**1. Определение ОС. Классификация ОС. Требования, предъявляемые к ОС**

ОС – обеспечивает уровень абстракции между ПО и аппаратным обеспеч. Избавляет программиста от необходимости непосредственного взаимодействия в аппаратурой. ОС обеспечивает управление всеми ресурсами компьютера, их выделение различным процессам (очередь печати). Часть ОС постоянно запущенная на компьют и реализующая основный функции в том числе о взаимодействии с аппаратным обеспечением называется ядро ОС. ОС обеспечивает управление доступом к данным различных пользователей, управления памятью и другими ресурсами с разграничением доступа к ним различными процессами.

Классификация ОС:

1. однопользовательский, многопользовательский – обеспечивает защиту данных пользователя друг от друга.
2. однозадачное и многозадачное – обеспечивают разграничение доступа между задачами.
3. по целям использования:

*1 .система пакетной обработки* – это ОС направленная на решение вычислительных задач, не требует интерактивности в процессе работы могут не иметь многозадачности и многопользовательского режима. Критерий эффективности таких систем является пропускная способность те количество задач в единицу времени.

2. *система разделения времени (интерактивные системы)* – обеспечивает работу одной системы и множества задач пользователя одновременно. Каждый из них имеет иллюзию монопольного владения системой. Время отклика – промежуток времени между действием пользователя и результатом этого действия.

3. *система реального времени* – системы управляющие технологическими ресурсами. Критерий эффективности является необходимость выполнения задач к конкретному моменту времени. Системы делятся на 2 типа: жесткие и гибкие.

В жестких системах превышение времени категорически недопустимо (управление поездом)

В гибких превышение допустимо, не нежелательно.

4. смешанные системы.

Требования к ОС.

1. реализация возложенных функций.
2. расширяемость. Система должна иметь возможность легкого дополнения новыми функциями и это не должна нарушать целостность системы.
3. переносимость. Система должна легко переноситься с одной аппаратной платформы на другую.
4. надежность и отказоустойчивость. Система должна быть защищена от внутренних и внешних ошибок, ее действия должны быть предсказуемы. Приложения не должны наносить вред системе.
5. совместимость. ОС должна иметь средства для выполнения приложений написанных для других ОС. Совместимость бывает 2 типов.

- совместимость двоичных кодов – откомпилированное приложения должны запускаться на различных системах.

- совместимость исходных кодов – программы должны компилироваться и запускаться на различных системах. Для этого программы пишутся на стандартизованных языках.

5. безопасность. Система должна содержать средства защиты данных.

6. производительность. Система должна макс эффективно использовать аппаратное обеспечение компьют.

**2. Основные принципы построения ОС.**

1) Частотный – основан на выделении данных по частоте и создания условий для их быстрого выполнения путем хранения в ОЗУ.

2) Принцип модульности. Понимаетcя модульность ОС, прикладных программ и аппаратных средств. Под модулем понимают функциональный элемент системы, законченный и выполняемый в ее рамках и средства сопряжения с подобными элементами этой или другой системы.

3) Принцип функциональной избирательности. В ОС выбираются модули, которые хранятся в памяти и составляют ядро ОС.

4) Принцип генерации – настраивание ОС на конфигурацию выч. системы и запросы пользователя. 5) Принцип функциональной избыточности – возможность выполнения одних и тех же операций различными способами

6) Принцип по умолчанию – хранение в системе некоторые базовые значения, если они не устанавливаются

7) Принцип перемещаемости – разные модули в разных ситуациях могут загружаться по разным адресам

8) Принцип защиты – ограждение программ и данных от нежелательных взаимных влияний

9) Прницип независимости программ от внешних устройств.

10) Принцип открытости – позволяет вводить в ОС новые модули.

**3. Понятие процесса и ресурса. Иерархия процессов**.

*Процесс*(или по-другому, задача) - абстракция, описывающая выполняющуюся программу. Для операционной системы процесс представляет собой единицу работы, заявку на потребление системных ресурсов.

Большинство ресурсов распределяются в режиме разделения времени. *Ресурс* выделяется задаче на заранее известный интервал времени, если программа не успеет за этот интервал закончить работу, то он изымается и отдается следующей задаче, а текущая задача – в конец очереди к этому ресурсу.

В современных ОС: тенденция развития: многозадачность, графический интерфейс, распределенные сетевые ресурсы.

**Классификация процессов.**

1.По принадлежности к центральному процессору: внутренние и внешние. По внешним признакам. Трасса процесса – последовательность и длительность пребывания процесса в каждом из своих состояний интервал существования процесса, время между порождением и завершением процесса. Процессы реального времени, интерактивные и пакетные.

2.По генеалогическому признаку – пораждающие и поражденные.

3. По динамическим признакам: последовательные интервалы сущ процессов не пересекаются.

Если на рассматриваемом интервале времени сущ-ют одновременно 2 процесса, они параллельн. Если хотя бы одна точка, когда один проц сущ, а второй нет, то такие проц комбинированные.

