекст восстанавливается, и работа потока возобновляется с прерванного места.

Часть контекста восстанавливается аппаратно по команде возврата из прерываний (например, адрес следующей команды и слово состояния машины), а часть — программным способом, с помощью явных команд извлечения данных из стека. При возврате из прерывания блокировка повторных прерываний данного типа снимается.

126. В чем смысл локальной настройки адресных констант.

127. Особенности файловой системы HPFS

Максимальный размер файла в HPFS сейчас 2GB, однако он обусловлен только размером поля под размер файла и файловый указатель (4 байта) в самой OS/2 и её API. Это не предел HPFS. Следует помнить, что в HPFS отсутствует понятие кластера, файл может занимать 1, 2, 3, 4 или любое другое количество секторов.

При создании/расширении файлов HPFS пытается минимизировать количество экстентов, используя для этого статистику, битмапы свободных секторов и другую информацию. Например, HPFS старается условно резервировать хотя бы 4 килобайта места в конце файлов, которые растут. Другой прием: расположение конкурентно растущих файлов или файлов открытых разными цепочками или процессами в разных полосах диска.

Каталоги в HPFS как и в FAT образуют древовидную структуру. Но при этом внутри каталога HPFS строит сбалансированное дерево (B*-Tree) на основе имен файлов для быстрого поиска файла по имени внутри каталога. Так, если каталог содержит 4096 файлов, FAT будет читать в среднем 64 сектора для поиска файла внутри каталога, а HPFS считает 2-4 сектора и найдет файл.

Размер блока в терминах которых выделяются каталоги равен 2KB в текущей версии HPFS. Размер записи описывающей файл зависит от размера имени файла. Если имя занимает 13 байтов (8.3), то 2-килобайтовый блок вмещает 41 описатель файлов. Блоки прошиты списком (как и описатели экстентов) для облегчения последовательного обхода.

HPFS не имеет FAT-овских проблем "утекания" дискового пространства при удалении большого количества файлов в каталоге.

При переименовании файла может возникнуть перебалансировка дерева. Эта операция может потребовать выделения дополнительных блоков на заполненном диске. В этом случае блоки берутся из специального пула, указатель на который лежит в SpareBlock.

Расширенные атрибуты и их разновидность ACL HPFS хранит в FNode. Если они не влезают в FNode HPFS хранит из почти как файл построив для этого B+-Tree. Имена расширенных атрибутов до HPFS386 не выстраивались в B-Tree.

128. Чем определяется количество PCB ?

Смотри вопрос 88

129. Дать определение - Управляющие программы ОС

Управляющая программа - системная программа, реализующая набор функций управления, который включает в себя управление ресурсами и взаимодействие с внешней средой системы обработки информации, восстановление работы системы после проявления неисправностей в технических средствах.

- Управление процессами
- Управление заданиями
- Управление памятью
- Управление вводом/выводом(ВУ)
- Управление данными(файловая система)

130. Влияние размера окна рабочего множества на эффективность работы системы.

Когда процесс выполняется, он двигается от одного рабочего множества к другому. Программа обычно состоит из нескольких рабочих множеств, которые могут перекрываться. Например, когда вызвана процедура, она определяет новое рабочее множество, состоящее из страниц, содержащих инструкции процедуры, ее локальные и глобальные переменные. После ее завершения процесс покидает это рабочее множество, но может вернуться к нему при новом вызове процедуры. Таким образом, рабочее множество определяется кодом и данными программы. Если процессу выделять меньше кадров, чем ему требуется для поддержки рабочего множества, он будет находиться в состоянии трешинга.

Принцип локальности ссылок препятствует частым изменениям рабочих наборов процессов. Формально это можно выразить следующим образом. Если в период времени (t-T, t) программа обращалась к страницам W(t,T), то при надлежащем выборе T с большой вероятностью эта программа будет обращаться к тем же страницам в период времени (t, t+T). Другими словами, принцип локальности утверждает, что если не слишком далеко заглядывать в будущее, то можно достаточно точно его прогнозировать исходя из прошлого. Понятно, что с течением времени рабочий набор процесса может изменяться (как по составу страниц, так и по их числу).

Наиболее важное свойство рабочего множества - его размер. ОС должна выделить каждому процессу достаточное число кадров, чтобы поместилось его рабочее множество. Если кадры еще остались, то может быть инициирован другой процесс. Если рабочие множества процессов не помещаются в память и начинается трешинг, то один из процессов можно выгрузить на диск.

Решение о размещении процессов в памяти должно, следовательно, базироваться на размере его рабочего множества. Для впервые инициируемых процессов это решение может быть принято эвристически. Во время работы процесса система должна уметь определять: расширяет процесс свое рабочее множество или перемещается на новое рабочее множество. Если в состав атрибутов страницы включить время последнего использования t_i (для страницы с номером i), то принадлежность i-й страницы к рабочему набору, определяемому параметром t в момент времени t будет выражаться неравенством: $t-T < t_i < t$.

131. Взаимодействие COMMAND.COM с клавиатурой.

COMMAND.COM - командный процессор, который обеспечивает интерфейс
пользователя с установкой.

Система загружает две части программы Command.com в память:

1. резидентная часть - обрабатывает все ошибки дисковых операций ввода/вывода и управляет прерываниями:

• INT 22h - адрес обработки завершения задачи;
• INT 23h - адрес реакции на CtrlBreak;
• INT 24h - адрес реакции на фатальную ошибку.

2. транзитная часть - загружается в самые старшие адреса памяти. Она содержит настраивающий загрузчик и предназначена для загрузки .COM или .EXE файлов с диска в память для выполнения.

При инициализации ядра и системы активизируются и исполняются следующие процессы операционной системы: администратор памяти, программа работы с ВУ, программа поддержки файловой системы и процесс, подчиненный таймеру. После запуска системы управление передается внутренним процедурам COMMAND.COM, среди которых можно выделить программу системного ввода, анализирующую входной поток заданий. Программа работает в режиме сторожа; входной поток заданий накапливается во входных очередях (первый уровень классического планирования процессов). Как только система определяет, что у нее есть ресурсы (наиболее важно наличие ОП), операционная система загружает планировщик как транзитную программу, которая анализирует наличие остальных ресурсов, инициализируя очередь процессов к процессору.

После завершения выполнения транзитной программы вызывается специальная функция завершения, которая в свою очередь освобождает память от транзитной программы и возвращает управление программе, вызвавшей загрузку транзитной программы (Command.com).

132. Обьяснить функцию загрузчика - выделения памяти.

Загрузчик - программа, которая подготавливает объектную программу к выполнению и инициирует ее выполнение.

Более детально функции Загрузчика следующие:

- выделение места для программ в памяти (распределение);

- фактическое размещение команд и данных в памяти (загрузка);
- разрешение символических ссылок между объектами (связывание);
- настройка всех величин в модуле, зависящих от физических адресов в соответствии с выделенной памятью (перемещение);
- передача управления на входную точку программы (инициализация).

Не обязательно функции Загрузчика должны выполняться именно в той последовательности, в какой они описаны. Опишем эти функции более подробно.

Функция распределения, по-видимому понятна из ее названия. Для размещения программы в оперативной памяти должно быть найдено и выделено свободное место в памяти. Для выполнения этой функции Загрузчик обычно обращается к операционной системе, которая выполняет его запрос на выделение памяти в рамках общего механизма управления памятью.

Функция загрузки сводится к считыванию образа программы с диска (или другого внешнего носителя) в оперативную память.

Функция связывания состоит в компоновки программы из многих объектных модулей. Поскольку каждый из объектных модулей в составе программы был получен в результате отдельного процесса трансляции, который работает только с одним конкретным модулем, обращения к процедурам и данным, расположенным в других модулях, в объектных модулях не содержат актуальных адресов. Загрузчик же "видит" все объектные модули, входящие в состав программы, и он может вставить в обращения к внешним точкам правильные адреса. Загрузчики, которые выполняют функцию связывания вместе с другими функциями, называются Связывающими Загрузчиками. Выполнение функции связывания может быть переложено на отдельную программу, называемую Редактором связей или Компоновщиком. Редактор связей выполняет только функцию связывания - сборки программы из многих объектных модулей и формирование адресов в обращениях к внешним точкам. На выходе Редактора связей мы получаем загрузочный модуль.

Функция перемещения необходимо потому, что программа на любом языке разрабатывается в некотором виртуальном адресном пространстве, в котором адресация ведется относительно начала программной секции. При написании программы и при ее трансляции, как правило, неизвестно, по какому адресу памяти будет размещена программа (где система найдет свободный участок памяти для ее размещения). Поэтому в большинстве случаев в командах используется именно адреса меток и данных. Однако, в некоторых случаях в программе возникает необходимость использовать реальные адреса, которые определяться только после загрузки. Все величины в программе, которые должны быть привязаны к реальным адресам, должны быть настроены с учетом адреса, по которому программа загружена.

133. Применение канала при синхронизации процессов

Одним из наиболее простых способов передачи информации между процессами по линиям связи является передача данных через pipe (канал, трубу или, как его еще называют в литературе, конвейер). Представим себе, что у нас есть некоторая труба в вычислительной системе, в один из концов которой процессы могут «сливать» информацию, а из другого конца принимать полученный поток. Такой способ реализует потоковую модель ввода/вывода. Информацией о расположении трубы в операционной системе обладает только процесс, создавший ее. Этой информацией он может поделиться исключительно со своими наследниками – процессами-детьми и их потомками. Поэтому использовать pipe для связи между собой могут только родственные процессы, имеющие общего предка, создавшего данный канал связи, если разрешить процессу, создавшему трубу, сообщать о ее местонахождении в системе другим процессам, сделав вход и выход трубы каким-либо образом видимыми для всех остальных.
Как правило, при создании процесса стандартный ввод и стандартный вывод, поэтому если мы создаем канал, то мы должны указать 2 процесса, между которыми он создается; оба процесса должны быть родственными(в Unix все наследники от Shell), им указываем в полях ввод и вывод ID друг друга. Дочерние процессы наследуют все назначенные файловые дескрипторы родителя.

134. Функция программы системного вывода.

Смотри вопрос 43

135. Состав резидентной части ОС.

Часть операционной системы, постоянно находящаяся в основной памяти, называется *ядром системы или резидентной* частью операционной системы.

Совокупность программ, обеспечивающих функционирование ВС и находящихся в системной области оперативной памяти, составляет ядро супервизора.

Ядро ОС содержит программы для реализации следующих функций:

- обработка прерываний;
- создание и уничтожение процессов;
- переключение процессов из состояния в состояние;
- диспетчеризацию заданий, процессов и ресурсов;

- приостановка и активизация процессов;
- синхронизация процессов;
- организация взаимодействия между процессами;
- манипулирование РСВ;
- поддержка операций ввода/вывода;
- поддержка распределения и перераспределения памяти;
- поддержка работы файловой системы;
- поддержка механизма вызова—– возврата при обращении к процедурам;
- поддержка определенных функций по ведению учета работы машины;

136. Каким образом система управления заданиями учитывает организацию памяти

137. Недостаток способа хранения расположения файла в системе UNIX.

Индексные Дескрипторы — *это объект Unix, который ставится во взаимнооднозначное соответствие с содержимым файла.* То есть для каждого ИД существует только одно содержимое и наоборот, за исключением лишь той ситуации, когда файл ассоциирован с каким-либо внешним устройством. Напомним содержимое ИД:

- поле, определяющее тип файла (каталоги и все остальные файлы);
- код привилегии/защиты;
- количество ссылок к данному ИД из всевозможных каталогов файловой системы;
- (нулевое значение означает свободу ИД)
- длина файла в байтах;
- даты и времена (время последней записи, дата создания и т.д.);
- поле адресации блоков файла.

Как видно — в ИД нет имени файла. Давайте посмотрим, как организована адресация блоков, в которых размещается файл.

В поле адресации находятся номера первых десяти блоков файла, то есть если файл небольшой, то вся информация о размещении данных файла находится непосредственно в ИД. Если файл превышает десять блоков, то начинает работать некая списочная структура, а именно, 11й элемент поля адресации содержит номер блока из пространства блоков файлов, в которых размещены 128 ссылок на блоки данного файла. В том случае, если файл еще больше — то используется 12й элемент поля адресации. Сутьего в следующем — он содержит номер блока, в котором содержится 128 записей о номерах блоках, где каждый блок содержит 128 номеров блоков файловой системы. А если файл еще больше, то используется 13 элемент — где глубина вложенности списка увеличена еще на единицу.

Таким образом мы можем получить файл размером $(10+128+128^2+128^3)*512$.

138. Если мы зададим вопрос — зачем все это надо (таблицы свободных блоков, ИД и т.д.), то вспомним, что мы рассматриваем взаимосвязь между аппаратными и программными средствами вычислительной системы, а в данном случае подобное устройство файловой системы позволяет сильно сократить количество реальных обменов с ВЗУ, причем эшелонированная буферизация в ОС Unix делает число этих обменов еще меньше.

139. Причина приостановки процесса.

Если активный процесс требует действия или ресурса, которых в данный момент операционная система не может выполнить, он переводится в подготовленное состояние и считается приостановленным. **Приостановленный** процесс не может продолжить свое выполнение до тех пор, пока его не активизирует любой другой процесс (кратковременное исключение определенных процессов в периоды пиковой нагрузки). В случае длительной приостановки процесса, его ресурсы должны быть освобождены. Решение об освобождении ресурса зависит от его природы. Основная память должна освобождаться немедленно, а вот принтер может и не быть освобожден. Приостановку процесса обычно производят в следующих случаях:

а) если система работает ненадежно и может отказать, то текущие процессы надо приостановить, исправить ошибку и затем активизировать приостановленные процессы;
б) если промежуточные результаты работы процесса вызвали сомнение, то нужно его приостановить, найти ошибку и запустить либо сначала, либо с контрольной точки;
в) если ВС перегружена, что вызвало снижение ее эффективности;
г) если для продолжения нормального выполнения процессу необходимы ресурсы, которые ВС не может ему предоставить.

Инициатором приостановки может быть либо сам процесс, либо другой процесс. В однопроцессорной ВС при однопрограммном режиме работы выполняющийся процесс может приостановить сам себя или быть принудительно остановиться с пульта управления без возможности восстановления, т.к. ни один другой процесс не может выполняться одновременно с ним. В мультипроцессорной системе или при многопрограммном режиме работы любой выполняющийся процесс может быть приостановлен другим процессом.

140. Стратегии управления привелегиями.

141. Причины возникновения "бесконечного"откладывания.

Приоритеты программ обработки прерываний зависят от употребляемости соответствующих ресурсов. Наибольший приоритет имеет программа обработки прерывания по таймеру (вызывается 18,2 раза в секунду) и наименьший - прерывание по вводу/выводу. Пользовательские процессы имеют различные

приоритеты в зависимости от порождающих их процессов и от времени выполнения на процессоре. Также необходимо следить за очередью блокированных процессов, с тем, чтобы среди них не было "бесконечно" ожидающих процессов. Вполне возможно, что событие, которое они ожидают, вообще может не произойти - тогда их надо вывести из системы. Также может оказаться, что блокированный процесс имеет слишком низкий приоритет и он может бесконечно долго находиться в режиме бесконечного откладывания в очереди блокированных процессов - тогда необходима система динамического, адаптивного изменения приоритетов.
В общем случае, система должна следить за процессами в очередях готовых и заблокированных процессов, чтобы не было бесконечно ожидающих процессов или процессов, монопольно удерживающих выделенные им ресурсы. Из сказанного следует, что эффективность системы управления процессами в значительной степени влияет на характеристики ВС, связанные с пропускной способностью. Плохая организация управления процессами может привести к тому, что некоторые процессы могут так и остаться в подготовленном состоянии.

142. Причина возникновения зацикливания приоритетов.

143. Какие программы находятся в ядре операционной системы. (Виды программ)

Ядро ОС содержит программы для реализации следующих функций:

- обработка прерываний;
- создание и уничтожение процессов;
- переключение процессов из состояния в состояние;
- диспетчеризацию заданий, процессов и ресурсов;
- приостановка и активизация процессов;
- синхронизация процессов;
- организация взаимодействия между процессами;
- манипулирование РСВ;
- поддержка операций ввода/вывода;
- поддержка распределения и перераспределения памяти;
- поддержка работы файловой системы;
- поддержка механизма вызова—— возврата при обращении к процедурам;

144. поддержка определенных функций по ведению учета работы машины;

145. Концепция рабочего множества

Смотри вопрос 91

146. Способы борьбы с фрагментацией.

Уменьшить фрагментацию при мультипрограммировании с фиксированными разделами можно, если загрузочные модули получать в перемещаемых адресах. Такой модуль может быть загружен в любой свободный раздел после соответствующей настройки.

Борьба с фрагментацией – перемещение всех занятых участков в сторону старших или младших адресов, так, чтобы вся свободная память образовала свободную единую область(Перемещаемые разделы).

Страничная организация памяти полностью исключает внешнюю фрагментацию

Для борьбы с фрагментацией памяти, а также для решения проблемы перемещения программы по адресному пространству, используется, так называемая, виртуальная память. Суть ее работы заключается в следующем. Пусть имеется некоторое адресное пространство программы, то есть то адресное пространство, в терминах которого оперирует программа. И имеется адресное пространство физическое, которое зависит от времени. Оно характеризует реальное состояние физической оперативной памяти.

В машинах, поддерживающих виртуальную память, существует механизм преобразования адресов из адресного пространства программы в физическое адресное пространство, то есть при загрузке задачи в память машины операционная система размещает реальную задачу в той оперативной памяти, которая является свободной, вне зависимости от того, является ли этот фрагмент непрерывным, либо он фрагментирован. Это первое действие выполняет операционная система. Она знает о состоянии своих физических ресурсов: какие свободны, какие заняты.

Второе: операционная система заполняет некоторые аппаратные таблицы, которые обеспечивают соответствие размещения программы в реальной оперативной памяти с адресным пространством, используемым программой. То есть можно определить, где в физической памяти размещена какая часть программы, и какая часть адресного пространства программы поставлена ей в соответствие.

После этого запускается программа, и начинает действовать аппарат (или механизм) виртуальной памяти. Устройство управления выбирает очередную команду. Из этой команды оно выбирает операнды, то есть адреса и те индексные регистры, которые участвуют в формировании адреса. Устройство

управления (автоматически) вычисляет исполнительный адрес того значения, с которым надо работать в памяти. После этого автоматически (аппаратно) происходит преобразование адреса исполнительного программного (или виртуального) в адрес исполнительный физический с помощью тех самых таблиц, которые были сформированы операционной системой при загрузке данной программы в память. И продолжается выполнение команды. Аналогично выполняется и, например, команда безусловного перехода на какой-то адрес. Точно так же устройство управления вычисляет сначала адрес исполнительный, после чего он преобразуется в адрес физический, а потом значение этого физического адреса помещается в счетчик команд. Это и есть механизм виртуальной памяти.

147. Почему и когда блокируется система прерывания.

Обработка прерывания завершается после выполнения процессором **команды** *IRET* **возврата из обработчика прерывания.** Процессор восстанавливает значения сохраненных в стеке регистров, таким образом устанавливается флаг IF и прерывания вновь разрешаются. Однако, многие обработчики прерываний содержат команду STI разрешения прерываний в своем теле, разрешая таким образом прерывания с более высоким приоритетом. При этом необходимо быть уверенным в том, что при возникновении вложенных прерываний не возникнут конфликты. Обычно это условие выполняется, и прерывания с высшим приоритетом (которые имеют большую важность для нормального функционирования системы) получают возможность быть обработанными.

Что касается прерываний данного и низшего уровней приоритетов, то они остаются заблокированными схемой анализа приоритетов, даже после завершения обработки прерывания. Для разрешения этих прерываний существует специальная **команда завершения прерывания** *(End Of Interrupt, EOI),* которая заключается в посылке определенного кода на управляющий регистр контроллера прерываний. Эта команда сбрасывает бит в регистре обслуживаемых прерываний и разрешает прохождение сигналов запроса прерываний через схему анализа приоритетов. Обычно такая команда завершает обработчик прерывания, однако она может располагаться в любом месте внутри обработчика. В последнем случае возможно вложенное прерывание того же уровня, что и обрабатываемое, т.е. будет выполняться та же самая процедура обработки, что обрабатывается в текущий момент. Такая процедура должна удовлетворять определенным требованиям: она должна быть повторно входимой (реентерабельной). Забота об этом ложится на программиста. Следует также помнить, что если команду STI в обработчике задавать не обязательно (т.к. при возврате из прерывания команда IRET восстанавливает из стека значение регистра флагов, сохраненное в момент вызова прерывания, т.е. с установленным флагом IF), то команду завершения прерывания (EOI) нужно подать обязательно, иначе будут запрещены прерывания на схеме анализа приоритетов, а текущее прерывание останется в стадии выполнения.