4.по принадлежности к ОС: системные и пользовательские(прикладные)

5. по результативности: эквивалентные процессы – которые на одинаковых входных данных выдают одинаковые выходные данные. Причем работают АО одной или разным программам. Трассы эквивалентных процессов в общем случае не совпадают.

Тождественные – эквивалентные процессы работающие по одной и той же программе, но с различными трассами.

Равные – тождественные процессы с одинаковыми трассами. Все остальные процессы различны.

6.по связности.

Два процесса являются взаимосвязанными, если между ними поддерживаются, с помощью системы управления процессами, функциональные пространственно-временные управляющие информационные или другие связи, иначе процессы изолированные. Изолированные со слабой связью могут влиять друг на друга. Если два взаимосвяз процесса используют вместе некоторые ресурсы, но информационно не связаны, те не обмениваются инф, то эти процессы информационно независимы. При наличии информацион связей, процессы взаимодействуют.

**4. Диаграмма состояния процесса.**

При использовании такой абстракции все, что выполняется в вычислительных системах (не только программы пользователей, но и, возможно, определенные части операционных систем), организовано как набор процессов. Понятно, что реально на однопроцессорной компьютерной системе в каждый момент времени может исполняться только один процесс. Для мультипрограммных вычислительных систем псевдопараллельная обработка нескольких процессов достигается с помощью переключения процессора с одного процесса на другой. Пока один процесс выполняется, остальные ждут своей очереди. каждый процесс может находиться как минимум в двух состояниях: процесс исполняется и процесс не исполняется.

Процесс, находящийся в состоянии процесс исполняется, через некоторое время может быть завершен операционной системой или приостановлен и снова переведен в состояние процесс не исполняется. Приостановка процесса происходит по двум причинам: для его дальнейшей работы потребовалось какое-либо событие (например, завершение операции ввода-вывода) или истек временной интервал, отведенный операционной системой для работы данного процесса. После этого операционная система по определенному алгоритму выбирает для исполнения один из процессов, находящихся в состоянии процесс не исполняется, и переводит его в состояние процесс исполняется. Новый процесс, появляющийся в системе, первоначально помещается в состояние процесс не исполняется.

Это очень грубая модель, она не учитывает, в частности, то, что процесс, выбранный для исполнения, может все еще ждать события, из-за которого он был приостановлен, и реально к выполнению не готов. Для того чтобы избежать такой ситуации, разобьем состояние процесс не исполняется на два новых состояния: готовность и ожидание

Всякий новый процесс, появляющийся в *системе*, попадает в состояние готовность. Операционная система, пользуясь каким-либо алгоритмом планирования, выбирает один из готовых процессов и переводит его в состояние исполнение. В состоянии исполнение происходит непосредственное выполнение программного кода процесса. Выйти из этого состояния процесс может по трем причинам:

* операционная система прекращает его деятельность;
* он не может продолжать свою работу, пока не произойдет некоторое событие, и операционная система переводит его в состояние ожидание;
* в результате возникновения прерывания в вычислительной системе (например, прерывания от таймера по истечении предусмотренного времени выполнения) его возвращают в состояние готовность.

Из состояния ожидание процесс попадает в состояние готовность после того, как ожидаемое событие произошло, и он снова может быть выбран для исполнения.

Наша новая модель хорошо описывает поведение процессов во время их существования, но она не акцентирует внимания на появлении процесса в системе и его исчезновении. Для полноты картины нам необходимо ввести еще два состояния процессов: рождение и закончил исполнение

Теперь для появления в вычислительной системе процесс должен пройти через состояние рождение. При рождении процесс получает в свое распоряжение адресное пространство, в которое загружается программный код процесса; ему выделяются стек и системные ресурсы; устанавливается начальное значение программного счетчика этого процесса и т. д. Родившийся процесс переводится в состояние готовность. При завершении своей деятельности процесс из состояния исполнение попадает в состояние закончил исполнение

**5. Контекст и дескриптор процесса.**

**Контекст процесса –** информация, необходимая для возобновления выполнения процесса, включающая в себя состояние регистров, программного счетчика, режим работы CPU, указатели на открытые файлы…

**Дескриптор** –ОС для каждого процесса создает дескриптор вида D=(I,V,T,P,R),\_\_где I-идентификатор(имя) процесса, V – определ. виртуальную машину процесса, T – включает состояние процесса, Р –указывает на родительские и дочерние процессы, R- приоритет процесса

Чтобы ОС могла управлять процессами, на каждый процесс заводится дескриптор( описатель), он содержит идентификатор процесса. PID (process identification): приоритет процесса, состояние процесса, информация о ресурсах процесса, указатель на родителя и потомки, контекст задачи (состояние процесса, если он прервется). Эти дескрипторы хранятся в виде таблицы процессов.

**6. Классификация процессов.**

1.По принадлежности к центральному процессору: внутренние и внешние. По внешним признакам. Трасса процесса – последовательность и длительность пребывания процесса в каждом из своих состояний интервал существования процесса, время между порождением и завершением процесса. Процессы реального времени, интерактивные и пакетные.

2.По генеалогическому признаку – пораждающие и поражденные.

3. По динамическим признакам: последовательные интервалы сущ процессов не пересекаются.

Если на рассматриваемом интервале времени сущ-ют одновременно 2 процесса, они параллельн. Если хотя бы одна точка, когда один проц сущ, а второй нет, то такие проц комбинированные.

4.по принадлежности к ОС: системные и пользовательские(прикладные)

5. по результативности: эквивалентные процессы – которые на одинаковых входных данных выдают одинаковые выходные данные. Причем работают АО одной или разным программам. Трассы эквивалентных процессов в общем случае не совпадают.

Тождественные – эквивалентные процессы работающие по одной и той же программе, но с различными трассами.

Равные – тождественные процессы с одинаковыми трассами. Все остальные процессы различны.

6.по связности.

Два процесса являются взаимосвязанными, если между ними поддерживаются, с помощью системы управления процессами, функциональные пространственно-временные управляющие информационные или другие связи, иначе процессы изолированные. Изолированные со слабой связью могут влиять друг на друга. Если два взаимосвяз процесса используют вместе некоторые ресурсы, но информационно не связаны, те не обмениваются инф, то эти процессы информационно независимы. При наличии информацион связей, процессы взаимодействуют.

**7 . Понятие потока. Способы реализации. Планирование.**

Каждому процессу соответствует одиночный управляющий поток, представляющий из себя поток исполняемых команд. При запуске многопотокового процесса в системе с одним процессоров потоки работают поочередно. Процессор быстро переключается между потоками, создавая впечатления параллельной работы потоков.

|  |  |
| --- | --- |
| Элементы процесса | Элементы потока |
| Адресное пространство | Счетчик команд |
| Глобальные переменные | Регистры |
| Открытые файлы | Стек |
| Дочерние процессы | Состояние потока |
| Необработан. аварийные сигналы |  |

Аргументы в пользу использования потоков.1) выполнение большинством программных приложений действий, которые могут временно блокироваться. 2) легкость создания потоков и их уничтожение, поскольку с потоком не связано никаких ресурсов. 3) Возможность совмещения во времени операций ввода – вывода или вычислений, когда в них имеется одновременная потребность. 4) концепция потоков удобна в системах с несколькими процессорами.

Способы реализации потоков

3 способа реализации пакета потоков

1)В пространстве пользователя.

Процесс

Поток

Таблица потоков

Ядро Таблица процессов

Ядро о потоках ничего не знает и управляет обычными однопоточными процессами. Каждому процессу необходима собственная таблица потоков для отслеживания потоков в процессе

Достоинства*:* Переключение потоков происходит быстрее, чем между процессами. Каждый процесс имеет свой алгоритм планирования

Недостатки*.* Проблема реализации блокир. системных запросов. Пока не обработается один поток не будет запущен другой

2) В пространстве ядра.

Процесс

Поток

Ядро Таблица процессов

Таблица потоков

Достоинства Все запросы, которые блокируют поток реализуются как системные. Когда поток блокируется, ядро запускает поток из этого или другого процесса. Недостатки Все системные запросы требуют значит. больше времени так как связаны с переключением потоков

3)Смешанная

Потоки

Ядро Поток ядра

Ядро знает только о потоке своего уровня. Некоторые из этих потоков могут содержать по несколько потоков пользовательского уровня. Потоки пользователя используют потоки ядра по очереди

Планирование потоков

При планировании потоков определяется поддерживание потоков на уровне пользователя или ядра. Рассмотрим процесс А с потоками (А1, А2… ) и процесс В с потоками (В1, В2…). При планировании потоков на уровне пользователя, ядро не знает о существовании потоков, выполняет обычное планирование. Выбирает процесс А и предоставляет ему квант времени. Планировщик потоков внутри процесса А выбирает поток А1. Если квант времени для процесса А (50 нс) > чем квант времени для потока А1 (5нс), то планироваться будет выполнение (А1, А2 …)

При реализации потоков на уровне ядра, ядро не обязательно принимает во внимание какой поток принадлежит какому процессу. По истечении кванта времени управление передается потоку другого процесса (А1, В1, А2, В2….).

**8. Планирование и диспетчеризация процессов. Стратегии планирования.**

Есть три уровня планирования процесса:

1.долгосрочное планирование или планирование задач – определяет в каком порядке задачи поступают в систему и будут пораждены соответствующими процессами. Этим занимается планировщик.

2. краткосрочное планирование или диспетчеризация. Этим занимается диспетчер, но иногда называют планировщик. Диспетчер выбирает из очереди готовых процессов тот, который должен поступить на выполнение. Диспетчеризация выполняется в следующем случае:

1. если система использует вытесняющее планирование и выполняющийся процесс перешел из состояния выполнения в любое другое состояние, те закончился его квант времени.