148. Способы хранения свободного дискового пространства

Смотри вопрос 22

149. Показать способ реализации защиты памяти при страничной организации.

Защита страничной памяти основана на контроле уровня доступа к каждой странице, возможны следующие уровни доступа:

- только чтение
- и чтение и запись
- только выполнение

В этом случае каждая страница снабжается трехбитным кодом уровня доступа. При трансформации логического адреса в физический сравнивается значение кода разрешенного уровня доступа с фактически требуемым. При их несовпадении работа программы прерывается.

При таком распределении с каждой математической страницей отождествляется специальный признак, который указывает на возможность или запрет использования соответствующей страницы при выполнении той или иной программы. Эти признаки хранятся в специальном регистре, который называют регистром защиты математических страниц. Состояние этого регистра в любой момент времени определяет доступность математических страниц. При этом защита памяти на уровне математических страниц существенно упрощается. Однако при записи программ в физических адресах механизм защиты памяти на уровне математических страниц уже не работает. В этой связи наряду с защитой памяти в поле математических адресов используют и защиту физических адресов.

При защите физических адресов в условиях страничного распределения памяти наиболее часто применяется защита памяти с использованием ключей, т. е. так называемая защита по ключам. В соответствии с принципом защиты по ключам каждой странице оперативной памяти присваивается ключ памяти. Все множество ключей хранится в так называемой памяти ключей, которая защищена от доступа рабочих программ. Запись в память ключей может производиться только супервизором. Каждой программе, размещенной в определенной странице оперативной памяти, присваивается соответствующий ключ защиты (ключ программы). Ключ защиты указывается в слове состояния программы (ССП). В процессе выполнения программы из кода адреса выделяется номер страницы, который рассматривается как входной адрес памяти ключей. Выбираемый по данному адресу ключ памяти сравнивается с ключом защиты данной программы,который является составной частью слова состояния программы. При

установлении логической равнозначности обращение к оперативной памяти разрешается. В противном случае осуществляется прерывание по защите памяти. Исключением является случай, когда код ключа защиты равен нулю, что определяет свободный доступ к любой странице оперативной памяти. Код ключа защиты супервизора всегда равен нулю. В ЕС ЭВМ при защите физических адресов оперативная память рассматривается разделенной на блоки объемом в 2048 байтов (т. е. каждый блок составляет половину страницы). Память ключей состоит из 8-разрядных записей, каждые из которых включают в себя 4-разрядный ключ защиты, разряд выборки, разряд обращения, разряд изменения и разряд четности. Если разряд выборки установлен в единичное состояние, то блок памяти с соответствующим ключом памяти защищен как от выборки, так и от записи.
Разряд обращения устанавливается в единицу при любом обращении к данному блоку. Просмотр состояния этих разрядов позволяет определить неиспользуемые страницы с целью их удаления из оперативной памяти. При этом для определения неиспользуемой страницы необходимо просмотреть разряды обращений двух блоков памяти, составляющих данную страницу.
Разряд изменения устанавливается в единицу только тогда, когда в страницу вносятся изменения. Если страница не изменяется, во внешней памяти имеется ее копия и ее следует вывести из оперативной памяти (она является кандидатом на удаление), то нет необходимости выполнять специальную программу для вывода ее во внешнюю память, а соответствующие блоки оперативной памяти можно сразу использовать. Если же страница претерпела изменения, то необходимо переписать данную страницу во внешнюю память и только после этого использовать освободившийся блок оперативной памяти.

150. Именованные каналы при синхронизации процессов
Являются однонаправленнымисредствами передачи данных, однако в отличие от просто каналов имеют имя. После создания FIFO он может быть открыт процессом на запись/чтение.
Возможны варианты:

- Если считывается меньшее число, чем есть в канале, то остаток сохраняется для дальнейшего чтения
- Если читатель считывает больше, чем есть, то это исключительная ситуация и получатель должен сам ее обрабатывать
- Если канал пуст и ни один процесс не открыл его для записи, а другой читает, то возвращается 0 и все нормально, но если 0 и кто-то открыл на запись, то блокируется

Запись в канал меньше, чем заданная емкость гарантированно атомарная(не прерываемая). Если в канал пишут несколько процессов, то каждая запись атомарная. Если в канал пишется больше, чем размер канала, то писатель блокируется, атомарность не гарантируется.

151. Почему "спулинг" подвержен тупику

Многие устройства не допускают совместного использования. Прежде всего, это устройства с последовательным доступом. Такие устройства могут стать закрепленными, то есть быть предоставленными некоторому вычислительному процессу на все время жизни этого процесса. Однако это приводит к тому, что вычислительные процессы часто не могут выполняться параллельно — они ожидают освобождения устройств ввода/вывода. Для организации использования многими параллельно выполняющимися задачами устройств ввода/вывода, которые не могут быть разделяемыми, вводится понятие *виртуальных* устройств. Использование принципа виртуализации позволяет повысить эффективность вычислительной системы.

Вообще говоря, понятие виртуального устройства шире, нежели использование этого термина для обозначения *спулинга* (SPOOLing — simultaneous peripheral operation on-line, то есть имитация работы с устройством в режиме “он-лайн”). Главная задача спулинга — создать видимость параллельного разделения устройства ввода/вывода с последовательным доступом, которое фактически должно использоваться только монопольно и быть закрепленным. Например, в случае, когда несколько приложений должны выводить на печать результаты своей работы, если разрешить каждому такому приложению печатать строку по первому же требованию, то это приведет к потоку строк, не представляющих никакой ценности. Однако можно каждому вычислительному процессу предоставлять не реальный, а виртуальный принтер и поток выводимых символов (или управляющих кодов для их печати) сначала направлять в специальный файл (*спул-файл,* spool-file)на магнитном диске. Затем, по окончании виртуальной печати, в соответствии с принятой дисциплиной обслуживания и приоритетами приложений выводить содержимое спул-файла на принтер. Системный процесс, который управляет спул-файлом, называется *спулером* (spool-reader или spool-writer).

При вводе заданий используется режим **спулинга** — режим буферизации для выравнивания скоростей при вводе и считывании информации из буфера. Система, использующая режим спулинга, сильно подвержена тупикам.

В таких системах буфер может быть полностью заполнен до того, как данные из него станут поступать на печатающее устройство, а его размера недостаточно для буферизации всех данных вывода. При этом возникает тупиковая ситуация, т.к. размер буфера недостаточен для помещения всех

поступивших данных, а принтер не сможет начать вывод. В данном случае единственным выходом является рестарт системы, но при этом теряется вся информация.

В таких системах хорошим выходом было бы неполное заполнение буфера (на 75-80%) и передача после этого управления печатающему устройству. По освобождении буфера можно его снова неполностью заполнить, затем снова перейти к распечатке и т.д.

152. Во многих современных ОС принтер начинает распечатку до заполнения буфера (по мере заполнения). При этом часть буфера (распечатанная) освобождается и туда можно заносить новую информацию. В таких системах вероятность возникновения тупиковой ситуации значительно меньше.

153. Недостатки алгоритма банкира.

При использовании алгоритма "банкира" подразумевается, что :

- системе заранее известно количество имеющихся ресурсов;
- система знает, сколько ресурсов может максимально потребоваться процессу;
- число пользователей (процессов), обращающихся к ресурсам, известно и постоянно;
- система знает, сколько ресурсов занято в данный момент времени этими процессами (в общем и каждым из них);
- процесс может потребовать ресурсов не больше, чем имеется в системе.

В алгоритме "банкира" введено понятие надежного состояния. Текущее состояние вычислительной машины называется **надежным**, если ОС может обеспечить всем текущим пользователям (процессам) завершение их заданий в течение конечного времени. Иначе состояние считается **ненадежным**. Надежное состояние — это состояние, при котором общая ситуация с ресурсами такова, что все пользователи имеют возможность со временем завершить свою работу. Ненадежное состояние может со временем привести к тупику.

Система удовлетворяет только те запросы, при которых ее состояние остается надежным, т.е. при запросе ресурса система определяет сможет ли она завершить хотя бы один процесс, после этого выделения.

Пример решения задачи выделения ресурсов по алгоритму "банкира".

Имеем 6 ресурсов и 6 процессов (рис .2.17.). В таблице показано состояние занятости ресурсов на данный момент (1 свободный ресурс).

Процесс	Выделенное число ресурсов	Требуемое число ресурсов
A	2	3
B	1	3
C	0	2
D	1	2
E	1	4
F	0	5

Рис. 2.17

Эта таблица отражает надежное состояние, т.к. существует такая последовательность выделений — освобождений ресурсов, при которой все процессы за конечное время будут завершены.

Недостатки алгоритма банкира:

- Если количество ресурсов большое, то становится достаточно сложно вычислить надежное состояние.
- Достаточно сложно спланировать, какое количество ресурсов может понадобиться системе.
- Число работающих пользователей (процессов) остается постоянным.
- Пользователи должны заранее указывать максимальную потребность в ресурсах, однако, потребность может определяться динамически по мере выполнения процесса.
- Пользователь всегда заказывает ресурсов больше, чем ему может потребоваться.
- Алгоритм требует, чтобы процессы возвращали выделенные им ресурсы в течение некоторого конечного времени. Однако, для систем реального времени требуются более конкретные гарантии. Т.е. в системе должно быть задано максимальное время захвата всех ресурсов процессом (через это время ресурсы должны быть освобождены).

154. Зачем используется многоуровневая система памяти.

Запоминающие устройства компьютера разделяют, как минимум, на два уровня: основную (главную, оперативную, физическую) и вторичную (внешнюю) память.

Основная память представляет собой упорядоченный массив однобайтовых ячеек, каждая из которых имеет свой уникальный адрес (номер). Процессор извлекает команду из основной памяти, декодирует и выполняет ее. Для выполнения команды могут потребоваться обращения еще к нескольким ячейкам основной памяти. Обычно основная память изготавливается с применением полупроводниковых технологий и теряет свое содержимое при отключении питания.

Вторичную память (это главным образом диски) также можно рассматривать как одномерное линейное адресное пространство, состоящее из последовательности байтов. В отличие от оперативной памяти, она является энергонезависимой, имеет существенно большую емкость и используется в качестве расширения основной памяти.

Эту схему можно дополнить еще несколькими промежуточными уровнями, как показано на рис. 8.1. Разновидности памяти могут быть объединены в иерархию по убыванию времени доступа, возрастанию цены и увеличению емкости.

Рис. 8.1. Иерархия памяти

Многоуровневую схему используют следующим образом. Информация, которая находится в памяти верхнего уровня, обычно хранится также на уровнях с большими номерами. Если процессор не обнаруживает нужную информацию на i-м уровне, он начинает искать ее на следующих уровнях. Когда нужная информация найдена, она переносится в более быстрые уровни.

Локальность

Оказывается, при таком способе организации по мере снижения скорости доступа к уровню памяти снижается также и частота обращений к нему.

Ключевую роль здесь играет свойство реальных программ, в течение ограниченного отрезка времени способных работать с небольшим набором адресов памяти. Это эмпирически наблюдаемое свойство известно как принцип локальности или локализации обращений.

Свойство локальности (соседние в пространстве и времени объекты характеризуются похожими свойствами) присуще не только функционированию ОС, но и природе вообще. В случае ОС свойство локальности объяснимо, если учесть, как пишутся программы и как хранятся данные, то есть обычно в течение какого-то отрезка времени ограниченный фрагмент кода работает с ограниченным набором данных. Эту часть кода и данных удается разместить в памяти с быстрым доступом. В результате реальное время доступа к памяти определяется временем доступа к верхним уровням, что и обусловливает эффективность использования иерархической схемы. Надо сказать, что описываемая организация вычислительной системы во многом имитирует деятельность человеческого мозга при переработке информации. Действительно, решая конкретную проблему, человек работает с небольшим объемом информации, храня не относящиеся к делу сведения в своей памяти или во внешней памяти (например, в книгах).

Кэш процессора обычно является частью аппаратуры, поэтому менеджер памяти ОС занимается распределением информации главным образом в основной и внешней памяти компьютера. В некоторых схемах потоки между оперативной и внешней памятью регулируются программистом, однако это связано с затратами времени программиста, так что подобную деятельность стараются возложить на ОС.

155. Способ защиты памяти при сегментной организации памяти.

Каждый сегмент имеет имя, размер и другие параметры (уровень привилегий, разрешенные виды обращений, флаги присутствия).

Каждый сегмент – линейная последовательность адресов, начинающаяся с 0. Максимальный размер сегмента определяется разрядностью процессора (при 32-разрядной адресации это 2^{32} байт или 4 Гбайт). Размер сегмента может меняться динамически (например, сегмент стека). В элементе таблицы сегментов помимо физического адреса начала сегмента обычно содержится и длина сегмента. Если размер смещения в виртуальном адресе выходит за пределы размера сегмента, возникает исключительная ситуация.

Логический адрес – упорядоченная пара v=(s,d), номер сегмента и смещение внутри сегмента.

В системах, где сегменты поддерживаются аппаратно, эти параметры обычно хранятся в таблице дескрипторов сегментов, а программа обращается к этим дескрипторам по номерам-селекторам. При этом в контекст каждого процесса входит набор сегментных регистров, содержащих селекторы текущих сегментов кода, стека, данных и т. д. и определяющих, какие сегменты будут использоваться при разных видах обращений к памяти. Это позволяет процессору уже на аппаратном уровне определять допустимость обращений к памяти, упрощая реализацию защиты информации от повреждения и несанкционированного доступа.

Каждый сегмент описывается дескриптором сегмента.
ОС строит для каждого исполняемого процесса соответствующую таблицу дескрипторов сегментов и при размещении каждого из сегментов в ОП или внешней памяти в дескрипторе отмечает его текущее местоположение (бит присутствия).
Дескриптор содержит поле адреса, с которого сегмент начинается и поле длины сегмента. Благодаря этому можно осуществлять контроль
1) размещения сегментов без наложения друг на друга
2) обращается ли код исполняющейся задачи за пределы текущего сегмента.
В дескрипторе содержатся также данные о правах доступа к сегменту (запрет на модификацию, можно ли его предоставлять другой задаче), защита.

96. Особенности операционных систем локальных вычислительных сетей.
Смотри 17

156. Что такое колено жизни.

157. Особенности применения сообщений при синхронизации процессов

В модели сообщений процессы налагают на передаваемые данные некоторую структуру. Весь поток информации они разделяют на отдельные сообщения, вводя между данными, по крайней мере, границы сообщений. Примером границ сообщений являются точки между предложениями в сплошном тексте или границы абзаца. Кроме того, к передаваемой информации могут быть присоединены указания на то, кем конкретное сообщение было послано и для кого оно предназначено. Примером указания отправителя могут служить подписи под эпиграфами в книге. Все сообщения могут иметь одинаковый фиксированный размер или могут быть переменной длины. В вычислительных системах используются разнообразные средства связи для передачи сообщений: очереди сообщений, sockets (гнезда) и т. д. Каналы сообщений всегда имеют буфер конечной длины. Когда мы будем говорить о емкости буфера для сообщений, мы будем измерять ее в сообщениях.

Для прямой и непрямой адресации достаточно двух примитивов, чтобы описать передачу сообщений по линии связи — ***send*** и ***receive***. В случае прямой адресации мы будем обозначать их так:

send(P, message)—послать сообщение message процессу **P**;
receive(Q, message) — получить сообщение message от процесса **Q**.

В случае непрямой адресации мы будем обозначать их так:

send(A, message) — послать сообщение message в почтовый ящик **A**;
receive(A, message) — получить сообщение message из почтового ящика **A**.

Примитивы ***send*** и ***receive*** уже имеют скрытый от наших глаз механизм взаимоисключения. Более того, в большинстве систем они уже имеют и скрытый механизм блокировки при чтении из пустого буфера и при записи в полностью заполненный буфер. Реализация решения задачи producer-consumer для таких примитивов становится неприлично тривиальной. Надо отметить, что, несмотря на простоту использования, передача сообщений в пределах одного компьютера происходит существенно медленнее, чем работа с семафорами и мониторами.

158. Условия выбора сектора и кластера.

Совокупность дорожек одного радиуса на всех поверхностях всех пластин называется **цилиндром** (cylinder). Каждая дорожка разбивается на фрагменты, называемые **секторами** (sectors) или **блоками** (blocks).

Все дорожки имеют равное число секторов, в которых можно максимально записать одно и то же число байт.

Сектор имеет фиксированный для конкретной системы размер, выражающийся степенью двойки. Чаще всего – 512 байт. Поскольку дорожки разного радиуса имеют одинаковое число секторов, плотность записи на дорожках различна – больше к центру.

Сектор – наименьшая адресуемая единица обмена данными с оперативной памятью.

Для того, чтобы контроллер мог найти на диске нужный сектор, необходимо задать ему все составляющие адреса сектора: номер цилиндра, номер поверхности и номер сектора.

Так как прикладной программе нужен не сектор, а некоторое количество байт, не обязательно кратное размеру сектора, типичный запрос включает чтение нескольких секторов, которые содержат требуемую информацию и одного или двух секторов с избыточными данными.

ОС при работе с диском использует собственную единицу дискового пространства, которое называют **кластером**. При создании файла на диске место ему выделяется кластерами. Например, размер кластера может быть равен 1024 байт.

159. Какая схема системы приоритетов реализованна в контроллере прерываний IBM PC

Программируемый контроллер прерываний *(Programmable Interrupt Controller, PIC)*, реализованный на микросхеме i8259, предназначен для поддержки аппаратных прерываний от восьми различных устройств. Каждое устройство имеет собственный приоритет при обслуживании заявок на прерывания. Кроме того, сигнал запроса прерывания может быть **замаскирован**, т.е. при появлении этого сигнала контроллер прерываний его игнорирует. Для увеличения количества обслуживаемых внешних устройств, микросхемы контроллеров можно каскадировать, подключая выходы дополнительных микросхем ко входам основной.

160. Контроллер прерываний можно запрограммировать для работы в нескольких режимах:

- Режим фиксированных приоритетов *(Fixed Priority, Fully Nested Mode)*. В этом режиме запросы прерываний имеют жесткие приоритеты от 0 (высший) до 7 (низший) и обрабатываются в соответствии с этими приоритетами. Если запрос с меньшим приоритетом возникает во время обработки запроса с более высоким приоритетом, то он игнорируется. BIOS при включении питания и при перезагрузке программирует контроллер прерываний на работу именно в этом режиме. В дальнейшем, режим при необходимости, можно изменить программным способом.
- Режим с автоматическим сдвигом приоритетов *(Automatic Rotation)*. В этом режиме после обработки прерывания данному уровню присваивается минимальный приоритет, а все остальные приоритеты изменяются циклически таким образом, что запросы следующего за только что обработанным уровнем имеют наивысший, а предыдущего — наинизший приоритет. Например, после обработки запроса уровня 6 приоритеты устанавливаются следующим образом: уровень 7 имеет приоритет 0, уровень 8 — приоритет 1 и т.д., уровень 5 — приоритет 6 и уровень 6 — приоритет 7. Таким образом, в этом режиме всем запросам дается возможность получить обработку.
- Режим с программно-управляемым сдвигом приоритетов *(Specific Rotation)*. Приоритеты можно изменять так же, как и в режиме автоматического сдвига, однако для этого необходимо подать специальную команду, в которой указывается номер уровня, которому нужно присвоить максимальный приоритет.
- Режим автоматического завершения обработки прерывания *(Automatic End of Interrupt)*. В этом режиме, в отличие от остальных, обработчик прерывания *не должен* при своем завершении посылать контроллеру специальный **сигнал завершения обработки аппаратного прерывания** *(End Of Interrupt, EOI)*, который разрешает обработку прерываний с текущим и более низкими приоритетами. В данном режиме эти прерывания разрешаются в момент начала обработки прерывания. Режим используется редко, т.к. все обработчики прерываний должны быть **повторно входимыми (реентерабельными)** в случае, если до завершения обработки возникнет запрос на прерывание от того же источника.

- Режим специальной маски *(Special Mask Mode).* В данном режиме можно замаскировать отдельные уровни прерываний, изменив приоритетное упорядочение обработки запросов. После отмены этого режима восстанавливается прежний порядок обработки прерываний.
- Режим опроса *(Polling Mode).* В этом режиме обработка прерываний не производится автоматически, сигналы запроса прерываний лишь фиксируются во внутренних регистрах контроллера. Процедуру обработки прерывания необходимо вызывать "вручную", в соответствии с хранящимся в контроллере номером уровня с максимальным приоритетом, по которому имеется запрос.