2. если выполняющийся процесс завершился.

3. если использ невытесняющ планиров. При невытесняющ – только когда заканчивается выполнение процесса или же он переходит в состояние ожидания. Краткосрочное выполняется часто.

3. среднесрочное планирование – тесно связано с управлением памяти (виртуальной). Среднесроч планир отвечает за выбор процессов которые при необходимости можно скинуть во вторичную память (жесткий диск) причем эти процессы должна быть максимально долго не нужны системе. Механизм называется swapping.

Стратегии диспетчеризации.

Требования:

1. главное требование справедливость выделения времени всем процессам.

2. ограниченного ожидания. Процесс не должен ожидать своей очереди бесконечно.

3. минимизация полного времени оборотов – время от поступления задачи в систему, до момента когда покинет систему.

Минимизация времени ожидания в очереди waiting time.

Вытесняющее и не вытесняющее планирование.

Приоритетное планирование. Выбор процесса из очереди может осущ с помощью приоритета. Приоритет – число характеризующее степень процесса привилегированности перед другими процессами. Системн проц более привелег чем пользовательский. Приоритеты бывают абсолютные и относительные. В системе с абсолютным приоритетом из очереди выбирается более приоритетный процесс. При поступлении в систему более приор процесс

**9. Планирование в системах пакетной обработки данных. Дисциплины FCFS, SJN, SRN.**

FCFS - First-Come, First-Served - первым пришел, первым обслужен. Процессы обслуживаются в строгом порядке своего поступления в систему. Не вытесняющий алгоритм. Достоинство: простота. Недостатки: большая зависимость эффективности работы вычислительной системы от порядка поступления процесса. В случает если длительные процессы поступили раньше, то коротких проц нужно слишком долго ждать. Следовательно увеличивается полное время оборота, а также время ожидания.

SJN - Shortest Job Next — следующим выполняется самое короткое задание. На выполнение поступает процесс которому необходимо наименьшее время выполнения. За счет этого минимизируется полное время оборота и время ожидания. Не вытесняющий алгоритм. Достоинства: очень хороший алгоритм. Недостатки: не применяется ибо нереален.

SRN – Shortest Remain Next –Вытесняющая модификация предыдущего алгоритма. Время выполнения представляет собой абсолютный приоритет. При поступлении в систему нового процесса сравнивается время, которое необходимо новому процессу и оставшееся время выполнения текущего проц. Выполняться будет тот процесс, у которого значение меньше. Алгоритм мочень хорош и столь же нереален.

**10. Планирование в интерактивных системах. Дисциплина RR (круговое планирование), дисциплины приоритетного планирования.**

*RR(Round Robin)*- карусель. Каждому процессу выделяется квант времени. В случае если за выделен квант времени процесс завершается ,следующий процесс не простаивает. Ему выдел след квант врем. Размер кванта времени – главные вопрос. Если квант времени большой, то короткие процессы успевают завершиться и алгоритм выражается в FCFS. Если квант времени слишком маленький, то существенно возрастают накладные расходы между процессами. Может использоваться приоритетное планирование в зависимости от приоритета , после выполнения процесс может становиться не в конец очереди. Алгоритм активно используется в современных ОС.

**11. Планирование в системах реального времени**

, где m – кол-во событий, Ti- время которое уходит на обработку событий. Pi- периодичность, с которой поступает событие.

Для того, чтобы планировать однородные процессы (время на обработку и периодичность у всех процессов равны) можно использовать циклической планирование. Но для реальных систем это не подходит, потому, что процессы неоднородны. Чтобы планировать реальные процессы, планировщик должен знать частоту с которой должен работать каждый процесс. Первый алгоритм планирования. Статический RMS (Rate Monotonic Scheduling) – процессы должны бытии удовлетворять следующ условиям: процесс должен был быть завершен за время его периода; один проц не должен зависеть от другого; каждому процессу требуется одинаковое процессорное время на каждом интервале. У непериодичн процессов нет жестких сроков. Требования проц происходят мгновенно. Недостаток: он не учитывает момент поступления следующего события.

Динамический алгоритм планирования EDF(Earliest Deadline First) – учитывает ближайший момент поступления события. Приоритет каждого процесса выставляется в соответствии с тем, сколько времени осталось дедлайна. Таким образом процессы с течением времени изменяются. Способен планировать сильно загруженную систему в отличии от статистического алгоритма.

**12. Системные вызовы. Схема 12. Системные вызовы. Схема обработки системных вызовов**

Системный вызов (С.В.) - обращение прикладной программы к ядру ОС для выполнения какой-либо операции

*Требования к реализации С.В.:*

1. обеспечить переключение в привилегированный режим
2. обладать высокой скоростью вызова процедур ОС
3. обеспечивать единообразное обращение к системным вызовам для аппаратных платформ на которых работает ось
4. допускать легкое расширение вызова набора системных вызовов
5. обеспечить контроль со стороны ОС

При любом системном вызове выполняется программное прерывание int80h. Передаётся номер системного вызова, затем параметры и вызывается обработчик данного прерывания – диспетчер С.В., который сохраняет содержимое в системном стеке, проверяет попадает ли запрошенный номер вызова в диапазон, по номеру находят соответствующие процедуры и передает ей управление

*Типы С.В.:*

1. синхронное –процесс приостанавливается и ждет завершения выполнения системного вызова
2. асинхронное – процесс продолжает выполняться II с действием системного вызова

fork –создание нового процесса

exec – подмена текущего процесса другим исполняемым файлом

open, close, read, write – работа с файлами

*Схема обработки системных вызовов*:



Текущий процесс shell делает вызов fork порождая вторую копию shell, которая вызывает exec, указывая в качестве параметра имя используемого файла, образ которого необходимо загрузить в память. Утилита ls начинает выполняться, по завершении работы ls созданный процесс исчезает (exit). Системный вызов wait позволяет заблокировать выполнение процесса, пока кто-нибудь из потомков не завершит работу.

**13. Назначение и классы прерываний. Механизм обработки прерываний. Учет приоритета прерываний**

Прерывание – прекращение выполнения текущей команды или последовательности команд для обработки некоторого события специальными программами с последующим возвратом к выполнению вызванной программы

Mov ax,1

Mov bx,2

Jre x

Прерывания происходят в произвольной точке

В зависимости от источника прерывания делятся на:

1) внешние

2) внутренние

3) программные

Внешние или аппаратные обрабатываются драйвером соответсв. внешних устройств. Внутренние – спец.модулями ядра, а программные – процедурами ОС , обслужив. системные вызовы.

**Внешние прерывания** – могут возникать в результате действий пользователя или в результате поступления сигналов от аппар.устр-в. Поэтому внешние прер-я называют аппаратными, отражая тот факт, что возникает прерывание вследствии подачи нек-й аппаратурой электр.сигнала. Сигнал подается на специальный вход прерываний процессора

*Является асинхронным по отношению к потоку команд выполняемой программы*. Бывают прерывания:

А) маскируемые (можно сделать так, что прерывание происходить не будет)

Б) немаскируемые

**Внутренние прерывания (исключения) –** происходят внутри процессора в процессе выполнения программы: синхронно выполнению программы при появлении аварийной ситуации в ходе выполнения некоторой инструкции программы. *Происходят синхронно выполнению текущих потоков команд*

**Программные прерывания –** они не являются истинными прерываниями. Возникают при выполнении особой команды процессора, выполнение к-й имитирует прерывание, т.е. переход на новую последовательность инструкций.

**Механизм обработки прерываний -** два основных способа реализации прерываний:

1)векторный

2)опрашиваемый.

В обоих случаях предоставляется приоритет, а в случае векторного – адреса обработчика прерываний.

Вектор прерываний – электр. сигнал, доставляемый на соответствующие коды процессора, несущих в себе информацию об определенном, закрепленном на данном у-ве номере, который идентифицирует соответствующий обработчик прерываний

**Векторные механизмы –** каждому уст-ву назначается вектор прерываний. Вектор прерыв., передаваемый в процессор предств.собой целое число 0..255, указывающее на 1 из 256 программ обработки прерываний. Вектор прерываний бывает фиксир., конфиг., программн.

**Опрашиваемые** – есть уровни приоритета прерываний, например IRQ, может быть связано несколько устройств и неск-о обр.прерываний. При появлении на шине прерываний с д.уровнем процессов опрашивает все устр-ва , к-е связаны с данным уровнем прерываний до тех пор пока разработчик не подтвердит, что это прерывание.

***Учет приоритета прерываний***

При использовании схемы с относительными приоритетами, при одновременном использовании запросов прерываний из разных классов выбирается запрос, имеющий высший приоритет. После выполнения его обработчика выполняется обработчик 2-го прерывания

Решение о выборе какой запрос обрабатывать принимается только в момент завершения обработки прерывания. В случае абсолютных приоритетов при поступлении более приоритетного прерывания во время работы обработчика менее приоритетного текущий обработчик прерывается и обрабатывается на вновь поступившем

**14. Варианты межпроцессного взаимодействия**

Причины взаимодействия: 1. повышение скорости работы; 2. совместное использование данных; 3. модульная конструкция системы; 4.удобство пользователя (процесс текстового редактора).

Взаимодействующие (кооперативные). Категория средств взаимодействия: 1. сигнальные – передается минимальное количество информ вплоть до 1 бита. Степень воздействия сигнала на поведение процесса минимальна. Все зависит от того, знает ли процесс как интерпретировать сигнал. Неправильная реакция на сигнал или игнорирование могут привести к трагическим последствиям. Сигналы вызывают прерывание выполнения задачи и выполнения заранее предусмотрен действий. Два типа сигнала: 1. синхронный – чаще всего приходят от системы прерывания проц блокируемых аппаратурой, например деление на 0.