161. Достоинства и недостатки файловой системы UNIX

Блок начальной загрузки	Суперблок	Индексные дескрипторы	Блоки файлов	Область сохранения
0				N-M+1

Суперблок файловой системы - содержит оперативную информацию о состоянии файловой системы, а также данные о параметрах настройки файловой системы. В частности суперблок имеет информацию о

- количестве индексных дескрипторов (ИД) в файловой системе;
- размере файловой системы;
- свободных блоках файлов;
- свободных ИД;
- еще ряд данных, которые мы не будем перечислять в силу уникальности их назначения

Вот информация о суперблоке. Какие можно сделать выводы и замечания?

- суперблок всегда находится в ОЗУ; (Недостаток)
- все операции по освобождению блоков, занятию блоков файлов, по занятию и освобождению ИД происходят в ОЗУ (минимизация обменов с диском). Если же содержимое суперблока не записать на диск и выключить питание, то возникнут проблемы (несоответствие реального состояния файловой системы и содержимого суперблока). Но это уже требование к надежности аппаратуры системы.

Свойство файловой системы по оптимизации доступа, критерием которого является количество обменов, которые файловая система производит для своих нужд, не связанных с чтением или записью информации файлов. (Преимущество)

162. Что такое зацикливание приоритетов?

Смотри вопрос 137

163. Чем ограничена виртуальная память?

Смотри вопрос 83

164. Функции системы управления задачами.

Смотри вопрос 58

165. Особенности файловой системы HPFS

Смотри вопрос 6

166. Обьяснить - косвенный ввод-вывод.

Когда используются управляющие программы. Пользовательский процесс не сам осуществляет i/o

192. Структура PSW.

Каждый процессор оснащен аппаратурой для управления последовательностью выполнения команд. В эту аппаратуру входит специальный регистр, постоянно хранящий адрес следующей команды. Он называется **словом состояния программы** *(Program Status Word, PSW* Различают три вида PSW: текущее, старое и новое. Текущее PSW или PC (PAC), как уже было сказано, содержит адрес следующей исполняемой команды. Старое PSW содержит адрес команды, прерванной программы, с которой она будет выполняться после обработки прерывания и передаче управления к прерванной программе. Новое PSW содержит адрес точки входа в программу обработки прерывания.

Обычно PSW содержит следующую информацию : регистр флагов, регистр маски, указательно на следующую команду. В современных системах PSW обычно хранит флаги процессора.

193. Определить этапы развития систем управления вводом/выводом.

-- Прямое управление вводом/выводом от программ, где вопросы управления вводом выводо были возложены на программиста

-- Использование буферного регистра. Процесс заполнения буфера контролируется устройством управлении ввода вывода. После заполнения буфера процессор переключается в режим ввода вывода и забирает оттуда данные.

-- Занятие цикла ОП(Прямой доступ памяти). Используется специальное устройств КВВ(для больших систем) или КПДП (для ПК), которые управляют передачей данные из ВУ в ОП без участия процессора.

Другой вариант:

-- Режим непосредственного опроса – если процесс активизировал какое – то действие и ждет ответа до тех пор пока действие не завертится

-- способ почтового ящик а- какой то узел записывает информацию в почтовый ящик, а получатель забирает оттуда информацию

-- чистое прерывание – если появилась какая-то информация, то дается сигнал прерывания и ему предоставляется информация.

194. Принципы повышения эффективности управления данными.

Принципы повышения эффективности управления данными заключается в оптимизации ввода вывода – то есть оптимизация обращения к диску(оптимизация времени поиска и времени передачи) и оптимизации размещения файлов.

195. Какая программа выполняет запись информации в область сохранения.

Если относительно сохранения в область сохранения для модулей, то сохранение выполняет вызываемая программа.

196. Назначение - Система управления данными

Назначение системы управления данным состоит в том, чтобы данные можно было легко записать и считать(например из жестоко диска). Различают систему управления данными с точки зрения ВУ и с точки зрения ОС. Управления данными с точки зрения ВУ – оптимизация ввода вывода. Управления данными с точки зрения ОС – оптимизация размещения файлов. Файл – именованная совокупность данных, которая с точки зрения пользователя представляет собой единое целое.

197. Способы организации кеш памяти.

Рассмотрим организацию кэш-памяти более детально, отвечая на четыре вопроса об иерархии памяти.

1. Где может размещаться блок в кэш-памяти?

Принципы размещения блоков в кэш-памяти определяют три основных типа их организации:
Если каждый блок основной памяти имеет только одно фиксированное место, на котором он может появиться в кэш-памяти, то такая кэш-память называется кэшем с прямым отображением (direct mapped). Это наиболее простая организация кэш-памяти, при которой для отображение адресов блоков основной памяти на адреса кэш-памяти просто используются младшие разряды адреса блока. Таким образом, все блоки основной памяти, имеющие одинаковые младшие разряды в своем адресе, попадают в один блок кэш-памяти, т.е.

(адрес блока кэш-памяти) =

(адрес блока основной памяти) mod (число блоков в кэш-памяти)
Если некоторый блок основной памяти может располагаться на любом месте кэш-памяти, то кэш называется полностью ассоциативным (fully associative).
Если некоторый блок основной памяти может располагаться на ограниченном множестве мест в кэш-памяти, то кэш называется множественно-ассоциативным (set associative). Обычно множество представляет собой группу из двух или большего числа блоков в кэше. Если множество состоит из n блоков, то такое размещение называется множественно-ассоциативным с n каналами (n-way set associative). Для размещения блока прежде всего необходимо определить множество. Множество определяется младшими разрядами адреса блока памяти (индексом):

(адрес множества кэш-памяти) =

(адрес блока основной памяти) mod (число множеств в кэш-памяти)

Далее, блок может размещаться на любом месте данного множества.

Диапазон возможных организаций кэш-памяти очень широк: кэш-память с прямым отображением есть просто одноканальная множественно-ассоциативная кэш-память, а полностью ассоциативная кэш-память с m блоками может быть названа m-канальной множественно-ассоциативной. В современных процессорах как правило используется либо кэш-память с прямым отображением, либо двух- (четырех-) канальная множественно-ассоциативная кэш-память.

2. Как найти блок, находящийся в кэш-памяти?

У каждого блока в кэш-памяти имеется адресный тег, указывающий, какой блок в основной памяти данный блок кэш-памяти представляет. Эти теги обычно одновременно сравниваются с выработанным процессором адресом блока памяти.

Кроме того, необходим способ определения того, что блок кэш-памяти содержит достоверную или пригодную для использования информацию. Наиболее общим способом решения этой проблемы является добавление к тегу так называемого бита достоверности (valid bit).

Адресация множественно-ассоциативной кэш-памяти осуществляется путем деления адреса, поступающего из процессора, на три части: поле смещения используется для выбора байта внутри блока кэш-памяти, поле индекса определяет номер множества, а поле тега используется для сравнения. Если общий размер кэш-памяти зафиксировать, то увеличение степени ассоциативности приводит к увеличению количества блоков в множестве, при этом уменьшается размер индекса и увеличивается размер тега.

3. Какой блок кэш-памяти должен быть замещен при промахе?

При возникновении промаха, контроллер кэш-памяти должен выбрать подлежащий замещению блок. Польза от использования организации с прямым отображением заключается в том, что аппаратные решения здесь наиболее простые. Выбирать просто нечего: на попадание проверяется только один блок и только этот блок может быть замещен. При полностью ассоциативной или множественно-ассоциативной организации кэш-памяти имеются несколько блоков, из которых надо выбрать кандидата в случае промаха. Как правило для замещения блоков применяются две основных стратегии: случайная и LRU.

В первом случае, чтобы иметь равномерное распределение, блоки-кандидаты выбираются случайно. В некоторых системах, чтобы получить воспроизводимое поведение, которое особенно полезно во время отладки аппаратуры, используют псевдослучайный алгоритм замещения.

Во втором случае, чтобы уменьшить вероятность выбрасывания информации, которая скоро может потребоваться, все обращения к блокам фиксируются. Заменяется тот блок, который не использовался дольше всех (LRU - Least-Recently Used).

Достоинство случайного способа заключается в том, что его проще реализовать в аппаратуре. Когда количество блоков для поддержания трассы увеличивается, алгоритм LRU становится все более дорогим и часто только приближенным. На рисунке 37 показаны различия в долях промахов при использовании алгоритма замещения LRU и случайного алгоритма.

Размер кэш-памяти	LRU	Random	LRU	Random	LRU	Random
16 KB	5.18%	5.69%	4.67%	5.29%	4.39%	4.96%
64 KB	1.88%	2.01%	1.54%	1.66%	1.39%	1.53%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Рис. 37. Сравнение долей промахов для алгоритма LRU и случайного алгоритма замещения
при нескольких размерах кэша и разных ассоциативностях при размере блока 16 байт

4. Что происходит во время записи?

При обращениях к кэш-памяти на реальных программах преобладают обращения по чтению. Все обращения за командами являются обращениями по чтению и большинство команд не пишут в память. Обычно операции записи составляют менее 10% общего трафика памяти. Желание сделать общий случай более быстрым означает оптимизацию кэш-памяти для выполнения операций чтения, однако при реализации высокопроизводительной обработки данных нельзя пренебрегать и скоростью операций записи.

К счастью, общий случай является и более простым. Блок из кэш-памяти может быть прочитан в то же самое время, когда читается и сравнивается его тег. Таким образом, чтение блока начинается сразу как только становится доступным адрес блока. Если чтение происходит с попаданием, то блок немедленно направляется в процессор. Если же происходит промах, то от заранее считанного блока нет никакой пользы, правда нет и никакого вреда.

Однако при выполнении операции записи ситуация коренным образом меняется. Именно процессор определяет размер записи (обычно от 1 до 8 байтов) и только эта часть блока может быть изменена. В общем случае это подразумевает выполнение над блоком последовательности операций чтение-модификация-запись: чтение оригинала блока, модификацию его части и запись нового значения блока. Более того, модификация блока не может начинаться до тех пор, пока проверяется тег, чтобы убедиться в том, что обращение является попаданием. Поскольку проверка тегов не может выполняться параллельно с другой работой, то операции записи отнимают больше времени, чем операции чтения.

Очень часто организация кэш-памяти в разных машинах отличается именно стратегией выполнения записи. Когда выполняется запись в кэш-память имеются две базовые возможности:
сквозная запись (write through, store through) - информация записывается в два места: в блок кэш-памяти и в блок более низкого уровня памяти.
запись с обратным копированием (write back, copy back, store in) - информация записывается только в блок кэш-памяти. Модифицированный блок кэш-памяти записывается в основную память только когда он замещается. Для сокращения частоты копирования блоков при замещении обычно с каждым блоком кэш-памяти связывается так называемый бит модификации (dirty bit). Этот бит состояния показывает был ли модифицирован блок, находящийся в кэш-памяти. Если он не модифицировался, то обратное копирование отменяется, поскольку более низкий уровень содержит ту же самую информацию, что и кэш-память.

Оба подхода к организации записи имеют свои преимущества и недостатки. При записи с обратным копированием операции записи выполняются со скоростью кэш-памяти, и несколько записей в один и тот же блок требуют только одной записи в память более низкого уровня. Поскольку в этом случае обращения к основной памяти происходят реже, вообще говоря требуется меньшая полоса пропускания памяти, что очень привлекательно для мультипроцессорных систем. При сквозной записи промахи по чтению не влияют на записи в более высокий уровень, и, кроме того, сквозная запись проще для реализации, чем запись с обратным копированием. Сквозная запись имеет также преимущество в том, что основная память имеет наиболее свежую копию данных. Это важно в мультипроцессорных системах, а также для организации ввода/вывода.

Когда процессор ожидает завершения записи при выполнении сквозной записи, то говорят, что он приостанавливается для записи (write stall). Общий прием минимизации остановов по записи связан с использованием буфера записи (write buffer), который позволяет процессору продолжить выполнение команд во время обновления содержимого памяти. Следует отметить, что остановы по записи могут возникать и при наличии буфера записи.

198. Как найти место расположения программы начальной загрузки.

Программа начальной загрузки находится в MBR. После нее идет резидентный том(том операционной системы).

199. Чем отличается чистая страничная организация памяти от страничной по запросам.

Чисто страничная организация памяти подразумевает, что все страницы программы находятся в ОП. При страничной организации по запросам в таблице страниц появляется бит присутствия и страницы грузятся в ОП по мере необходимости. При каждом обращении к памяти происходит чтение из таблицы страниц информации о виртуальной странице, к которой произошло обращение. Если данная

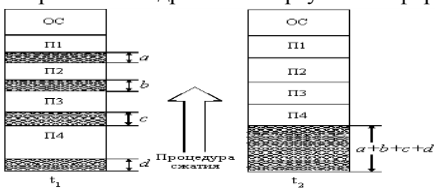
страница находится в оперативной памяти то производится обращение. Если же нужная страница в данный момент выгружена на диск, то происходит так называемое страничное прерывание.

200. Недостаток организации памяти с перемещаемыми разделами.

Память с перемещаемые разделами – это организация памяти, смысл которой заключается в следующем. Есть одна очередь заданий и память выделяется по мере поступления. Как только система определила наличие нужного количества внутренней фрагментации(внутренняя фрагментация – если во время загрузки программы в памяти возникают свободные области, которые эта программа не заняла), то производится перемещения всех программ в низкоадресное пространство памяти. Недостатки:

- проблемы с настройкой адресных констант; (возможно применение схем текущей или глобальной переадресации)
- значительные временные затраты на перемещение программ в низкоадресное пространство памяти

Одним из методов борьбы с фрагментацией является перемещение всех занятых участков в сторону старших либо в сторону младших адресов, так, чтобы вся свободная память образовывала единую свободную область . В дополнение к функциям, которые выполняет ОС при распределении памяти переменными разделами, в данном случае она должна еще время от времени копировать содержимое разделов из одного места памяти в другое, корректируя таблицы свободных и занятых областей. Эта процедура называется "сжатием". Сжатие может выполняться либо при каждом завершении задачи, либо только тогда, когда для вновь поступившей задачи нет свободного раздела достаточного размера. В первом случае требуется меньше вычислительной работы при корректировке таблиц, а во втором - реже выполняется процедура сжатия. Так как программы перемещаются по оперативной памяти в ходе своего выполнения, то преобразование адресов из виртуальной формы в физическую должно выполняться динамическим способом.



Хотя процедура сжатия и приводит к более эффективному использованию памяти, она может потребовать значительного времени, что часто перевешивает преимущества данного метода.

201. Почему и когда блокируется система прерывания.

202. Средства межпроцессного взаимодействия – сокеты. Их особенности.

См. 209.

203. Сущность работы программы согласования файловой системы.

Программа согласования файловой системы ведет борьбу с рассогласованием файловой системы. Система пытается проверить соответствуют записи занятых файлов реальным записям. Рассогласования может появится например при непредвиденном выключении системы, когда информация о записях файлов (например таблицы ФАТА), не была скопирована на жесткий диск, хотя физически информация поменялась.

В программах согласования составляют 2 таблицы: занятых и свободных кластеров.

Далее проверяются записи, если есть указатель на кластер тогда в таблицу занятых кластеров вносится 1. Если он свободен – тогда 1 в таблицу свободных кластеров.

Тогда сочетание 1(таблица занятых) и 0 (таблица свободных) – это нормальное сочетание, а вот 1 (таблица занятых) и 1 (таблица свободных) – несогласования. Если на 1 и тот же кластер ссылается 2 файла, то фиксируется ошибка и кластер дублируется еще раз и ссылка расшивается. Часто для повышения эффективной работы с файлом рекомендуется отображение файла в ОП.

204. Что такое виртуальная операционная система.

Виртуальная ОС - Так называемая ОС, которая позволяет многим пользователям работающим на одной и той же технической базе(одно ус-во) одновременно работать в различных операционных средах. Виртуальные ОС – разрешает нескольким пользователям работать на одной и той же машине на разных ОС.

205. Проблемы управления параллельными процессами. Почему они сложны для реализации.

Добавляется планирование в пространстве; появляется проблема синхронизации процессов. ЕСЛИ в системе нет одинаковых устройств то очереди обрабатываются по линейной схеме => планирование только во времени ИНАЧЕ планирование во времени и в пространстве. Для планирования параллельных процессор используют семиуровневую модель планирования. Семиуровневая модель – схема системы планирования с учетом (Маштабируемость, Разделяемость, Параллельность) 1. Предварительное (входное) планирование исходного потока заявок (задача фильтрации) 2. Структурный анализ взаимосвязи входного потока заявок по ресурсам с определением общих ресурсов (Анализ) 3. Структурный анализ заявок и определение возможности распараллеливания (Задача распараллеливания) 4. Адаптация распределения работ соответственно особенностям ВС (Задача адаптирования) 5. Составление плана – расписания выполнения взаимосвязанных процедур. Оптимизация плана по времени решения и кол-ву ресурсов (Задача Оптимизации) 6. Планирование потока задач претендующих на захват времени процессора на каждый прцессор – задача распределения. 7. Выделение процессорного времени, активизация задач. Перераспределение работ в ВС, при отказе оборудования (задача распределения – перераспределения).

206. Достоинства и недостатки страничной организации памяти.

Страничная организация памяти — это такой способ управления памятью, при котором пространство адресов памяти разбивается на блоки фиксированной длины. При страничной организации все пространство оперативной памяти физически делится на отрезки равной длины, называемые физическими страницами. Программы и данные делятся на называемые страницами. Размер страницы кратен 2^n, потому что убирается 1 сложение при формировании исполнительного адреса.

Существует 3 вида страничной организации памяти: чистостраничная организаця; страничная организация по запросам; виртуальная организация памяти

Достоинства:

a.

Недостатки:

b. появляются проблемы фрагментации

207. Обработка кэш промаха.

В современных машинах используется 2-х уровневые Кеши, один из которых является ассоциативной памятью, в которой реализован TLB-буфер – буфер быстрого преобразования адреса. В кеше отображаются страницы, которые наиболее часто используются. Если в TLB есть страница, то идет обращение в кеш. Если страницы в TLB нету, то идет обращение к таблице страниц по прерыванию, если в таблице страниц нету страницы, тогда производится еще одно прерывания и страница подкачивается в ОП с жесткого диска.

208. Зачем используется маскирование прерываний.

После того как прерывание замаскировано контроллер игнорирует появление сигнала этого прерывания. Осуществляется путем загрузки регистра маски КПП. Используется если программист хочет написать свой обработчик прерываний.

209. Виды сокетов и их применение.

Межпроцессное взаимодействие, реализованное одним из вышеописаных способов имеет общий существенный недостаток — они могут быть реализованы для процессов выполняющихся на одном компьютере.

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.

Сокеты – это средства межпроцессорного взаимодействия между процессорами разных вычислительных систем

В BSD UNIX реализованы следующие основные типы сокетов:

- *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. Если для IPC System V используются ключи, то имена сокетов представлены *адресами.*

210. Связь модулей по данным. Виды связей.

Существует 2 вида связей между модулями:
-- по данным
-- по управлению
Связь по данным может быть через общие регистры или через адрес списка параметров. Связь по данным – данные через общий регистр или через системный регистр передается адрес списка параметров.

211. Какую стратегию планирования реализует планировщик.

Стратегия планирования определяется видом системы, в которой работает планировщик:
-- Динамическое (план составляется на том же оборудовании на котором выполняется решение задач во времени);
-- Статическое (план решения задач может составляться на другом оборудовании). Планировщик должен оптимально распределить задачи на ресурсы;
-- Балансовое (решение задачи балансирования нагрузки узлов в сети).
Пространственный планировщик – решает задачу максимального паросочетания.
Планировщик в неоднордных вычислительных системах с колиеством ресурсов больше 1 решает задачи планирования как в пространстве так и во времени.

212. Дать определение методу доступа к данным

Вопросы представления данных тесно связаны с операциями, при помощи которых эти данные обрабатываются. К числу таких операций относятся: выборка, изменение,включение и исключение данных. В основе всех перечисленных операций лежит операция доступа, которую нельзя рассматривать независимо от способа представления.