2. Асинхронный – могут поступать в любой момент.

В ОС набор сигналов определен для каких-то дополнительных сигналов. Необходимо определить как их обрабатывать. Канальные средства. Процессы взаимодействуют через линии связи предоставляемые ОС. Канальные средства могут обладать или не обладать буфером. 1. буфер нулевой емкости или отсутствующ. Никакая информация не сохраняется на линии связи. Процесс посылающий информ должен дождаться пока принимающая сторона не получит информацию и только тогда продолжит свое выполнение.. 2. буфер ограниченной емкости может хранить не более n единиц информации. 3. буфер неограниченной емкости. Теоретически возможет практически нет.

**15. Синхронизация параллельных процессов. Критические ресурсы.**

Атомарная операция – это неделимая операция в рассматриваемом контексте.

P:a b c R(P).

Q:d e f W(P)

a b c d e f

Детерминированный набор активностей – когда всякий раз при псевлоисполнении для одного и того же набора вх.данных получаются одинаковые вых.данные.

Его можно безбоязненно выполнять в режиме разделения времени. Существует так называемые условия Бернстайна, к-е определяют насколько набор активностей детерминирован. R(P)и W(P) – наборы вх.и вых данных прог-мы или активности P.

P: х=u\*w

Y=a+u

W(P)={x,y}

R(P)= {a,u,w}

Условия:

1.для двух активностей P и Q: W(P)(знак пересечения – перевернутая буква U) W(Q)=0 - непересекаются

W(P)(тот же знак) R(Q)=0 – непер. наборы вых.данных

R(P)(тот же знак)W(Q)=0 – не должны пересекаться вых.д.активностей.

Если выполняется 1-3 то P и Q детерминированы.

Про дет.набор команд говорят, что они находятся в состоянии гонки либо состязания.Один из вариантов решения проблемы – вариант взаимоисключения. Каждый процесс, обращающийся к разделению ресурсам исключает для всех других процессов возможность одновремменого общения с этими ресурсами, если это может привести к недетерм.их поведению наборов процессов.

**Критическая секция –** часть программы исполнение которой может привести к воникновению гонки для определ.набора программ. Она подразумевает наличие след.действий:вход в секцию, сама секция, выход из секции. Вход - это запрет др.процессам работы данной кр.секции, постановка в ожидание. Выход – освобождение данного ресурса, отсылка сообщений , об освобождении секций.

While(some condition)

{ entry section

Critical section

Exit section

Remainder section}

Критические ресурсы.

Под критическим разумеется некоторый программный либо аппаратный ресурс, который в каждый момент времени может использоваться одним и только одним процессом, потоком или прерыванием. Правильное обращение с критическими ресурсами является основой грамотного, добротного и надежного программного обеспечения. Поэтому все программные компоненты, вызываемые несколькими различными потоками, должны рассматриваться с точки зрения отнесения к критическим ресурсам. Для этого будем использовать характеристику, обратную критичности, которая называется безопасностью по отношению к тем или иным событиям в программной системе. Для управляющих программ обычно требуется максимальный уровень безопасности компонент – по отношению к сигналам, то есть любым асинхронно возникающим в системе событиям: аппаратным прерываниям, переключениям контекста выполнения (потока) и другим.

Определим два уровня собственной сигналобезопасности программ.

1. Безусловная сигналобезопасность. Обеспечивается повторно-входимыми программными компонентами, обращение к которым может происходить в любой момент времени без каких-либо конфликтов. Такие программы должны выполняться независимо от любой другой программы, включая самою себя. Это достигается несколькими способами:

- Программа оперирует исключительно локальными данными, которые размещаются в стеке. Такой метод используется при реализации функций, допускающих рекурсивный вызов. Следует отметить, что стандарт языка программирования С поддерживает рекурсивное обращение к любым функциям.

- Программа блокирует все события (прерывания) в системе на время своего выполнения. Этот метод делает сигналобезопасной любую программу, однако должен использоваться лишь для минимальных сегментов кода. Блокировка всех событий может оказаться полезной для гарантированного обеспечения атомарности семафорных операций.

- Программа в состоянии восстановить в начальное состояние свои разделяемые ресурсы по завершении повторного вызова. Технология восстановления – на любителя, но весьма удобна для практического использования в частности, в управляющих программах, выполняемых без поддержки операционной системы и диспетчера потоков.

2. Транзакционная сигналобезопасность. Программные компоненты с транзакционной сигналобезопасностью могут вызываться в любой момент времени, но без повторной входимости. То есть выполнение программы должно полностью завершаться до ее очередного вызова. Транзакционно безопасная программа зависит только от самой себя, но независима от любой другой программы. Например, операции чтения и записи кольцевого буфера являются безусловно сигналобезопасными друг относительно друга, поскольку манипуляции головой и хвостом буфера независимы. Чтение буфера может быть прервано операцией записи и наоборот. Однако, ни чтение, ни запись сами по себе не являются повторно входимыми и должны рассматриваться как критические секции. Если программные компоненты зависят как от самих себя, так и от других программ, они не являются сигналобезопасными. Например, операции чтения и записи разделяемой памяти не являются сигналобезопасными: их наложение и на самих себя и друг на друга может привести к искажениям данных. Такие программы следует рассматривать как критические разделяемые ресурсы и применять внешние методы их защиты: семафоры или взаимные исключения (mutexes).

**16. Понятие взаимного исключения. Критический участок.**

В многопроцессной системе из-за взаимодействия процессов возможны 4 ситуации 1) взаимное исключение 2) синхронизация 3)взаимоблокировка 4)коммуникация**. Критические области**- ресурс, который допускает обслуживание пользователя за один раз наз. критическим. Если несколько процессов пользуются таким ресурсом, они должны синхронизировать свои действия. Место в программе, где процесс обращается к такому ресурсу наз. критическим участком. В каждый момент времени только один процесс может выполнять свой критический участок при работе с ресурсом. Для предотвращения этого используется взаимное исключение. Для реализации задач взаимного исключения необход. выполнение след. условий. 1) 2 процесса не должны одновременно находиться в критич. областях. 2)при запросе критич. ресурса несколькими процессами, только один получает разрешение. 3) процесс вне крит. Области не может блокир. Другие процессы. 4) процесс использует Крит область конечное время 5)любой процесс переходит в неактивное состояние только за пределами критич. области. 6) невозможна ситуация, когда процесс вечно ждет попадания в крит. область.

**17. Способы реализации взаимного исключения: запрещение прерываний, блокировка памяти, строгое чередование.**

1. Запрет прерываний

Выход процесса из состояния исполнения может быть по 2-ум причинам:

- блокировка из-за нехватки ресурсов

- конец кванта времени

Квант времени заканчивается по прерыванию таймера, поэтому, если запретить системе получать и обрабатывать прерывания, то процесс из состояний выполнения не выйдет, пока сам не разрешит прерывания.

Недостаток:

В случае зацикливания программы в КС или невыполнения эпилога, систему приходится перезагружать.

Этот алгоритм используется внутри ядра системы.

2. Блокировка памяти (переменная «Замок»)

shared int lock = 0;

while (1){

while (lock)

lock = 1;

КС

lock = 0;

}

Пролог не является атомарным, поэтому до того, как замок будет закрыт (lock = 1), управление может быть передано другому процессу, который тоже сможет войти в КС. Нарушается условие взаимоисключения.

3. Строгое чередование

Вводится переменная, которая идентифицирует номер процесса, который должен войти в КС.

shared int turn = 0;

while (1){

wile (turn != pid);

КС

turn=1-turn;

}

Взаимоисключение гарантируется, но алгоритм не удовлетворяет условию прогресса, потому что, если процессу, идентификатор которого равен turn не нужно в КС, то он и не выйдет из КС и не изменит значение turn, при этом другой процесс может бесконечно долго ожидать входа в КС.

**18. Способы реализации взаимоисключения: алгоритм Деккера, операция «Проверка и установка» (команда TSL)**

1. Алгоритм Деккера

Используется массив флагов и переменная кода.

shared int turn = 0;

while (1){

ready[pid] = 1;

turn = 1-pid;

while (ready[1-pid] && turn == 1-pid);

КС

ready[pid] = 0;

}

Процесс может войти в КС только, если флаг готовности другого процесса равен 0 и значение переменной turn = идентификатору текущего процесса.

Доказательство:

Представим, что 2 процесса одновременно находятся в КС. Значит, флаги готовности у них обоих равны 1, значит, условие цикла while у обоих процессов не выполнялось из-за второй части этого условия, т.е. для каждого процесса переменная turn равна идентификатору этого процесса, чего быть не может, что и требовалось доказать.

2. операция «Проверка и установка» (команда TSL).

Алгоритм Деккера, расширенный на n процессов.

int Test and Set (int \*target)

{int tmp = \*target; \*target = 1;

Return temp;}

shared int lock = 0;

while (1){

while (Test and Set(&lock));

КС

Lock = 0;}.

**19. Примитивы межпроцессного взаимодействия. Решение проблемы производителя и потребителя с использованием примитивов межпроцессорного взаимодействия**

Простейшими примитивами являются sleep и wakeup. Sleep - системный запрос, в результате которого процесс блокируется пока его не запустит другой процесс. Wakeup – в качестве параметра содержит процесс, который следует запустить. Проблема состоит в следующем 2 процесса используют буфер ограниченного размера. Один -производитель помещает данные в этот буфер. Трудность возникает когда производитель хочет поместить в буфер очередную информацию и обнаружив, что он полон и будет ожидать. Другой потребитель будет ожидать пока буфер не будет заполнен в случае если захочет забрать из буфера данные, а буфер пуст.