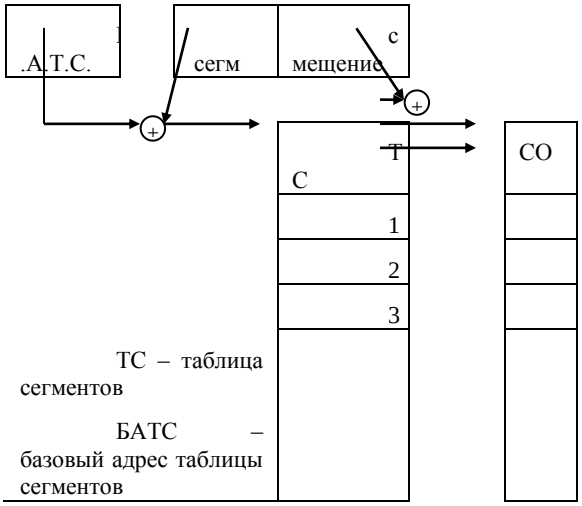
В задачах поиска предполагается, что все данные хранятся в памяти с определенной идентификацией и, говоря о доступе, имеют в виду прежде всего доступ к данным (называемым ключами), однозначно идентифицирующим связанные с ними совокупности данных.

Пусть нам необходимо организовать доступ к файлу, содержащему набор одинаковых записей, каждая из которых имеет уникальное значение ключевого поля. Самый простой способ поиска - последовательно просматривать каждую запись в файле до тех пор, пока не будет найдена та, значение ключа которой удовлетворяет критерию поиска. Очевидно, этот способ весьма неэффективен, поскольку записи в файле не упорядочены по значению ключевого поля. Сортировка записей в файле также неприменима, поскольку требует еще больших затрат времени и должна выполняться после каждого добавления записи. Поэтому, поступают следующим образом - ключи вместе с указателями на соответствующие записи в файле копируют в другую структуру, которая позволяет быстро выполнять операции сортировки и поиска. При доступе к данным вначале в этой структуре находят соответствующее значение ключа, а затем по хранящемуся вместе с ним указателю получают запись из фала.

Существуют два класса методов, реализующих доступ к данным по ключу:
методы поиска по дереву,
методы хеширования.

213. Как реализуется доступ к памяти при косвенном методе доступа.

ОС использует GDT для того, чтобы обеспечить разным программам доступ к другой области памяти. То, что подобная архитектура предусматривает косвенный доступ к памяти посредством LDT или GDT, позволяет системе использовать в качестве сегмента любой подходящий участок физической памяти. Принадлежащие одной программе сегменты вовсе не должны следовать в физической памяти друг за другом и даже могут иметь разный размер. Что касается самих программ, они имеют доступ ко всей той памяти, что описана в соответствующих GDT и LDT. Программа не ничего не знает и не заботится о том, где именно располагается тот или иной сегмент в физической памяти.



214. Какая системная программа подготавливает информацию для работы загрузчика.

Для работы загрузчика информация подготавливает редактор связей.

215. Драйвер - управляющая или обрабатывающая программа и почему?

Драйвер – обрабатывающая программа.

216. Структура ядра супервизора.

Совокупность программ, обеспечивающих функционирование ВС и находящихся в системной области оперативной памяти, составляет ядро супервизора.

Структура ядра:

-- резидентные программы

-- системные таблицы

217. Основная особенность страничной организации памяти по запросам и чем она отличается от виртуальной организации?

Страничная организация памяти по запросам предполагает, что в каждой строчке таблицы страниц появляется бит присутствия и страница грузится в ОП по пере необходимости. Виртуальная организация памяти подразумевает работу системы с программами объем которых значительно превышает реальный размер памяти. Отсюда возникает проблемы: какая часть памяти должна находится в ОП?; В случае отсутствия нужной страницы памяти – какую страницу удалить. Разница между этими организациями заключается в том, что в организации по запросам страницы, к которым непосредственно обращались грузились в ОП, система не могла работать с программой, размер которой больше ОП. При виртуальной организации памяти система может работать с программами, размер которых превышает размер ОП,

218. Особенности файловой системы HPFS

HPFS - сокращенное название высокопроизводительной файловой системы (high performance file system), совместно разработанной в 1989 году корпорациями IBM и Microsoft. Первые 16 секторов раздела HPFS составляют загрузочный блок. Эта область содержит метку диска и код начальной загрузки системы. Во время форматирования раздела HPFS делит его на полосы по 8 Мбайт каждая. Каждая полоса - ее можно представить себе как виртуальный "мини-диск" - имеет отдельную таблицу объемом 2 Кбайт, в которой указывается , какие секторы полосы доступны, а какие заняты. Чтобы максимально увеличить протяженность непрерывного пространства для размещения файлов, таблицы попеременно располагаются в начале и в конце полос (рисунок 9.3). Этот метод позволяет файлам размером до 16 Мбайт (минус 4 Кбайта, отводимые для размещения таблицы) храниться в одной непрерывной области.

Затем файловая система HPFS оценивает размер каталога и резервирует необходимое пространство в полосе, расположенной ближе всего к середине диска. Сразу же после форматирования объем диска в HPFS кажется меньше, чем в FAT, так как заранее резервируется место для каталогов в центре диска. Место резервируется в середине диска для того, чтобы физические головки, считывающие данные, никогда не проходили более половины ширины диска. Поиск свободное места произвоится по битовой маске в каждой зарезервированной таблице. HPFS использует для хранения элементов каталогов структуру данных, называемую B-деревом. Каждый элемент каталога начинается с числа, представляющего длину элемента, которая изменяется в зависимости от длины имени файла. Затем следуют время и дата создания файла, его размер и атрибуты (только для чтения, архивный, скрытый и системный), а также указатель на F-узел файла. Каждый файл (и каталог) имеет F-узел - структуру данных, занимающую один сектор и содержащую принципиально важную информацию о файле. F-узел содержит указатель на начало файла, первые 15 символов имени файла, дополнительные временные маркеры последней записи и последнего доступа, журнал, хранящий информацию о предыдущих обращениях к файлу, структуру распределения, описывающую размещение файла на диске, и первые 300 байт расширенных атрибутов

файла. (Расширенные атрибуты редко занимают более 300 байт, что фактически означает, что HPFS для получения этой информации приходится читать на один сектор меньше, чем FAT.) Программы LAN Server и LAN Manager фирмы IBM также сохраняют в F-узле информацию об управлении пользовательским доступом (Access Control). Заметьте, что F-узлы хранятся в смежных с представляемыми ими файлами секторах, поэтому, когда файл открывается, то четыре автоматически считываемых в кэш сектора содержат F-узел и три первых сектора файла.
В системе HPFS вместо таблицы размещения файлов применяется битовый массив, который содержит флаг, помечающий используемые секторы. Если область битового массива будет разрушена, пользователь этого не заметит, даже если это случится во время работы системы. F-узел файла также содержит информацию о размещении каждого файла. Поэтому область битового массива может быть восстановлена после поиска этой информации в F-узлах. Пользователь не увидит даже предупреждения - поиск выполняется автоматически

219. Смысл глобальной и локальной переадресации и когда они применяются.

Глобальная перенастройка адресных констант – перенастройка всех адресных констант, локальная – вычисление адреса той переменной, которая находится реально в ОП. Локальная перенастройка адресных констант применяется в связи с тем, что большая часть кода вообще не используется.

220. Смысл многоуровневой памяти.

Смысл многоуровневой памяти заключается в расширении адресного пространства системы. То есть для программиста существует одно адресное пространство, а фактически это пространство делится на несколько уровней, которые могут иметь разную физическую структуру. При обращении к памяти происходит преобразования виртуального адреса в реальный. Чаще всего используется двухуровневая модель памяти – Оперативная память – Жесткий диск.

221. Способы повышения системы управления данными.

См. 194.

222. Принципы повышения эффективности работы ВС.

Смотреть вопрос 308

223. Перечислить методы доступа к данным.

Смотри 212

224. К какому приоритетному уровню относится кэш прерывание.

Кэш прерывание – прерывание цепей контроллера

---

225. Преимущества применение сообщений при синхронизации процессов

Сообщения применяются для синхронизации времени. Изначально процессы по некоторому алгоритму выбирают процесс-координатора.

Выбранный процесс координирует все процессы:
Сообщения точка-точка (если известно, кто потребитель).
Если неизвестно, кто потребитель, то:
    сообщения широковещательные;
    сообщения в ответ на запрос.
Если неизвестно, кто потребляет и кто производит, то:
    сообщения и запросы через координатора:
широковещательный запрос

---

226. Уровни планирования в параллельной системе.

Интенсивное внедрение распределенных систем и систем массового распараллеливания требует изменения и дополнения системы организации и планирования вычислительного процесса, например:

- в функции ПВ добавляется структурный анализ взаимосвязи входного потока заявок;
- требуется новый тип планировщика (назовем его ПТ — планировщик транспортный, рис. 4.17) для распараллеливания заданий, синхронизации процессов по данным — обеспечения поступления требуемых данных и поддержки связей между вычислительными узлами при реализации связи по данным;
- обработанные задания (или их распараллеленные модули) транслятором ОС и ПТ ставятся в новую очередь — буфер БУФ;
- к ПП добавляется функция *адаптирования*, при выполнении которой задания распределяются соответственно особенностям данной системы (например: специальные схемы для систем гиперкуб, транспьютерных систем или дополнительная схема для неоднородной среды).
- к ПН добавляется функция балансирования в случае реконфигурации (отказа некоторых элементов) системы.

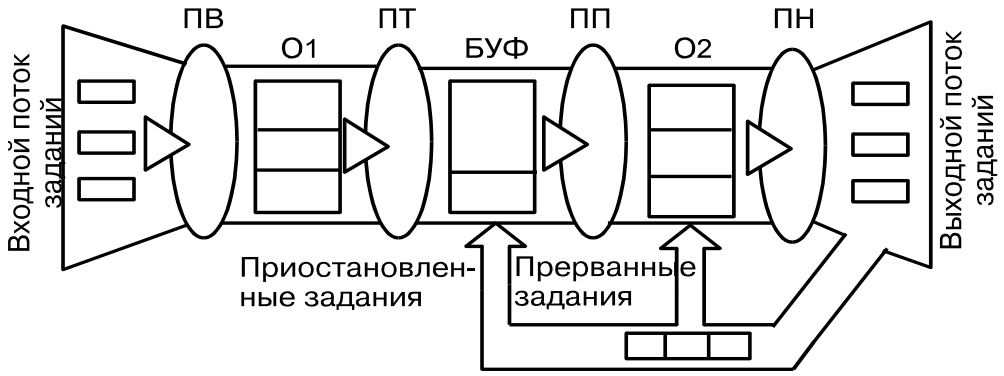


Рис. 4.17. Схема прохождения заданий через ВС

ПВ - плнировщик высокого уровня, ПП – планировщик промежуточного уровня, ПН – планировщик нижнего уровня.

Полную систему планирования в этом случае можно представить в виде семиуровневой модели (табл. 4.1), где каждый уровень часто рассматривают как отдельную задачу. Порядок выполнения уровней может быть изменен в зависимости от особенности системы и целей задачи планирования.

ПВ	1	Предварительное входное планирование исходного потока заявок, претендующих на захват ресурсов вычислительной системы	Задача Ввода
	2	Структурный анализ взаимосвязи входного потока заявок по ресурсам и определение общих ресурсов	Задача анализа
ПТ	3	Структурный анализ заявок и определение возможности распараллеливания каждой работы	Задача распараллеливания
ПП	4	Адаптирование распределения работ соответственно особенностям вычислительной системы	Задача адаптирования
	5	Составление расписания выполнения взаимосвязанных процедур: оптимизация плана по времени решения, количеству используемых ресурсов и количеству пересылок	Задача оптимизации
	6	Планирование потока процессов, претендующих на захват времени процессора/процессоров вычислительной системы	Задача распределения
ПН	7	Выделение времени процессоров ВС активизированным процессам, перераспределение (реконфигурация) работ в вычислительной среде (отказ оборудования)	Задача распределения и перераспре-деления

---

227. Показать принцип защиты памяти при страничной организации.

При помощи атрибутов, записанных в строке таблицы страниц, можно организовать контроль доступа к конкретной странице и ее защиту. (например, биты защиты: 0 - read/write, 1 - read only ...)

---

228. Почему применяются многоуровневые системы памяти?

В идеале память должна быть максимально быстрой (быстрее, чем обработка одной инструкции, чтобы работа центрального процессора не замедлялась обращением к памяти), достаточно большой и чрезвычайно дешевой. На данный момент не существует технологий, удовлетворяющих всем этим требованиям, поэтому используется другой подход. Системы памяти конструируются в виде иерархии слоев.

---

229. Какую информацию и как компилятор передает настраивающему загрузчику.

Если используется настраивающийся загрузчик, а в нем всегда выполняется редактирование до исполнения, то компилятор подготавливает информацию для настройки и подключения (внутренние и внешние векторы), какие модули подключены и какие внутренние функции и процедуры могут использоваться в других модулях. Компилятор в каждой скомпилированной строке отмечает бит переместимости, т.е.отмечается каждая команда, ее можно преобразовать в адресном виде.

---

230. **Чем и когда определяется количество РСВ?** (Я даже не знаю что он имеет в виду, но есть два варианта и оба из его книги)

Количество PCB определяется количеством выполняемый в данный момент процессов.

1))))) *Блок управления процессом (PCB — process control block)*

Выполнение функций ОС, связанных с управлением процессами, осуществляется с помощью **блока управления процессом** (PCB). **Вход в процесс** (фиксация системой процесса) — это создание его блока управления (PCB), а **выход из процесса** — это его уничтожение, т. е. уничтожение его блока управления.

Таким образом для каждого активизированного процесса система создает PCB, в котором в сжатом виде содержится информация о процессе, используемая при управлении. PCB — это системная структура данных, содержащая определенные сведения о процессе и имеющая следующие поля:

1. Уникальный индентификатор процесса (имя)
2. Текущее состояние процесса.
3. Приоритет процесса.
4. Указатели участка памяти выделенного программе, подчиненной данному процессу.
5. Указатели выделенных ему ресурсов.
6. Область сохранения регистров.
7. Права процесса (список разрешенных операций)
8. Связи зависимости в иерархии процессов (список дочерних процессов, имя родительского процесса)
9. Пусковой адрес программы, подчиненной данному процессу.

Когда ОС переключает процессор с процесса на процесс, она использует области сохранения регистров в PCB для запоминания информации, необходимой для рестарта (повторного запуска) каждого процесса с точки прерывания, когда он в следующий раз получит в свое распоряжение процессор. Количество процессов в системе ограничено и определяется самой системой, пользователем во время генерации ОС или при загрузке. Неудачное определение количества одновременно исполняемых программ может привести к снижению полезной эффективности работы системы, т.к. переключение процессов требует выполнения дополнительных операций по сохранению и восстановлению состояния процессов. Блоки управления системных процессов создаются при загрузке системы. Это необходимо, чтобы система выполняла свои функции достаточно быстро, и время реакции ОС было минимальным. Однако, количество блоков управления системными процессами меньше, чем количество самих системных процессов. Это связано с тем, что структура ОС имеет либо оверлейную, либо динамически — последовательную структуру иерархического типа, и нет необходимости создавать для программ, которые никогда не будут находиться одновременно в оперативной памяти, отдельные PCB. При такой организации легко учитывать приоритеты системных процессов, выстроив их по приоритетам заранее при инициализации системы. Блоки управления проблемными (пользовательскими) процессами создаются в процессе активизации процессов динамически. Все PCB находятся в выделенной системной области памяти.

В каждом PCB есть поле состояния процесса. Все блоки управления системными процессами располагаются в порядке убывания приоритетов и находятся в системной области памяти. Если приоритеты системных блоков можно определить заранее, то для проблемных процессов необходима таблица приоритетов проблемных программ. Каждый блок PCB имеет стандартную структуру, фиксированный размер, точку входа, содержит указанную выше информацию и дополнительную информацию для синхронизации процессов. Для синхронизации в PCB имеются четыре поля:

1-2. Поля для организации цепочки связи.

3-4. Поля для организации цепочки ожидания.

В цепочке связи указывается адрес PCB вызываемого (поле 1) и вызывающего (поле 2) процесса.

В цепочке ожидания, в поле 3 указывается адрес PCB вызываемого процесса, если вызываемый процесс занят. В поле 4 занятого процесса находится число процессов, которые ожидают данный.

Если процесс А пытается вызвать процесс В, а у процесса В в PCB занята цепочка связей, то есть он является вызываемым по отношению к другим процессам, тогда адрес процесса В записывается в цепочке ожидания PCB процесса А, а в поле счетчика ожидания PCB процесса В добавляется 1. Как только процесс В завершает выполнение своих функций, он передает управление вызывающему процессу следующим образом: В проверяет состояние своего счетчика ожидания, и, если счетчик больше 0, то среди PCB других процессов ищется первый (по приоритету или другим признакам) процесс, в поле 3 PCB которого стоит имя ожидаемого процесса, в данном случае В, тогда управление передается этому процессу.

2)))))))))))))))))))))))

При появлении сигнала прерывания управление передается ОС, которая запоминает состояние прерванного процесса в области сохранения регистров PCB. Далее ОС анализирует тип прерывания и передает управление соответствующей программе обработки этого прерывания. Инициатором прерывания может быть выполняющийся процесс или оно может быть вызвано некоторым событием, связанным или даже не связанным с этим процессом.

Рассмотрим механизм передачи управления **программе обработки прерывания (IH)**. Как было сказано выше, ОС запоминает состояние прерванного процесса и передает управление IH. Эта операция называется **переключением контекста**. При реализации переключения используются **слова состояния программы (PSW)**, с помощью которых осуществляется управление порядком выполнения команд. В **PSW** содержится информация относительно состояния процесса, обеспечивающая продолжение прерванной программы на момент прерывания.

Существуют три типа PSW: текущее, новое и старое PSW (рис.2.13.). Адрес следующей команды (активной программы), подлежащей выполнению, содержится в текущем PSW, в котором также указываются и типы прерываний, разрешенных и запрещенных в данный момент.

Процессор реагирует только на разрешенные прерывания, а запрещенные игнорирует или задерживает их выполнение. Адрес следующей команды, хранящийся в **текущем PSW**, в командном цикле передается счетчику команд, а адрес новой команды записывается в PSW (по первым двум битам команды можно определить ее длину). Кроме этого, в PSW находятся: признак результата предыдущей команды, поля масок некоторых прерываний (например, каналов и прерываний исключительных ситуаций).

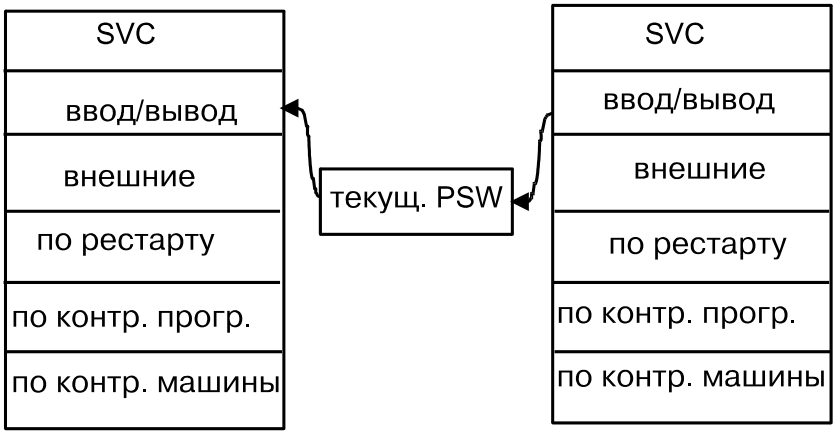


Рис. 2.13

В **новых PSW** содержатся адреса, по которым резидентно размещаются программы обработки прерываний. При возникновении разрешенного прерывания, в одном из **старых PSW** (соответствующем типу прерывания) система сохраняет содержимое текущего PSW — адрес следующей команды данного процесса, которая должна выполниться по окончании обработки прерывания и передаче управления данному прерванному процессу (т.е. адрес команды, которая следует за текущей в данном процессе и которая должна была бы выполниться при отсутствии сигнала прерывания). Таким образом, обеспечивается корректное возвращение в прерванный процесс после обработки прерывания.

Одно из новых PSW, которое содержит адрес обработчика прерывания, соответствующего поступившему сигналу прерывания, записывается в текущее PSW, и происходит переход на соответствующую программу обработки прерывания. Таким образом с помощью механизма смены PSW организуется вход в прерывающую программу. Старые PSW могут храниться в постоянной области памяти вместе с векторами прерываний; в стеке той программы, которая прерывается или в PCB прерванного процесса. Если старое PSW находится в постоянной области памяти, то количество выделенных для этого PSW равно количеству классов прерываний. В этом случае глубина прерываний определяется количеством старых PSW.

При выполнении процедуры выхода из прерывания, ОС может передать управление процессору для продолжения выполнения прерванного процесса, если ОС не допускает перехвата процессора более приоритетному процессу (первому в очереди готовых процессов), тогда данный процесс переводится в готовое состояние.

231. Концепция окна рабочего множества.

Деннинг (De68) предложил оценивать интенсивность подкачки страниц для программы согласно концепции, которую он назвал теорией поведения программы с рабочим множеством. Если говорить неформально, то рабочее множество — это подмножество страниц, к которым процесс активно обращается. Деннинг утверждал, что для обеспечения эффективного выполнения программы необходимо, чтобы ее рабочее множество находилось в первичной

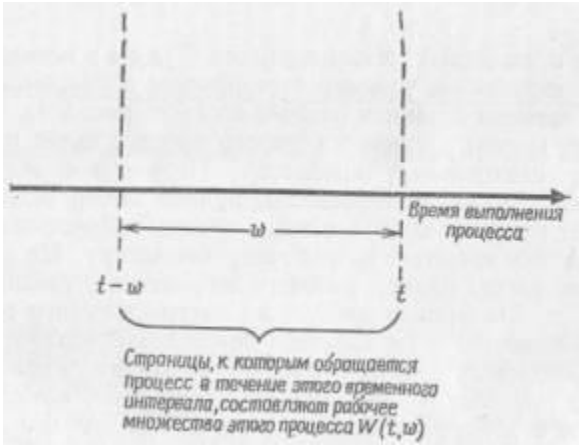


Рис. 9.4 Одно из определений рабочего множества страниц процесса.

памяти. В противном случае может возникнуть режим чрезмерно интенсивной подкачки страниц, так называемое пробуксовывание, или трешинг (De68b), поскольку программа будет многократно подкачивать одни и те же страницы из внешней памяти.

Стратегия управления памятью в соответствии с рабочим множеством стремится к тому, чтобы рабочее множество программы было в первичной памяти. Решение о том, следует ли добавить новый процесс к набору активных процессов (т. е. об увеличении степени мультипрограммирования), должно базироваться на том, имеется ли в основной памяти достаточно свободного места, чтобы разместить рабочее множество страниц этого процесса. Подобное решение, особенно в случае впервые инициируемых процессов, зачастую принимается эвристически, поскольку система не может заранее знать, каким окажется рабочее множество данного процесса.

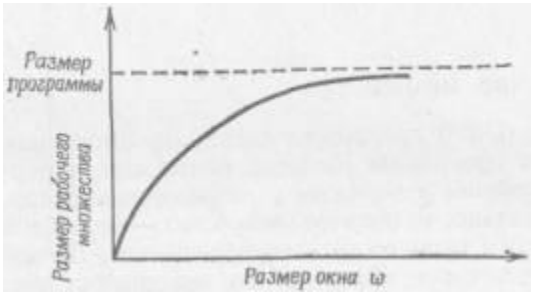


Рис. 9.5 Размер рабочего множества как функция размера окна.

Рабочее множество страниц процесса W(t, w) в момент времени t есть набор страниц, к которым этот процесс обращается в течение интервала времени процесса от t—w до t (см. рис. 9.4). Время процесса — это время, в течение которого процесс имеет в своем распоряжении центральный процессор. Переменная w называется размером окна рабочего множества, причем выбор величины этого окна играет решающую роль с точки зрения эффективности стратегии управления памятью по рабочему множеству. На рис. 9.5 показано, как растет размер рабочего множества с увеличением размера окна w. Эта кривая следует из математического определения рабочего множества и не является результатом обработки эмпирически наблюдаемых размеров рабочих множеств. Реальное рабочее множество процесса — это множество страниц, которые должны находиться в первичной памяти, чтобы процесс мог эффективно выполняться.

Во время работы процесса его рабочие множества динамически меняются. Иногда к текущему рабочему множеству добавляются или из него удаляются некоторые страницы. Иногда происходят резкие изменения, например, когда процесс переходит к этапу, требующему совершенно нового рабочего множества. Таким образом, любые предположения относительно размера и содержимого начального рабочего множества процесса не обязательно будут справедливыми для последующих рабочих множеств данного процесса. В связи с этим существенно усложняется задача реализации четкой стратегии управления памятью по рабочим множествам.

На рис. 9.6 показано, как мог бы использовать первичную память процесс при управлении памятью по рабочим множествам. Вначале, поскольку процесс запрашивает страницы своего рабочего множества не все сразу, а по одной, он постепенно получает достаточный объем памяти для размещения текущего рабочего множества. Затем на какой-то период времени этот объем памяти стабилизируется, поскольку процесс интенсивно обращается к страницам

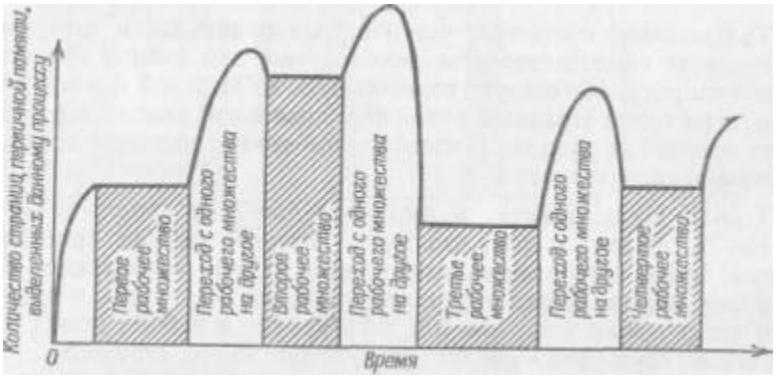


Рис. 9.6 Выделение основной памяти при управлении памятью на основе рабочих множеств.

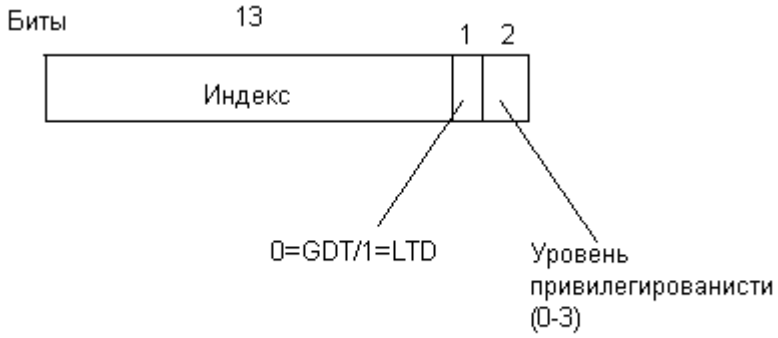
своего первого рабочего множества. Со временем процесс перейдет на следующее рабочее множество, что показывает кривая линия от первого ко второму рабочему множеству. Вначале эта кривая поднимается выше уровня первого рабочего набора, поскольку для процесса по его запросам быстро производится подкачка страниц нового рабочего множества. При этом система не может сразу определить, то ли процесс просто расширяет свое рабочее множество, то ли он переходит на новое. После того как процесс начнет

стабильно обращаться к страницам своего следующего рабочего множества, система видит, что в течение выбранного окна происходит обращение к меньшему количеству страниц, чем ранее, и сокращает объем памяти, выделяемой процессу, до числа страниц этого второго рабочего множества. Каждый раз, когда осуществляется переход с одного рабочего множества на другое, кривая линия поднимается, а затем спадает, показывая, каким образом система адаптируется к подобному переходу.

Рис. 9.6 иллюстрирует одну из серьезных трудностей, связанных с выбором стратегии управления памятью на основе рабочих множеств; проблема состоит в том, что рабочие множества меняются во времени, причем следующее рабочее множество процесса может существенно отличаться от предыдущего. Стратегия управления памятью должна обязательно учитывать этот факт, чтобы избегать перегрузок первичной памяти и возникающего вследствие этого трешинга.

232. Как найти локальную дескрипторную таблицу текущего процесса.

Основа виртуальной памяти системы Pentium состоит из двух таблиц: локальной таблицы дескрипторов (LDT) и глобальной таблицы дескрипторов (GDT). У каждой программы есть своя LDT, но GDT одна, ее совместно используют все программы в компьютере. Таблица LDT описывает сегменты, локальные для каждой программы, включая ее код, данные, стек и т.д., тогда как GDT несет информацию о системных сегментах, включая саму операционную систему.
Чтобы получить доступ к сегменту, программа системы Pentium сначала загружает селектор для этого сегмента в один из 6-ти сегментных регистров машины. Во время выполнения регистр CS содержит селектор для сегмента кода команды, а в регистре DS хранится селектор для сегмента данных. Каждый селектор представляет собой 16-разрядный номер.



Селектор в системе Pentium

Один из битов селектора несет информацию о том, является ли данный сегмент локальным или глобальным.

---

233. Дать определение файла, записи, сектора, кластера.

Кластер. В запоминающих устройствах — минимальный объем дискового пространства, который может быть выделен для размещения файла. Кластер состоит из одного или нескольких смежных секторов. Чем меньше размер кластера, тем более эффективно используется дисковая память. Кластеры также называют единичными блоками.

В компьютерных сетях — группа независимых компьютеров, работающих вместе в виде единой системы, предоставляющей клиентам общий набор служб. Кластер позволяет расширить как доступность служб, так и масштабируемость и управляемость их операционной системы.

Сектор — наименьший физический блок памяти на диске, определенной емкости(обычно 512 байт).

Запись — структура данных как совокупность областей (элементов), каждая со своим собственным именем и типом. Элементы записи могут быть доступны и как блок элементов, и по отдельности.

Файл — законченная именованная совокупность информации, набор данных, используемый программой, или документ, созданный пользователем. **Файл** — основной элемент хранения данных в компьютере; такая организация позволяет отличить один набор данных от другого. Файл является единым связным элементом, который пользователь может найти, изменить, удалить, сохранить или послать на устройство вывода.

---

234. Когда и зачем формируются управляющие таблицы ОС?

При управлении большими системами используется метод таблиц управления, который характеризуется тем, что вся информация необходимая для принятия решения концентрируется в таблицах. Самая первая таблица – это таблица векторов прерываний. Сведенья о всех ресурсах сведены к таблицам. В системе кроме того, что есть таблицы есть и таблица таблиц.

---

235. Какая программа и когда формирует область векторов прерываний.

В компьютерах семейства IBM PC, построенных на основе микропроцессора Intel 80x86 и использующих операционную систему MS-DOS, механизм прерываний реализуется как при помощи аппаратных, так и при помощи программных средств.

Микропроцессор поддерживает обработку 256 различных прерываний, каждое из которых идентифицируется номером в диапазоне 00h-FFh. Сегментные адреса обработчиков прерываний, называемые **векторами прерываний**, содержатся в **таблице векторов прерываний**, которая располагается в начале оперативной памяти (по адресу 0000:0000). Каждый вектор прерывания имеет размер 4 байта, таким образом, таблица векторов занимает 1024 байта. Для определения адреса обработчика любого прерывания нужно номер прерывания умножить на 4.

В защищенном режиме старшие модели семейства процессоров вместо таблицы векторов используют **таблицу дескрипторов прерываний,** сходную по своему назначению с таблицей векторов.

Операционная система не контролирует содержимое таблицы векторов, хотя и имеются средства для чтения и изменения векторов. При помощи функций MS-DOS значение векторов можно изменить, поместив нужный адрес непосредственно в таблицу векторов прерываний. В этом случае контроль за содержимым векторов и их соответствием размещению обработчиков в памяти полностью ложится на программиста, а последствия в случае ошибки могут быть самыми непредсказуемыми.

---

236. Проблемы многоуровневой памяти.

На характеристики памяти влияют два аспекта. Во-первых, кэш скрывает относительно низкую скорость памяти. После того как программа проработала некоторое время, кэш заполняется строками программы, увеличивая скорость. Однако когда операционная система переключается от одной программы к другой, кэш остается заполненным данными первой программы, а необходимые строки новой программы должны загружаться уже из физической памяти. Такая операция может стать главной причиной снижения производительности, если она происходит слишком часто.
Во-вторых, при переключении от одной программы к другой регистры управления памятью должны меняться. В реальных диспетчерах памяти должно перезагружаться, явно или динамически, большое количество регистров. В любом случае подобная операция занимает некоторое время.

---

237. Правила выбора размера страницы. Факторы, влияющие на выбор.

Пусть средний размер процесса равен S байт, а страница – P байт. Кроме того, предположим, что запись для каждой страницы требует E байт. Тогда приблизительное количество страниц, необходимое для процесса, равно S/P, что займет S*E/P байт для таблицы страниц. Потеря памяти в последней странице процесса вследствие внутренней фрагментации равна P/2. Таким образом, общие накладные расходы вследствие поддержки таблицы страниц и потери от внутренней фрагментации равны сумме этих двух составляющих:

$$\text{расход} = SE/P+P/2.$$

Первое слагаемое (размер таблицы страниц) увеличивается при уменьшении размера страниц. Второе слагаемое (внутренняя фрагментация) при увеличении размера страний возрастает. Оптимальный вариант должен находиться где-то посредине. Если взять первую производную по переменной P и приравнять ее к нулю, мы получим равенство:

$$- S*E/(P*P) + 1/2 = 0.$$

Из этого равенства мы можем получить формулу, дающую оптимальный размер страницы.В результате получим:

$$P = \text{корень из } (2*S*E).$$

При больших страницах может быть половина страницы пустой если в нее записан маленький фрагмент текста (внутренняя фрагментация). Если в оперативную память грузить такие страницы, то много ОП будет использоваться не эффективно.
С другой стороны, если программа будет слишком большой, то ей понадобится много маленьких страниц, что заберет много времени на их загрузку.

238. Назначение - Системы управления задачами.

Важнейшей частью операционной системы, непосредственно влияющей на функционирование вычислительной машины, является подсистема управления процессами. Процесс (или по-другому, задача) - абстракция, описывающая выполняющуюся программу. Для операционной системы процесс представляет собой единицу работы, заявку на потребление системных ресурсов. Подсистема управления процессами планирует выполнение процессов, то есть распределяет процессорное время между несколькими одновременно существующими в системе процессами, а также занимается созданием и уничтожением процессов, обеспечивает процессы необходимыми системными ресурсами, поддерживает взаимодействие между процессами.

239. Чем определяется насыщение системы прерываний.

Насыщение системы прерываний – это когда система обрабатывает прерывание источника и получает прерывание от этого же источника. От сюда: насыщение системы прерываний определяется количеством прерываний, вызванных от одного источника.

---

240. Показать принцип защиты памяти при сегментной организации.

При помощи атрибутов, записанных в строке таблицы страниц, можно организовать контроль доступа к конкретной странице и ее защиту. (например, биты защиты: 0 - read/write, 1 - read only ...)

---

241. Характеристика распределенной операционной системы чем она отличается от сетевой.

В сетевой операционной системе пользователи знают о существовании многочисленных компьютеров, могут регистрироваться на удаленных машинах и копировать файлы с одной машины на другую. Каждый компьютер работает под управлением локальной операционной системы и имеет своего собственного локального пользователя. Сетевые операционные системы несущественно отличаются от однопроцессорных операционных систем. Ясно, что они нуждаются в сетевом интерфейсном контроллере и специальном низкоуровневом программном обеспечении, поддерживающем работу контроллера, а также в программах, разрешающих пользователям удаленную регистрацию в системе и доступ к удаленным файлам.
Распределенная операционная система, напротив, представляется пользователям традиционной однопроцессорной системой, хотя она составлена из множества процессоров. При этом пользователи не должны беспокоиться о том, где работают их программы или расположены файлы; все это должно автоматически и эффективно обрабатываться самой ОС.

---

242. Назначение программы - главный планировщик.

Когда компьютер работает в многопрограммном режиме, на нем могут быть активными несколько процессов, пытающихся одновременно получить доступ к процессору. Эта ситуация возникает при двух и более процессов в состоянии готовности. Если доступен только один процессор, необходимо выбирать между процессами. Отвечающая за это часть операционной системы называется планировщиком.

---

243. Какие программы находятся в ядре операционной системы.

Ядро ОС содержит программы для реализации следующих функций:

- обработка прерываний;
- создание и уничтожение процессов;
- переключение процессов из состояния в состояние;
- диспетчеризацию заданий, процессов и ресурсов;
- приостановка и активизация процессов;
- синхронизация процессов;
- организация взаимодействия между процессами;
- манипулирование PCB;
- поддержка операций ввода/вывода;
- поддержка распределения и перераспределения памяти;
- поддержка работы файловой системы;
- поддержка механизма вызова— возврата при обращении к процедурам;
- поддержка определенных функций по ведению учета работы машины.

---

244. Влияние размера окна рабочего множества на эффективность работы системы.
Смотри вопрос 231

---

245. Влияние использования ассоциативной памяти на формирование исполнительного адреса.

В машинах преобразование адреса идёт на обратном уровне. В машинах используются двухуровневые кэши. Один из кэшей является ассоциативной памятью, в которой реализуется TLB-буфер (буфер быстрого преобразования адреса). Ассоциативная память ведёт обращение не по адресу, а по содержимому.

---

246. Каким образом система управления заданиями учитывает организацию памяти

---

247. **Почему используется двойная буферизация.**

Пользовательский процесс предоставляет буфер размером в n символов в пространстве пользователя, после чего выполняется чтение n символов. Процедура обработки прерываний помещает приходящие символы в буфер до тех пор, пока он не заполнится. Затем она активирует процесс пользователя. У такой схемы есть недостаток: что случиться, если в момент прибытии символа страница памяти, в которой расположен буфер, окажется выгруженной из физической памяти? Конечно, буфер можно зафиксировать в памяти; если слишком много процессов начнут фиксировать свои страницы в памяти, пул доступных страниц памяти уменьшиться, в результате чего снизится производительность (рис.1).
Следующий подход состоит в создании буфера, в который обработчик прерываний будет помещать поступающие символы, в ядре (рис.2). Когда этот буфер наполняется, достается страница с буфером пользователя, и содержимое буфера копируется туда за одну операцию. Такая схема намного эффективнее.
Однако даже при использовании этой схемы имеются определённые проблемы. Что случится с символами, прибывающими в тот момент, когда страница пользователя загружается с диска? Поскольку буфер полон , их некуда поместить. Решение проблемы состоит

в использовании второго буфера в ядре, в который помещаются символы после заполнения первого буфера до его освобождения (рис.3). При этом буферы как бы меняются местами, то есть первый буфер начинает играть роль запасного. Такая схема часто называется двойной буферизацией.

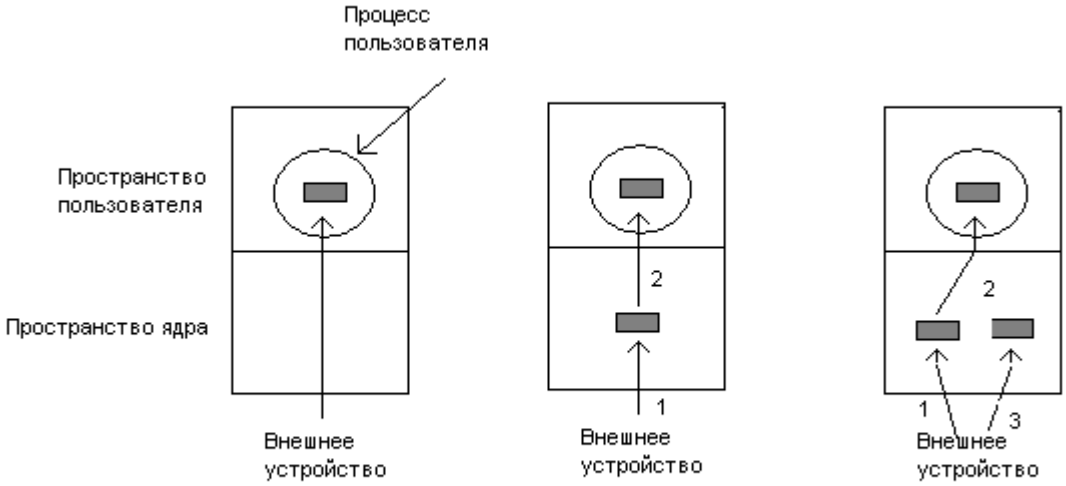


рис.1 рис.2 рис.3

---

248. Применение разделяемой памяти при синхронизации процессов (надо уточнить что он имеет в виду!)

Активизированные в системе процессы могут выполняться параллельно, если им для выполнения не требуется одинаковых ресурсов. Однако, это довольно редкая ситуация в системе. В общем случае несколько процессов могут выполняться параллельно лишь некоторое малое время. В остальное время они синхронизируются или взаимодействуют. **Взаимодействие** — это передача данных одного процесса другому. **Синхронизация** необходима для корректности передачи данных или для обращения к **общему ресурсу** (ОР).

Синхронизация *при передаче данных* заключается в следующем: пока процесс — передатчик не сформирует передаваемые данные и не перешлет их, процесс — приемник не должен выполнять никаких действий, т.е. должен ожидать пересылки данных, и тогда он может продолжить свое выполнение. Процесс — приемник может и сам создать процесс — передатчик, например, при возникновении необходимости ввода данных. При этом процесс — приемник блокируется и передает управление дочернему процессу, по окончании выполнения которого он может продолжить (возобновить) свое выполнение. При реализации такого вида синхронизации необходимо определить порядок предшествования во времени для некоторые точек трасс процессов и определить условие, разрешающее переход некоторых точек трасс на определенные процессы. Эти выделенные точки называются **точками синхронизации** (когда два выполнявшихся параллельно процесса подошли к точке, в которой необходимо определить, кто из них передатчик и кто приемник; либо точка фиксирует момент, когда создается и запускается дочерний процесс (передается управление от родительского).

Синхронизация *при обращении к ОР* необходима, чтобы исключить одновременный доступ к нему сразу нескольких процессов. Здесь возникает задача взаимного исключения, когда только один процесс может в данный момент "захватить" ОР, а остальные должны ожидать. **Критический участок (КУ)** — это часть процесса, где происходит обращение к ОР. Процесс, который в данный момент обладает ОР, находится в КУ. В период нахождения в КУ процесс является монополистом по отношению к ОР. Поэтому необходимо, чтобы процесс как можно быстрее проходил свой КУ и не блокировался в нем, иначе может сложиться ситуация, когда несколько (очевидно, наименее приоритетных) процессов будут бесконечно долго ожидать выделения ОР. При выходе процесса из КУ, вход туда может быть разрешен одному из ожидающих в очереди к ОР процессов. Это также своего рода синхронизация. Таким образом, процессы, обращающиеся к ОР могут выполняться лишь последовательно.

Примитивы синхронизации

Для синхронизации процессов используются примитивы. Некоторые из них (в основном, семафоры и мониторы) реализованы в ядре (для синхронизации ввода/вывода, переключения контекста (переход по прерыванию) и т.д.).

Примитивы синхронизации могут быть реализованы следующим образом:

2. Самые простые примитивы — **флажки** (примитивы взаимоисключения). Если флажок равен 0, т.е. условие ложно, то процесс не может войти в КУ, т.к. там уже находится другой процесс; если флажок равен 1 (условие истинно), то процесс входит в КУ, обнуляя флажок (если имеется очередь процессов к ОР, то в КУ входит наиболее приоритетный из них). На основе флажков реализованы алгоритмы синхронизации Деккера. Флажки — программная реализация решения задачи взаимного исключения. Это самый низкий уровень.
3. **Команда test_and_set** — аппаратное средство синхронизации (более высокий уровень). Это специальная команда, выполняющая вместе (неделимо) 3 действия:
   - чтение значения переменой;
   - запись ее значения в область сохранения;
   - установка нового значения этой переменной;

4. **Семафор** — некоторая (защищенная) переменная, значение которой можно считывать и изменять только при помощи специальных операций P и V. Преимущество по сравнению с test_and_set — разделение действий на отдельные.

Операция **P(S)**: проверяет значение семафора S; если S>0, то S:=S-1, иначе (S=0) ждать (по S) .

Операция **V(S)**: проверяет, есть ли процессы, приостановленные по данному семафору; если есть, то разрешить одному из них продолжить свое выполнение (войти в КУ); если нет, то S:=S+1.

Операция P должна стоять до входа в КУ , а операция V — после выхода из КУ.

Недостатки:

3) громоздкость — в каждом процессе каждый КУ должен окаймляться операциями P и V;
4) невозможность решения целого ряда задач с помощью таких примитивов.

Монитор — примитив высокого уровня. Здесь "забор" ставится вокруг ОР, что удобнее (чем семафоры), т.к. ресурсов в несколько раз меньше, чем процессов (их КУ) и ставить "заборы" вокруг КУ слишком громоздко. Можно сказать, что монитор — это некоторый программный модуль, в котором объединены ОР и подпрограммы для работы с ними. При обращении к ОР, т.е. соответствующей процедуре монитора для работы с ОР, осуществляется вход в монитор. В каждый конкретный момент времени может выполняться только одна процедура монитора — это и есть решение задачи взаимного исключения. Если процесс обратился к процедуре монитора, а другой процесс уже находится внутри монитора, то обратившийся процесс блокируется и переводится во внешнюю очередь процессов — блокированных по входу в монитор. Процесс, вошедший в монитор (т.е. выполняющий его процедуру), также может быть блокирован им, если не выполняется некоторое *условие* X для выполнения процесса. Условие X в мониторе — это некоторая переменная типа condition. С ней связывается некоторая внутренняя очередь процессов, блокированных по условию X. Внутренняя очередь приоритетнее внешней. Для блокирования процесса по переменной X и помещения его во внутреннюю очередь по этой переменной используется операция монитора WAIT(X). При выполнении операции SIGNAL(X) из очереди, связанной с переменной X, извлекается и разблокируется первый процесс. Если внутренняя очередь пуста, то в монитор разрешается войти первому процессу из внешней очереди.

---

249. Особенности файловой системы HPFS

Смотреть 56, 218

---

250. Методы повышения эффективности организации вычислительного процесса в ВС.

Смтореть вопрос 308.

---

251. Основные принципы организации вычислительного поцесса повышающие эффективность работы ВС

---

252. Факторы определяющие размер таблицы страниц при чистой страничной организации и организации по запросам.

Есть два фактора:

1. Таблица страниц может быть слишком большой.
2. Отображение должно быть быстрым.

---

253. Как решается задача защиты памяти в различных схемах организации памяти.

Проблема защиты и перераспределения памяти 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. Кроме того, адреса проверяются по отношению к предельному регистру для гарантии, что они не используются для адресации памяти в не текущий раздел.

---

254. Условие перехода процесса из активного состояния в заблокированное.

Процесс блокируется, поскольку с точки зрения логики он не может продолжать свою работу (обычно это связано с отсутствием входных данных, ожидаемых процессом). Также возможна ситуация, когда процесс, готовый и способный работать, останавливается, поскольку операционная система решила предоставить на время процессор другому процессу.

---

255. В чем идея технологии MEMORY CHENNEL.

Идея состоит в том, что каждый компьютер имеет организацию виртуальной памяти, в каждой машине определены страницы, отражение которых имеет место в других компьютерах. Управление страниц осуществляется через эти каналы. Также MEMORY CHENNEL является одним из механизмов реализации когерентности – это явное размещение с указанием разделяемых страниц, и неявное указание доступа к ним с использованием Load, Store.

---

256. Почему точек входа PCB меньше, чем количество задач.

Потому, те задачи системы, которые блокированы (ожидают ресурс…), могут быть выгружены на диск в целях экономии памяти.

257. Дать определение кластер.

Кластер - единица дискового пространства, которое ОС использует при работе с диском**.** При создании файла на диске место ему выделяется кластерами. Например, размер кластера может быть равен 1024 байт.

258. Средства меж процессного взаимодействия – сокеты. Их особенности.

Сокеты обеспечивают возможность взаимодействия между процессами на одном ПК или на разных ПК и обладают единым набором функций для работы с различными стеками. Сокет определяется № порта (с которым связано приложение) и IP-адресом локальной машины.

Типы сокетов:

1) надежный, орентирован на соединение байтовый поток.

2) надежный, орентирован на соединение поток пакетов.

3) ненадежная передача пакета.

1) и 2) гарантируют, что все байты будут доставлены в том порядке в котором были посланы путем создания виртуального канала между сокетами. Разница между 1) и 2) в том, что 2) сохраняет границы между пакетами (т.е. необходимо читать порциями из канала).

3) непредоставляет некаких гарантий, но обладает высокой скоростью передачи.

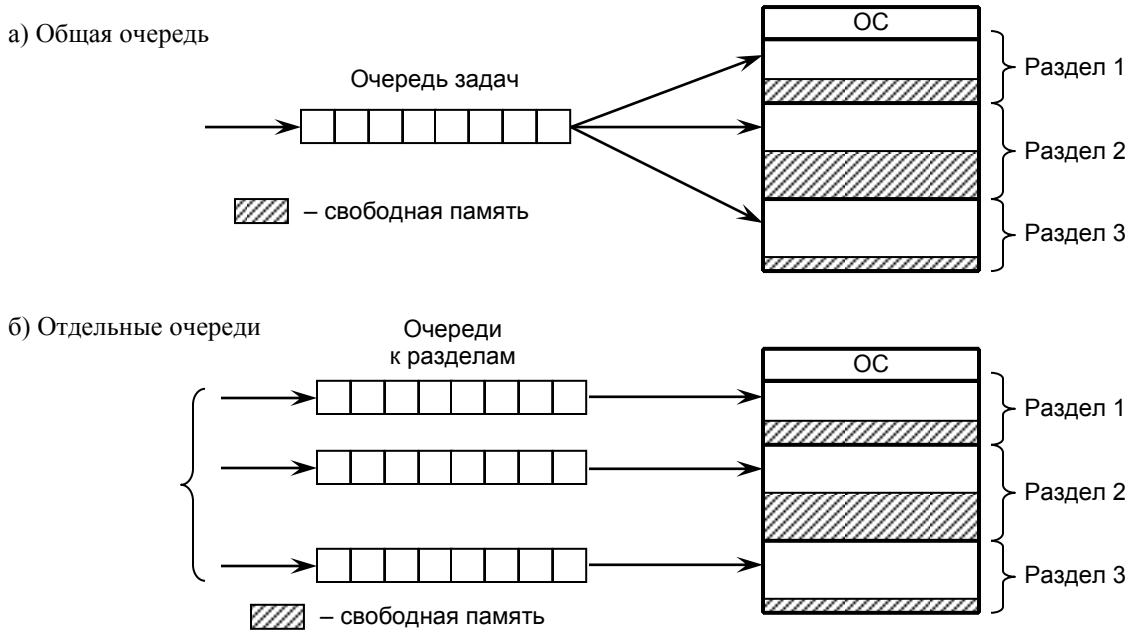
259. Недостатки распределения памяти MFT.

MFT (Распределение памяти фиксированными разделами).

Это простейший способ управления памятью. Память разбивается на несколько областей фиксированной величины, называемых *разделами.*

Это разбиение может быть выполнено вручную оператором во время старта системы или во ее установки. После этого границы разделов не изменяются.

Новый процесс, поступивший на выполнение, помещается либо в общую очередь, либо в очередь к некоторому разделу.



Распределение памяти фиксированными разделами

Система управления памяти решает следующие задачи:

- Сравнивает объем памяти, требуемый для нового процесса, с размерами свободных разделов и выбирает подходящий раздел.
- Осуществляет загрузку программы в один из разделов и настройку адресов.

Уже на этапе трансляции разработчик программы может задать раздел, в котором ее следует выполнять. Это позволяет сразу, без использования перемещающего загрузчика, получить машинный код, настроенный на конкретную область памяти.

Существенный недостаток — жесткость.

Уровень мультипрограммирования заранее ограничен числом разделов. Независимо от размера программы она будет занимать весь раздел, с другой стороны процесс, требующий несколько разделов, не может быть выполнен.

Этот простейший метод распределения памяти сейчас находит применение только в системах реального времени, благодаря детерминированности вычислительного процесса.

260. Правила смены страниц. Какие поля в TR используются для этого.

TR – регистр, который содержит указатель на TSS текущего процесса. В самом TSS используется поле CR3, которое содержит указатель на глобальную таблицу страниц процесса, в который происходит переход. При переходе, меняется таблица страниц, соответственно КЭШ TLB очищается.

261. Дать определение распределенная операционная система.(РОС)

В которой компоненты системы или ресурсы: процессоры, память, принтеры, графические станции, программы, данные и т.д., связанны вместе посредством сети, которая позволяет пользователям представлять ВС как единую вычислительную среду и иметь доступ к ее ресурсам. РОС предоставляется пользователям традиционной однопроцессорной системой. РОС обеспечивает свойство прозрачности, т.е. пользователи незнает, где работают их прогарммы и где физически расположены файлы. РОС осуществляет это все автоматически.

262. Перечислить методы доступа к данным.

263. Дать определение.Обрабатывающие программы операционной системы. Примеры.

Программы выполнения стандартных (в рамках ОС) функций, обработки исключительных ситуаций. Обработка – изменение информации с которой работает программа в составе ОС (драйверы, загрузчик).

264. В чем сложность для операционной системы в организации многопрограммного режима работы и какие задачи при этом решаются.

Организация защиты от взаимного влияния друг на друга на уровне оперативной и на уровне внешней памяти; разделение аппаратных и программных ресурсов; планирование (во времени, а в случае ПВС и в пространстве).

265. В каком виде компилятор передает информацию настраивающему загрузчику.

Непосредственно в настраивающем загрузчике каждый модуль может транслироваться отдельно. Чтобы передать сообщение редактору связей надо ему непосредственно указать, что надо транслировать. В каждом модуле в начале трансляции выделяются вектора перехода, внешние и внутренние.(Экспорт и Импорт процедур и ф-ций). Кроме того для выполнения настройки каждая команда отмечается битом переместимости. ОС выделяет и пользуетс глобально выделенной памятью, а загрузчик с локальной.

266. Перечислить уровни планирования в параллельной системе.

Семиуровневая модель – схема системы планирования с учетом (Маштабируемость, Разделяемость, Параллельность) 1. Предварительное (входное) планирование исходного потока заявок (задача фильтрации) 2. Структурный анализ взаимосвязи входного потока заявок по ресурсам с определением общих ресурсов (Анализ) 3. Структурный анализ заявок и определение возможности распараллеливания (Задача распараллеливания) 4. Адаптация распределения работ соответственно особенностям ВС (Задача адаптирования) 5. Составление плана – расписания выполнения взаимосвязанных процедур. Оптимизация плана по времени решения и кол-ву ресурсов (Задача Оптимизации) 6. Планирование потока задач претендующих на захват времени процессора на каждый прцессор – задача распределения. 7. Выделение процессорного времени, активизация задач. Перераспределение работ в ВС, при отказе оборудования (задача распределения – перераспределения).

267. Страничная фрагментауия. Как ее уменьшить.

Фрагментация – наличие большого числа несмежных участков свободной памяти маленького размера. Такого, что ни одна из вновь поступающих программ не может поместиться ни в одном из участков, хотя суммарный объем фрагментов может составить величину, превышающую требуемый объем памяти.

Борьба с фрагментацией – перемещение всех занятых участков в сторону старших или младших адресов, так, чтобы вся свободная память образовала свободную единую область.

Страничная фрагментация связана с большим размером страниц и называется внутренней фрагментацией. Уменьшение размера страниц приводит к увеличению размера таблицы страниц, которая может равна размеру самой памяти. Следовательно оптимальный размер страницы в байтах (p):

p=sqrt(2*s*e), где e-размер записи, s-размер процесса.

268. Виды фрагментации.

Фрагментации : внешняя и внутренняя.

Внутренняя ф. – возникновение свободных областей при загрузке программ. Свободные области проподают зря.

Внешняя ф. – ситуация когда в памяти существует свободные разделы, а программа больше самого большего свободного раздела. Она может поместится в два объединеных раздела.

269. Особенности применения сообщений при синхронизации процессов.

В модели сообщений процессы налагают на передаваемые данные некоторую структуру. Весь поток информации они разделяют на отдельные сообщения, вводя между данными, по крайней мере, границы сообщений. Примером границ сообщений являются точки между предложениями в сплошном тексте или границы абзаца. Кроме того, к передаваемой информации могут быть присоединены указания на то, кем конкретное сообщение было послано и для кого оно предназначено. Примером указания отправителя могут служить подписи под эпиграфами в книге. Все сообщения могут иметь одинаковый фиксированный размер или могут быть переменной длины. В вычислительных системах используются разнообразные средства связи для передачи сообщений: очереди сообщений, sockets (гнезда) и т. д. Каналы сообщений всегда имеют буфер конечной длины. Когда мы будем говорить о емкости буфера для сообщений, мы будем измерять ее в сообщениях. Для прямой и непрямой адресации достаточно двух примитивов, чтобы описать передачу сообщений по линии связи — ***send*** и ***receive***. В случае прямой адресации мы будем обозначать их так: ***send(P, message)***—послать сообщение message процессу **P**;
receive(Q, message) — получить сообщение message от процесса **Q**.В случае непрямой адресации мы будем обозначать их так:***send(A, message)*** — послать сообщение message в почтовый ящик **A**;
receive(A, message) — получить сообщение message из почтового ящика **A**.Примитивы ***send*** и ***receive*** уже имеют скрытый от наших глаз механизм взаимоисключения. Более того, в большинстве систем они уже имеют и скрытый механизм блокировки при чтении из пустого буфера и при записи в полностью заполненный буфер. Реализация решения задачи producer-consumer для таких примитивов становится неприлично тривиальной. Надо отметить, что, несмотря на простоту использования, передача сообщений в пределах одного компьютера происходит существенно медленнее, чем работа с семафорами и мониторами.

270. Сколько выходных очередей заданий формирует система.

В книге Симона написано, что 2: одна – готовые, другая - подготовленные

271. Чем определяется приоритет проблемных программ.

Смотреть 286

272. Чем определяется количество PCB.

Смотреть вопрос 230

273. Операции выполняемые в состоянии P3.
В P3 происходит выполнение процедуры сохранения, дешифрации прерывания, а также восстановления прерваной программы.

274. Принципы повышения эффективности работы ВС и как они реализованы в современных ВС.

275. Понятие окна рабочего множества.

Смотри вопрос 231

276. Дать определение сектор.
Сектор (или блок) – наименьшая адресуемая единица обмена данными с оперативной памятью.

Сектор имеет фиксированный для конкретной системы размер, выражающийся степенью двойки. Чаще всего – 512 байт. Поскольку дорожки разного радиуса имеют одинаковое число секторов, плотность записи на дорожках различна – больше к центру.

Для того, чтобы контроллер мог найти на диске нужный сектор, необходимо задать ему все составляющие адреса сектора; номер цилиндра, номер поверхности и номер сектора.

277. Правила замещиния страниц.

Стратегии выталкивания. Их цель — решить, какую страницу или сегмент следует удалить из первичной памяти, чтобы освободить место для помещения поступающей страницы или сегмента, если первичная память полностью занята.

В системах со страничной организацией все страничные кадры бывают, как правило, заняты. В этом случае программы управления памятью, входящие в операционную систему, должны решать, какую страницу следует удалить из первичной памяти, чтобы освободить место для поступающей страницы. Мы рассмотрим следующие стратегии выталкивания страниц.

- Принцип оптимальности.
- Выталкивание случайной страницы.
- Первой выталкивается первая пришедшая страница (FIFO).
- Первой выталкивается дольше всего не использовавшаяся страница (LRU).
- Первой выталкивается наименее часто использовавшаяся страница (LFU).
- Первой выталкивается не использовавшаяся в последнее время страница (NUR).
- Рабочее множество.

Принцип оптимальности (De70) говорит о том, что для обеспечения оптимальных скоростных характеристик и эффективного использования ресурсов следует заменять ту страницу, к которой в дальнейшем не будет новых обращений в течение наиболее длительного времени. Можно, конечно, продемонстрировать, что подобная стратегия действительно оптимальна, однако реализовать ее, естественно, нельзя, поскольку мы не умеем предсказывать будущее.

В связи с этим для обеспечения высоких скоростных характеристик и эффективного использования ресурсов мы попытаемся наиболее близко подойти к принципу оптимальности, применяя различные методы выталкивания страниц, приближающиеся к оптимальному

278. Дать определение. Управляющие программы операционной системы. Их функции.

Программы постоянно находящиеся в памяти (резидентные) организующие корректное выполнение процессов и функционирование всех устройств системы при решении задач. Составляют ядро ОС. Управление : Заданиями – слежение за прохождением заданий от входа до выхода на всех этапах его выполнения. Задачами – (Процессами) – слежение за всеми задачами активизированными в системе и процессами их выполнения на ресурсах. Памятью – решение задач эффективного u/ mem (internal) в соответствии с ее организацией.(Защита – согласование ин-фы в кешах) Данными – эффективное размещение и u/ данных на внешних носителях (проблема эффективности u/ процессора) Внешними ус-вами.

279. Определить условия перехода из состояния P3 в состояния P1 и P2.

Переход в P1 для выполнения прикладных программ (после восстановления в P3). Переход в P2 для обслуживания (обработки) прерывания (после дешифрации прерывания и сохранения прерваной программы).

280. Сущность проблемы когерентности данных.

Когерентность данных означает, что в любой момент времени для каждого элемента данных во всей памяти узла существует только одно его значение несмотря на то, что одновременно могут существовать несколько копий элемента данных, расположенных в разных видах памяти и обрабатываемых разными процессорами. Механизм когерентности должен следить за тем, чтобы операции с одним и тем же элементом данных выполнялись на разных процессорах последовательно, удаляя, в частности, устаревшие копии. Проще говоря, проблема когерентности памяти состоит в необходимости гарантировать, что всякое считывание элемента данных возвращает последнее по времени записанное в него значение.

281. Какая информация находится в старшей памяти.

Память 640Кб – 1Мб (384 Кбайта) адресного пространства называются старшей памятью (Upper Memory). Первоначально они были предназначены для размещения постоянных запоминающих устройств (ПЗУ). Практически под ПЗУ занята только часть адресов. Поэтому часть старшей памяти оказывается свободной. Эти участки используются для адресации к добавочным блокам оперативной памяти, которые носят название блоков старшей памяти (Upper Memory Blocks, UMB). Для поддержки старшей памяти используется драйвер HIMEM.SYS, который позволяет эффективно использовать UMB, загружая в них устанавливаемые драйверы устройств, а также резидентные программы. Загрузка системных программ в UMB освобождает от них стандартную память, увеличивая ее транзитную область.

282. Назвать нижние уровни управления внешними устройствами.

Основная идея организации программного обеспечения ввода-вывода состоит в разбиении его на несколько уровней, причем нижние уровни обеспечивают экранирование особенностей аппаратуры от верхних, а те, в свою очередь, обеспечивают удобный интерфейс для пользователей.

Ключевым принципом является независимость от устройств. Вид программы не должен зависеть от того, читает ли она данные с гибкого диска или с жесткого диска.

Очень близкой к идее независимости от устройств является идея единообразного именования, то есть для именования устройств должны быть приняты единые правила.

Другим важным вопросом для программного обеспечения ввода-вывода является обработка ошибок. Вообще говоря, ошибки следует обрабатывать как можно ближе к аппаратуре. Если контроллер обнаруживает ошибку чтения, то он должен попытаться ее скорректировать. Если же это ему не удается, то исправлением ошибок должен заняться драйвер устройства. Многие ошибки могут исчезать при повторных попытках выполнения операций ввода-вывода, например, ошибки, вызванные наличием пылинок на головках чтения или на диске. И только если нижний уровень не может справиться с ошибкой, он сообщает об ошибке верхнему уровню.

Еще один ключевой вопрос - это использование блокирующих (синхронных) и неблокирующих (асинхронных) передач. Большинство операций физического ввода-вывода выполняется асинхронно - процессор начинает передачу и переходит на другую работу, пока не наступает прерывание. Пользовательские программы намного легче писать, если операции ввода-вывода блокирующие - после команды READ программа автоматически приостанавливается до тех пор, пока данные не попадут в буфер программы. ОС выполняет операции ввода-вывода асинхронно, но представляет их для пользовательских программ в синхронной форме.

Последняя проблема состоит в том, что одни устройства являются разделяемыми, а другие - выделенными. Диски - это разделяемые устройства, так как одновременный доступ нескольких пользователей к диску не представляет собой проблему. Принтеры - это выделенные устройства, потому что нельзя смешивать строчки, печатаемые различными пользователями. Наличие выделенных устройств создает для операционной системы некоторые проблемы.

Для решения поставленных проблем целесообразно разделить программное обеспечение ввода-вывода на четыре слоя:
Обработка прерываний,
Драйверы устройств,
Независимый от устройств слой операционной системы,
Пользовательский слой программного обеспечения.

283. Фазы прерывания.

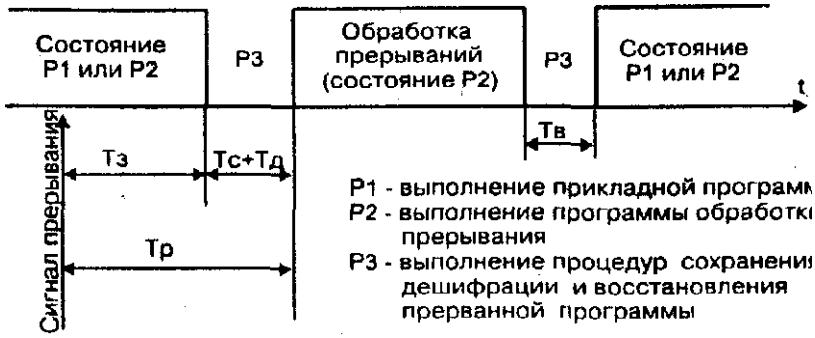


Рис. 2.20 Фазы прерывания

Тз — время задержки между моментом возникновения сигнала прерывания и прерыванием активного процесса. Оно зависит от принятого в системе (процессоре) способа обработки сигнала прерывания.
Тс — время сохранения необходимой информации. Зависит от количества сохраняемой информации при принятом способе обработки сигнала прерывания.
Тд — время дешифрации сигнала прерывания. Зависит от аппаратуры, дешифрирующей сигнал прерывания.
Тв — время восстановления прерванного процесса. Зависит от количества восстанавливаемой информации.

Время между возникновением сигнала прерывания и началом выполнения обработчика прерывания называется временем реакции системы на сигнал прерывания (Тр).

284. Способы повышения эффективности систем управления данными.
См. 194

285. Дать определение. Область сохранения. Где находится?

Смотреть вопрос 62.

286. В чем отличие определения приоритов проблемных и системных программ.

Пользовательские процессы имеют различные приоритеты в зависимости от порождающих их процессов и от времени выполнения на процессоре. Также необходимо следить за очередью блокированных процессов, с тем, чтобы среди них не было "бесконечно" ожидающих процессов. Вполне возможно, что событие, которое они ожидают, вообще может не произойти — тогда их надо вывести из системы. Также может оказаться, что блокированный процесс имеет слишком низкий приоритет и он может бесконечно долго находиться в режиме бесконечного откладывания в очереди блокированных процессов — тогда необходима система динамического, адаптивного изменения приоритетов.

287. Момент прерывания, чем он определяется?

Когда процессор получает сигнал прерывания, он должен прервать выполняемый в нем процесс. При этом возможны несколько вариантов реакции процессора на сигнал прерывания;

1. Прерывание возможно только после определенных команд, при этом необходимо сохранение минимального количества информации о состоянии системы.
2. Прерывание возможно после завершения любой команды процессора-
3. Прерывание возможно после завершения очередного такта процессора. При этом требуется сохранение большого объема информации. Большинство процессоров достроены таким образом, что сигнал прерывания проверяется ими только после завершения выполнения текущей команды выполняющегося процесса. Однако, в системе могут существовать прерывания, для которых невозможно ожидание завершения очередной команды, например, если возникает ошибка в самой команде.

288. Какие и в каких случаях применяются методы доступа.(вообще ничего не понятно)

-

289. Достоинства и недостатки файловой системы HPFS

Достоинства:

1.HPFS распределяет пространство, основываясь на физических 512-байтовых секторах, а не на кластерах, независимо от размера раздела. Система HPFS позволяет уменьшить и непроизводительные потери, так как в ней предусмотрено хранение до 300 байт расширенных атрибутов в F-узле файла, без захвата для этого дополнительного сектора.

2.Файловая система HPFS обеспечивает гораздо более низкий уровень фрагментации. Хотя избавиться полностью от нее не удается, снижение производительности, возникающее по этой причине, почти незаметно для пользователя.

3.Тот факт, что все пространство заранее распределено, также позволяет HPFS использовать специально оптимизированное программное обеспечение для более быстрой и эффективной работы с каталогами.

4. HPFS обладает повышенной отказоустойчивостью по сравнению с FAT.
В системе HPFS вместо таблицы размещения файлов применяется битовый массив, который содержит флаг, помечающий используемые секторы. Если область битового массива будет разрушена, пользователь этого не заметит, даже если это случится во время работы системы. F-узел файла также содержит информацию о размещении каждого файла. Поэтому область битового массива может быть восстановлена после поиска этой информации в F-узлах.

5. HPFS не налагает ограничений на максимальный размер файла.

290. Дать определение - Страничная фрагментация. Как уменьшить ее влияние.

При страничной организации памяти возникает проблема страничной фрагментации – процессу не может быть выделен кусок страницы, только сама страница полностью. За счет этого, если мы запрашиваем, например, 100 байт, то система выделит нам 4 кб. Теряется разница 4кб – 100б.

Уменьшить влияние страничной фрагментации можно размещая все данные программы вплотную (следуя друг за другом), т. е. выделяем место не под одно данное, а сразу под блок данных, а также используя разделяемые страницы – страницы, которые могут использоваться несколькими процессами.

291. Смысл алгоритма MESI.(as is conspect)

Алгоритм MESI (в простонародии алгоритм с обратной записью)является одним из алгоритмов, обеспечивающих когерентность (согласованность) памяти.

Small intro:

Механизмы ралиации когерентности могут быть явными и неявными для программиста.

Все архитектуры делятся на 4 категории:

5. Явное размещение и явный доступ (все операции над данными явны с точки зрения размещения и доступа. Команды Send() и Receive())
6. Неявное размещение и неявный доступ (доступ к данным прозрачен для программиста).Т.е. оперирование данными происходит на уровне команд читать/писать без явного указания адресов.
7. Неявное размещение (как в страничной организации) явный доступ. Используется разделяемое множество страниц на ВУ. При запросе страницы система автоматически обеспечивает согласование.
8. Явное размещение с указанием разделяемых(? Или раздела) страниц и неявный доступ с помощью команд Load() и Store().При этом используется технология **MEMORY CHANNEL**. Каждый компьютер имеет виртуальную память и разделяемые страницы, отображение этих страниц имеется во всех остальных компьютерах системы их удаление производится с использованием специальной команды. При этом используется специальная сетевая карта , обеспечивающая взаимодействие память-память.

MESI – один из алгоритмов *неявной* реализации когерентности. Идея его такова : Память сосредоточена.Подключение к ней идет через шину. Все транзакции также реализуются только через шину, поэтому есть возможность их отслеживания. Каждая строка в *кэшах* имеет следующие состояния :

5. M – модифицирована (операция R/W разрешена только в этом модуле)
6. E – монопольно копирована (операция R/W разрешена во всех модулях)
7. S – множественно копирована (операция R/W разрешена во всех модулях , где есть копия страницы)
8. I – запрещена к использованию.

Периодически по шине отправляются циклы опроса состояния строк.

292. Дать определение распределенная операционная система, чем она отличается от сетевой.

Существует два основных подхода к организации операционных систем для вычислительных комплексов, связанных в сеть, – это сетевые и распределенные операционные системы. Необходимо отметить, что терминология в этой области еще не устоялась. В одних работах все операционные системы, обеспечивающие функционирование компьютеров в сети, называются распределенными, а в других, наоборот, сетевыми. Мы придерживаемся той точки зрения, что сетевые и распределенные системы являются принципиально различными.

В сетевых операционных системах для того, чтобы задействовать ресурсы другого сетевого компьютера, пользователи должны знать о его наличии и уметь это сделать. Каждая машина в сети работает под управлением своей локальной операционной системы, отличающейся от операционной системы автономного компьютера наличием дополнительных сетевых средств (программной поддержкой для сетевых интерфейсных устройств и доступа к удаленным ресурсам), но эти дополнения существенно не меняют структуру операционной системы.

Распределенная система, напротив, внешне выглядит как обычная автономная система. Пользователь не знает и не должен знать, где его файлы хранятся, на локальной или удаленной машине, и где его программы выполняются. Он может вообще не знать, подключен ли его компьютер к сети. Внутреннее строение распределенной операционной системы имеет существенные отличия от автономных систем.

293. Структура очередей сообщений

Очереди сообщений являются одним из трех механизмов IPC (от англ.
Inter Process Communication -- межпроцессное взаимодействие). Другие
два -- это семафоры и разделяемая память. Очереди сообщений появились в UNIX system V release III и предназначались для асинхронной передачи сообщений между процессами.

В общих чертах обмен сообщениями выглядит примерно так: один процесс помещает сообщение в очередь посредством неких системных вызовов, а любой другой процесс может прочитать его оттуда, при условии, что и процесс-источник сообщения и процесс-приемник сообщения используют один и тот же ключ для получения доступа к очереди.Для каждой, вновь создаваемой очереди, в области ядра отводится пространство со следующей структурой:
Эта структура определена в файле bits/msg.h. Этот заголовочный файл
обязательно должен подключаться к вашим программам, использующим
очереди сообщений через подключение файла sys/msg.h.

```
/* Структура записи для одного сообщения в области ядра
Тип __time_t соответствует типу long int.
Все данные типы определены в заголовочном файле types.h*/
struct msqid_ds
{
struct ipc_perm msg_perm; /* структура описывает права доступа */
__time_t msg_stime;       /* время последней команды msgsnd (см. ниже) */
unsigned long int __unused1;
__time_t msg_rtime;       /* время последней команды msgrcv (см. ниже) */
unsigned long int __unused2;
__time_t msg_ctime;       /* время последнего изменения */
unsigned long int __unused3;
unsigned long int __msg_cbytes; /* текущее число байт в очереди */
msgqnum_t msg_qnum;      /* текущее число сообщений в очереди */
msglen_t msg_qbytes;     /* максимальный размер очереди в байтах */
__pid_t msg_lspid;       /* pid последнего процесса вызвавшего msgsnd() */
__pid_t msg_lrpid;       /* pid последнего процесса вызвавшего msgrcv() */
unsigned long int __unused4;
unsigned long int __unused5;
};
```

Первый элемент структуры - это ссылка на другую структуру, которая
имеет следующее определение в файле bits/ipc.h. подключение которого
производится через файл sys/ipc.h.

```
/* Структура используется для передачи информации о правах доступа в операциях
IPC. */
struct ipc_perm
{
__key_t __key;            /* Ключ. */
__uid_t uid;              /* UID владельца. */
__gid_t gid;              /* GID владельца. */
__uid_t cuid;             /* UID создателя. */
__gid_t cgid;             /* GID создателя. */
unsigned short int mode;    /* Права доступа. */
unsigned short int __pad1;
unsigned short int __seq;   /* Порядковый номер. */
unsigned short int __pad2;
unsigned long int __unused1;
unsigned long int __unused2;
};
```

Структура ipc_perm хранит UID, GID и права доступа к очереди.
Передача и прием сообщений

Для передачи и приема сообщений UNIX-подобные системы предоставляют
две функции: msgsnd() -- для передачи и msgrcv() -- для приема.
Определение обеих функций приведено ниже:

```
int msgsnd (int msqid, const void *msgp, size_t msgsz, int msgflg);
int msgrcv (int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg);
```

294. Зачем нужен интерлинг.

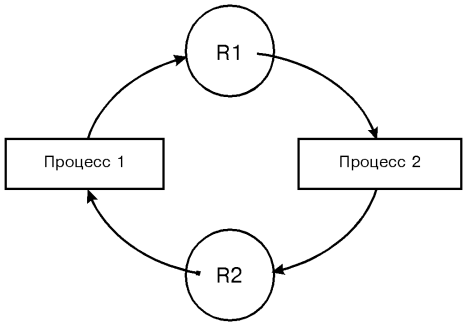
Интерлинг – перенумерация секторов отличная от физического расположения через 1,2 или 3 сектора для обеспечения времени подготовки контроллера к приему информации .

295. Правила выхода из тупика(as in book)

Small intro: **Тупик** – ситуация в системе из которой нет выхода. Бесконечное ожидание == тупику.

Тупик ***неизбежен***, если присутствует четыре условия его возникновения:

1. Условие *взаимного исключения*. Процесс обладает монопольным правом владения ресурсами во время всего существования процесса.
2. Условие *ожидания*. Процесс, обладая монопольным правом, пытается захватить новые ресурсы.
3. Условие *перераспределения*. Ресурс нельзя отнять до полного завершения процесса.
4. Условие: *круговая цепь ожидания*. Каждый из процессов цепочки требует дополнительный ресурс, который уже выделен процессу данной цепочки.



Тупики в системе спулинга

При вводе заданий используется режим **спулинга** — режим буферизации для выравнивания скоростей при вводе и считывании информации из буфера. Система, использующая режим спулинга, сильно подвержена тупикам.

В таких системах буфер может быть полностью заполнен до того, как данные из него станут поступать на печатающее устройство, а его размера недостаточно для буферизации всех данных вывода. При этом возникает тупиковая ситуация, т.к. размер буфера недостаточен для помещения всех поступивших данных, а принтер не сможет начать вывод. В данном случае единственным выходом является рестарт системы, но при этом теряется вся информация.

Можно выделить 4 стратегии борьбы с тупиками:

1. **Предотвращение тупика**. Требуется создать условия, исключающие возможность возникновения тупиковых ситуаций. Однако этот метод часто приводит к нерациональному использованию ресурсов.
2. **Обход тупика**. Этот метод предусматривает менее жесткие ограничения, чем первый и обеспечивает лучшее использование ресурсов. Метод учитывает возможность возникновения тупиков, и при увеличении вероятности тупиковой ситуации принимаются меры по обходу тупика.
3. **Обнаружение тупиков**. Требуется установить сам факт возникновения тупиковой ситуации, причем точно определить те процессы и ресурсы, которые в нее включены.
4. **Восстановление после тупиков**. Необходимо устранить тупиковую ситуацию, чтобы система смогла продолжить работу, а процессы, попавшие в тупиковую ситуацию, смогли завершиться и освободить занимаемые ими ресурсы. Восстановление — это серьезная проблема, так что в большинстве систем оно осуществляется путем полного выведения из решения одного или более процессов, попавших в тупиковую ситуацию. Затем выведенные процессы обычно перезапускаются с самого начала, так что вся или почти вся проделанная процессами работа теряется.

Предотвращение тупиков

Возникновение тупика невозможно, если нарушается одно из четырех необходимых условий возникновения тупиковых ситуаций. Первое условие при этом можно и не нарушать, т.е. считаем что процесс может обладать монопольным правом владения ресурсами. Нарушая остальные три условия получаем следующие три способа предотвращения тупиков:

I способ. Процесс запрашивает и получает все ресурсы сразу. Если процесс не имеет возможности получить все требуемые ресурсы сразу, т.е. некоторые из них на данный момент заняты, то он должен ожидать освобождения требуемых ресурсов. При этом он не должен удерживать за собой какие-либо ресурсы. Нарушается условие "ожидания дополнительных ресурсов" (2-е) и тупиковая ситуация невозможна. При этом способе имеем простой процессов, ожидающих выделения ресурсов, и работу вхолостую значительной части ресурсов. Можно прибегнуть к разделению программы на несколько программных шагов, работающих независимо друг от друга. При этом выделение ресурсов можно осуществлять для каждого шага программы, однако увеличиваются расходы на этапе проектирования прикладных программ.

Этот способ имеет два существенных недостатка:

1). Снижается эффективность работы системы;

2). Усиливается вероятность бесконечного откладывания для всех процессов.

II способ. Если процесс, удерживающий за собой определенные ресурсы, затребовал дополнительные и получил отказ в их получении, он должен освободить все ранее полученные ресурсы. При необходимости процесс должен будет запросить их снова вместе с дополнительными. Здесь нарушается условие "перераспределения" (3-е). При этом способе каждый процесс

может несколько раз запрашивать, получать и отдавать ресурсы системе. При этом система будет выполнять массу лишней работы — происходит деградация системы с точки зрения полезной работы. Недостатки этого способа:

1. Если процесс в течение некоторого времени удерживал ресурсы, а затем освободил их, он может потерять имевшуюся информацию.
2. Здесь также возможно бесконечное откладывание процессов — процесс запрашивает ресурсы, получает их, однако не может завершиться, т.к. требуются дополнительные ресурсы; далее он, не завершившись, отдает имеющиеся у него ресурсы системе; затем он снова запрашивает ресурсы и т.д.

Имеем бесконечную цепочку откладывания завершения процесса.

III способ. Стратегия линейности. Все ресурсы выстроены в порядке присвоенных приоритетов и захват новых ресурсов может быть выполнен только в порядке возрастания приоритетов. При этом нарушается условие "круговая цепь ожидания" (4-е).

Трудности и недостатки данного способа:

1. Поскольку запрос ресурсов осуществляется в порядке возрастания приоритетов, а они назначаются при установке машины, то в случае введения новых типов ресурсов может возникнуть необходимость переработки существующих программ и систем;
2. Приоритеты ресурсов должны отражать нормальный порядок, в котором большинство заданий используют ресурсы. Однако, если задание требует ресурсы не в этом порядке, то оно будет захватывать и удерживать некоторые ресурсы задолго до того, как появится необходимость их использования — теряется эффективность.
3. Способ не предоставляет удобства пользователям.

Обход тупиков

Алгоритм "банкира".

При использовании алгоритма "банкира" подразумевается, что :

6) системе заранее известно количество имеющихся ресурсов;
7) система знает, сколько ресурсов может максимально потребоваться процессу;
8) число пользователей (процессов), обращающихся к ресурсам, известно и постоянно;
9) система знает, сколько ресурсов занято в данный момент времени этими процессами (в общем и каждым из них);
10) процесс может потребовать ресурсов не больше, чем имеется в системе.

В алгоритме "банкира" введено понятие надежного состояния. Текущее состояние вычислительной машины называется **надежным**, если ОС может обеспечить всем текущим пользователям (процессам) завершение их заданий в течение конечного времени. Иначе состояние считается **ненадежным**. Надежное состояние — это состояние, при котором общая ситуация с ресурсами такова, что все пользователи имеют возможность со временем завершить свою работу. Ненадежное состояние может со временем привести к тупику.

Система удовлетворяет только те запросы, при которых ее состояние остается надежным, т.е. при запросе ресурса система определяет сможет ли она завершить хотя бы один процесс, после этого выделения.

Недостатки алгоритма банкира:

7) Если количество ресурсов большое, то становится достаточно сложно вычислить надежное состояние.
8) Достаточно сложно спланировать, какое количество ресурсов может понадобиться системе.
9) Число работающих пользователей (процессов) остается постоянным.
10) Пользователи должны заранее указывать максимальную потребность в ресурсах, однако, потребность может определяться динамически по мере выполнения процесса.
11) Пользователь всегда заказывает ресурсов больше, чем ему может потребоваться.
12) Алгоритм требует, чтобы процессы возвращали выделенные им ресурсы в течение некоторого конечного времени. Однако, для систем реального времени требуются более конкретные гарантии. Т.е. в системе должно быть задано максимальное время захвата всех ресурсов процессом (через это время ресурсы должны быть освобождены).

Обнаружение тупиков

Обнаружение тупика — это установление факта того, что возникла тупиковая ситуация и определение процессов и ресурсов, вовлеченных в тупиковую ситуацию. Для обнаружения тупиковой ситуации используют операцию **редукции графа**. Эта операция выполняется системой, когда есть подозрение, что какие-то процессы находятся в тупике. При этом определяется, какие процессы могут успешно завершиться (по графу) и освободить ресурсы. Если запросы ресурсов для некоторого процесса могут быть удовлетворены, то граф можно редуцировать на этот процесс. При этом процесс удаляется из графа, включая все дуги от него к требуемым ресурсам и от выделенных ресурсов к нему, т.е. эти ресурсы могут считаться свободными и выделяться другим процессам. Если граф можно редуцировать на все процессы, значит, тупиковой ситуации нет, а если этого сделать нельзя, то все "нередуцируемые" процессы образуют группу процессов, находящихся в тупиковой ситуации.

Восстановление после тупиков

Систему, оказавшуюся в тупике, необходимо вывести из него, нарушив одно или несколько необходимых условий его существования. При этом несколько процессов потеряют (полностью или частично) проделанную работу. Однако, это дешевле, чем оставлять систему в тупиковой ситуации. Возникают следующие сложности восстановления системы:

1) В первый момент может быть неочевидно, что система попала в тупиковую ситуацию.
2) В большинстве систем нет достаточно эффективных средств, позволяющих приостановить процесс на неопределенно долгое время, вывести его из системы и возобновить впоследствии. Некоторые процессы (например, процессы реального времени) должны работать непрерывно и не допускают приостановки и последующего восстановления.

3) Если в системе существуют эффективные средства приостановки/возобновления, то их использование требует значительных затрат машинного времени и внимания высококвалифицированного оператора.

Самый эффективный способ восстановления после тупиков — **механизм приостановки/возобновления**. Он позволяет кратковременно переводить процессы в состояние ожидания, а затем активизировать ожидающие процессы, причем без потери информации.
Часто при тупиковых ситуациях требуется перезапуск (рестарт) системы. При этом существует **перезапуск сначала** (вся полученная до рестарта информация теряется) или **с контрольной точки** (теряется только информация после контрольной точки). Система не определяет контрольных точек — их должен предусмотреть программист (для конкретного процесса).
Вот такой ниибаццо ответ.

296. Дать определение программы с точки зрения ОС.(as in conspect)

Small intro:
Задание – внешняя единица работы системы (описывается на специальном языке). Задание загружается в систему только тогда , когда система имеет свободные ресурсы и преобразуется в *задачу*.
Задача – внутренняя единица работы системы,для которой система выделила ресурсы кроме процессорного времени.Задача фиксируется в системе если ей выделены системные ресурсы (блок управления процессом).
Процесс – траектория процессора в адресном пространстве.С точки зрения ОС программы и данные – ресурс, выделенный ОС.
Программа – ресурс задачи.В БУП – ссылка на начало программы.
Данные – ресурс ,обрабатываемый задачей.
Виды программ:

1. Повторноисполняемые
2. Повторнонеисполняемые
3. Реэнтерабельные (т.н. pure procedures) для каждого входа обрабатывается свой блок данных, тело процедуры отделено от данных.

По структуре программы делятся на :

1. Простой структуры
2. Оверлейные (т.н. планируемого перекрытия)
3. Динамически-последовательные
4. Динамически-параллельные

297. Способы решения задачи когерентности.

See small intro at 291.

298. Правила обхода тупика.

See at 295. Как я понял в 295-ом вопросе необходимо более общее трактование ,т.е там надо рассказать про **всё** , а в этом вопросе только про обход. Если не прав поправьте и тогда к нему(Симоненко В.П.) будет вопрос : «Что такое выход из тупика?».

299. Применение сигналов при синхронизации процессов(as in conspect)

Сигналы – простейший способ межпроцессного взаимодействия. Постепенно сигналы стали использоваться в системе прерываний , однако их число ограничено и они слабо информативны.
Отправка сигнала требует *системного вызова* , а доставка *перерывания*.
Сигнал(событие) должен иметь какой-либо № и ID.
События могут быть асинхронными (e.g. CTRL+C). либо быть уведомлениями об особых ситуациях.
Различают 2 фазы:

1.Генерация и отправка
2. Доставка и обработка.

300. Какая программа выполняет запись информации в область сохранения.

Смотреть вопрос 62

301. Дать определение терминов “программа” и “данные” с точки зрения ОС

See at 296

302. В какой момент времени ОС фиксирует “процесс” и какая программа при этом используется

В книге Симона написано, что процесс фиксируется в подготовленном состоянии. Соответственно, его фиксирует планировщик.

303. Дать определение. Инициализация системы. Функции.(as in conspect)

Small intro: ОС предназначена для удовлетворения нужд пользователей и должна содержать все необходимые пользователю модули или программы и поддерживать аппаратные средства.
Инициализация – процесс структурной организации взаимосвязанных модулей,из которых состоит ядро ОС.
Различают инициализацию *ядра* ,которая происходит по умолчанию и системы ,которая происходит по файлам инициализации.

304. Приемы повышения эффективности работы с внешними носителями

Существует 2 подхода к повышению эффективности работы с внешними носителями:

1.Оптимизация физического обращения к диску (перемещения головок,…)
2.Оптимизация логического обращения (RAID)

В первом случае алгоритмы делятся на 2 вида.
Когда текущая дорожка неизвестна используются следующие алгоритмы:

1.Random access
2.FIFO
3.LIFO

Если текущая дорожка известна то используются:

1.SSTF(стратегия наименьшего времени обслуживания) , еще до ввода-вывода определяется текущее перемещение головки. Из очереди заявок выбирается заявка с ближайшим перемещением.
Недостаток: при частом обращении некоторые заявки долго ожидают.
2.SCAN – головка перемещается в одном направлении, по пути обслуживая заявки.Все вновь поступившие заявки включаются в обслуживание.Т.е происходит обратный проход.
3.C-SCAN – SCAN без обратного прохода.
4.NstopSCAN. Используются 2 очереди. В начале сканируется очередь, она фиксируется и новые заявки поступают в дополнительную очередь. При скачке назад очереди перезаписываются.
5. По приоритету.

При втором подходе(RAID).RAID-массив – избыточный массив независимых дисков ,рассматриваемых ОС как один диск.Обеспечивается либо параллельный , либо независимый доступ.

RAID0 – чисто параллельный доступ .Избыточность не используется.

Disk1	Disk2	Disk3	Disk4
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

RAID1 – «зеркало».

Disk1	Disk2	Disk3	Disk4
0	1	0	1
2	3	2	3
…	…	…	…

RAID2 – второй диск хранит код Хемминга(исправляет 1 ошибку и обнаруживает 2) первого диска.

Disk1	Disk2	Disk3	Disk4
0	F(0)		

RAID3 - Используется четность с чередующимися битами.
RAID4 – Используется технология независимых дисков.
RAID5 – Защита чередующимися блоками.

Disk1	Disk2	Disk3	Disk4	Disk5
0	1	2	3	F(0-3)
4	5	6	F(4-7)	7
8	9	F(8-11)	10	11
12	13	14	F(12-15)	15

RAID6

Disk1	Disk2	Disk3	Disk4	Disk5	Disk6
0	1	2	3	F(0-2)	F(0-3)
4	5	6	F(4-6)	F(4-7)	7

305. Программа согласования файловой системы. В каких случаях она применяется.

Это scandisk, fsck. Применяется, если не вышли из системы, не сохранив КЭШи файловой системы.

Для FAT: Составляем таблицу занятых кластеров (по дескрипторам файлов) и таблицу свободных кластеров (по FAT). Если кластер одновременно и «занят» и «свободен», то помечаем его как занятый. Если на один кластер ссылается два файла, то делается дубль кластера.

Такие системы, как JFS, XFS, Ext3 не нуждаются в таком восстановлении, поскольку записывают все выполняемые ими действия в «журнал».

306. Как решается задача согласованности данных.

See at 297 , как я понял согласованность и когерентность это одно и то же.

307. Способы защиты памяти при страничной и сегментной организации памяти.

Страничная:

Защита страничной памяти основана на контроле уровня доступа к каждой странице, возможны следующие уровни доступа:

только чтение
и чтение и запись
только выполнение

В этом случае каждая страница снабжается трехбитным кодом уровня доступа. При трансформации логического адреса в физический сравнивается значение кода разрешенного уровня доступа с фактически требуемым. При их несовпадении работа программы прерывается.

При таком распределении с каждой математической страницей отождествляется специальный признак, который указывает на возможность или запрет использования соответствующей страницы при выполнении той или иной программы. Эти признаки хранятся в специальном регистре, который называют регистром защиты математических страниц. Состояние этого регистра в любой момент времени определяет доступность математических страниц. При этом защита памяти на уровне математических страниц существенно упрощается. Однако при записи программ в физических адресах механизм защиты памяти на уровне математических страниц уже не работает. В этой связи наряду с защитой памяти в поле математических адресов используют и защиту физических адресов.

При защите физических адресов в условиях страничного распределения памяти наиболее часто применяется защита памяти с использованием ключей, т. е. так называемая защита по ключам. В соответствии с принципом защиты по ключам каждой странице оперативной памяти присваивается ключ памяти. Все множество ключей хранится в так называемой памяти ключей, которая защищена от доступа рабочих программ. Запись в память ключей может производиться только супервизором. Каждой программе, размещенной в определенной странице оперативной памяти, присваивается соответствующий ключ защиты (ключ программы). Ключ защиты указывается в слове состояния программы (ССП). В процессе выполнения программы из кода адреса выделяется номер страницы, который рассматривается как входной адрес памяти ключей. Выбираемый по данному адресу ключ памяти сравнивается с ключом защиты данной программы,который является составной частью слова состояния программы. При установлении логической равнозначности обращение к оперативной памяти разрешается. В противном случае осуществляется прерывание по защите памяти. Исключением является случай, когда код ключа защиты равен нулю, что определяет свободный доступ к любой странице оперативной памяти. Код ключа защиты супервизора всегда равен нулю. В ЕС ЭВМ при защите физических адресов оперативная память рассматривается разделенной на блоки объемом в 2048 байтов (т. е. каждый блок составляет половину страницы). Память ключей состоит из 8-разрядных записей, каждые из которых включают в себя 4-разрядный ключ защиты, разряд выборки, разряд обращения, разряд изменения и разряд четности.

Если разряд выборки установлен в единичное состояние, то блок памяти с соответствующим ключом памяти защищен как от выборки, так и от записи.

Разряд обращения устанавливается в единицу при любом обращении к данному блоку. Просмотр состояния этих разрядов позволяет определить неиспользуемые страницы с целью их удаления из оперативной памяти. При этом для определения неиспользуемой страницы необходимо просмотреть разряды обращений двух блоков памяти, составляющих данную страницу.

Разряд изменения устанавливается в единицу только тогда, когда в страницу вносятся изменения. Если страница не изменяется, во внешней памяти имеется ее копия и ее следует вывести из оперативной памяти (она является кандидатом на удаление), то нет необходимости выполнять специальную программу для вывода ее во внешнюю память, а соответствующие блоки оперативной памяти можно сразу использовать. Если же страница претерпела изменения, то необходимо переписать данную страницу во внешнюю память и только после этого использовать освободившийся блок оперативной памяти.

Сегментная:

Каждый сегмент имеет имя, размер и другие параметры (уровень привилегий, разрешенные виды обращений, флаги присутствия).

Каждый сегмент – линейная последовательность адресов, начинающаяся с 0. Максимальный размер сегмента определяется разрядностью процессора (при 32-разрядной адресации это 2^{32} байт или 4 Гбайт). Размер сегмента может меняться динамически (например, сегмент стека). В элементе таблицы сегментов помимо физического адреса начала сегмента обычно содержится и длина сегмента. Если размер смещения в виртуальном адресе выходит за пределы размера сегмента, возникает исключительная ситуация. Логический адрес – упорядоченная пара v=(s,d), номер сегмента и смещение внутри сегмента. В системах, где сегменты поддерживаются аппаратно, эти параметры обычно хранятся в таблице дескрипторов сегментов, а программа обращается к этим дескрипторам по номерам-селекторам. При этом в контекст каждого процесса входит набор сегментных регистров, содержащих селекторы текущих сегментов кода, стека, данных и т. д. и определяющих, какие сегменты будут использоваться при разных видах обращений к памяти. Это позволяет процессору уже на аппаратном уровне определять допустимость обращений к памяти, упрощая реализацию защиты информации от повреждения и несанкционированного доступа.

Каждый сегмент описывается дескриптором сегмента.

ОС строит для каждого исполняемого процесса соответствующую таблицу дескрипторов сегментов и при размещении каждого из сегментов в ОП или внешней памяти в дескрипторе отмечает его текущее местоположение (бит присутствия). Дескриптор содержит поле адреса, с которого сегмент начинается и поле длины сегмента. Благодаря этому можно осуществлять контроль

1) размещения сегментов без наложения друг на друга
2) обращается ли код исполняющейся задачи за пределы текущего сегмента.

В дескрипторе содержатся также данные о правах доступа к сегменту (запрет на модификацию, можно ли его предоставлять другой задаче), защита.

308. Как повысить эффективность работы ВС с общей памятью.

В шинно-оринтированных сетях процессоры соединяются с общей памятью посредством единого канала данных, называемого шина. Такая система очень проста для реализации. Рис. 1.17(а) иллюстрирует обычную шинную архитектуру. При обращении процессора к общей памяти на определенных линиях шины выставляются соответствующие сигналы.

Главным недостатком данного способа является то, что при увеличении числа процессоров, связанных с общей шиной, весьма значительно увеличивается время ожидания каждым процессором доступа к памяти. Данный способ пригоден для небольшого числа процессоров.

Один из способов решения данной проблемы заключается в обеспечении каждого процессора локальной кэш-памятью, как показано на рис.1.17(б).

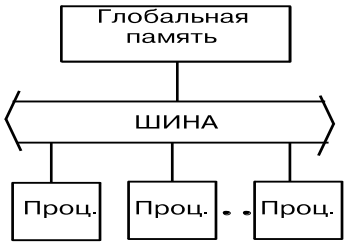


Рис.1.17.а Шинно-ориентированые сети

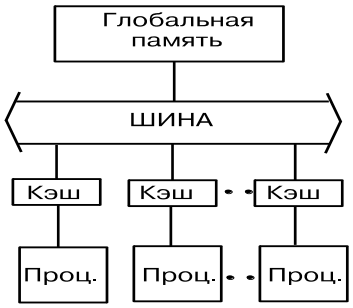


Рис. 1.17.б Кэш шинно-ориентированые сети

Использование памяти в шинно-ориентированных сетях с FLTM топологией отличается от FITM ВС. При обычных вычислениях, когда ссылка делается на какой либо фрагмент памяти, следующая ссылка вероятней всего будет сделана на фрагмент памяти, следующий за этим фрагментом. Этот вывод сделан на основе свойств "временной" и "пространственной" локальности теории рабочих множеств. Следовательно, весь этот фрагмент может быть помещен в локальную кэш-память процессорного элемента ВС и всю дальнейшую обработку данных он может вести не обращаясь к общей памяти. При этом существенно сокращается время работы процессора с общей памятью. В случае кэш-промаха (т.е. когда в локальной памяти не оказалось требуемого слова) процессору необходимо еще раз обратиться к общей памяти.

309. Каким образом и когда фиксируется приоритет процессов.

Small intro: При управлении процессами, как основной функции ОС ,можно выстроить следующую иерархию:
Задача – процесс – поток – нить.
Ресурсы в системе выделяются процессу ,а нити и потоки используют эти ресурсы.
Процессы могут быть созданы :

1. при инициализации ОС (для работы самой ОС т.е планировщиков,прерываний,…)
2. Другими процессами.

Процесс может пребывать в 4-х состояниях:

1.Подготовленном – задача зафиксирована в системе ,но ресурсы не выделены.
2.Готовом – как только выделены ресурсы задача переходит в готовое состояние – система выделяет ей РСВ (БУП – блок управления процессом) в нем среди прочих полей есть приоритет.

310. Особенности хранения свободного места в HPFS

Для определения свободен сектор или занят HPFS использует битмапы в которых каждый бит соответствует одному сектору. Если бит содержит 1 то это означает, что сектор занят, иначе он свободен. Если бы на весь диск был бы только один битмап то для его подкачки приходилось бы перемещать головки чтения/записи в среднем через половину диска. Чтобы избежать этого HPFS разбивает диск на "полосы" (Bands) длиной по 8 мегабайт и хранит битмапы свободных секторов в начале или конце каждой полосы. При этом битмапы соседних полос располагаются рядом:

```
+--------- 16MB --------+    *** - Use/Free sector bitmap.
!                       !
+--!-----------+----------!--+--------------+--------------+
!***  Полоса 0 ! Полоса 1  ***!***  Полоса 2 ! Полоса 3  ***!
+--------------+--------------+--------------+--------------+
0MB            8MB            16MB           24MB           32MB
```

Из этого следует что расстояние между двумя битмапами равно 16MB. Размер полосы (8MB) может быть изменен в следующих версиях HPFS т.к. на него нет прямых завязок. HPFS определяет размер полосы при чтении управляющих блоков с диска во время выполнения операции FSHelperAttach.

Сейчас размер битмапа равен 2K. (8MB/512/8 = 2K).

Полоса находящаяся в центре диска используется для хранения каталогов. Эта полоса называется Directory Band. Однако если она будет полностью заполнена HPFS начнет располагать каталоги файлов в других полосах.

311. Отличие функций программ системного ввода/вывода от режима спулинга.

Режим **спулинга** — режим буферизации для выравнивания скоростей при вводе и считывании информации из буфера. Система, использующая режим спулинга, сильно подвержена тупикам.

В таких системах буфер может быть полностью заполнен до того, как данные из него станут поступать на печатающее устройство, а его размера недостаточно для буферизации всех данных вывода. При этом возникает тупиковая ситуация, т.к. размер буфера недостаточен для помещения всех поступивших данных, а принтер не сможет начать вывод. В данном случае единственным выходом является рестарт системы, но при этом теряется вся информация.

312. Почему нельзя зациклить состояние P3 (очень спорный ответ)

Т.к это состояние т.н «переход на другую программную последовательность» , процесс переходит в него только по прерыванию либо напрямую из пользовательской программы. Зацикливание не возможно так как при обработке прерывания происходит его дешифрация ,которая является гарантировано атомарной операцией.