Для решения этой проблемы вводим переменную (count) для отслеживания колич-ва элементов в буфере (N). Если соunt = N, то производитель уходит в состояние ожидания, в противном случае данные помещаются в буфер и count увеличивается. При соunt =0, потребитель уходит в состояние ожидания. В противном случае оттуда извлекается информация и значение count уменьшается. Производитель может активизировать потребителя с помощью вызова Wakeup в тот момент, когда потребитель не находится в состоянии ожидания.

Алгоритм Петерсона дает нам решение задачи корректной организации взаимодействия двух процессов. Давайте рассмотрим теперь соответствующий алгоритм для n взаимодействующих процессов, который получил название алгоритм булочной, Основная его идея выглядит так. Каждый вновь прибывающий клиент (он же процесс) получает талончик на обслуживание с номером. Клиент с наименьшим номером на талончике обслуживается следующим. К сожалению, из-за неатомарности операции вычисления следующего номера алгоритм булочной не гарантирует, что у всех процессов будут талончики с разными номерами. В случае равенства номеров на талончиках у двух или более клиентов первым обслуживается клиент с меньшим значением имени (имена можно сравнивать в лексикографическом порядке). Разделяемые структуры данных для алгоритма — это два массива

*shared enum {false, true} choosing[n];*

*shared int number[n];*

Изначально элементы этих массивов инициируются значениями false и 0 соответственно. Введем следующие обозначения

*(a,b) < (c,d), если a < c или если a == c и b < d*

*max(a0, a1, ...., an) — это число k такое, что k >= ai для всех i = 0, ...,n*

Структура процесса Pi для алгоритма булочной приведена ниже

while (some condition) {

choosing[i] = true;

number[i] = max(number[0], ..., number[n-1]) + 1;

choosing[i] = false;

for(j = 0; j < n; j++){

while(choosing[j]);

while(number[j] != 0 && (number[j],j) < (number[i],i));

}

critical section

number[i] = 0;

remainder section

}

**20. Семафорные примитивы Дейкстры. Решение проблемы производителя и потребителя с помощью семафора**

Семафор – целочисленная не отрицательная перемен доступ любого процесса к которой за исключением момента ее инициализац может осущ только через 2 атамарные операции P V(Proberen Verhogen) (привер\ увелич). При выполнении операции Р, над семафором, сначала проверяется его значение. Если оно больше нуля, то из него вычитают 1 и продолжается работа процесса. Если оно равно 0, то процесс переходит в состояние блокировки или ожидания до тех пор, пока значение не станет больше нуля. После этого из него вычит 1 и продолжается работа процесса. При выполнении операции V к значению семаф прибавляется 1. в момент инициализации, семафору может быть присвоено любое значение. Мьютекс (mutex)- двоичный семафор (только 0 и 1). Семафор принимает значение 0 и 1. 1-свободно, 0-занято. Операция V над мьют не увеличивает его значение не больше чем на 1.

**21. Монитор Хоара как примитив синхронизации высокого уровня**

Монитор – особый тип данных, обладающий внутренними переменными описывающ (определяющ) его состояние. Значение этих переменных из вне могут быть изменены только с помощью вызова функц методов этого монитора. Монитор представляет собой особые конструкции языка программ и компилятора обрабат методы мониторов специальным образом, добавляя к ним пролог и эпилог, таким образом реализуется взаимоисключение. Достоинство: вероятность ошибок программ значительно меньше. Условные переменные – если функц не может выполняться дальше, пока не наступит некоторое событие. Она выполняет операцию wait над условной переменной. При этом процесс блокируется и в монитор может выйти другой процесс. Когда событие происходит, другой процесс выполняет операц. c.signal от этого ранее заблокированный процесс становится активным. Если на этой перемен блокировалось несколько процессов, то активизируется только один из них. Условные переменные в отличии от семафора не помнят предысторию. Соответ, если произведена операция signal над условн перемен которые которые никто не заблокиров, то данный сигнал будет утерян и никем не использован. Сообщение – используется два примитива. Оба имеют скрытый механизм взаимоисключ и блокировка из чтения пустого буфера и запись в переполнен. Решение задачи произв\потреб с помощью сообщ тривиально.

**22. Поняие виртуального и физического адреса.**

Оперативная память может быть представлена в виде последовательности бит, каждый из которых имеет свой уникальный физический адрес. Адреса, используемые процессами и ядром не обязательно соответствуют физич. адресам. Можно сказать, что процесс выполняется с собственным виртуальным адресном пространстве. Механизм отображения виртуального адреса в физическом в современных компьютерных системах выполняется на аппаратном уровне. с помощью устройства управления памятью процессора. Современные процессоры поддерживают объединение адресного пространства: размера сегмента и области фиксированного размера- границы. При этом для каждого сегмента или может быть задано собственное отображение виртуальных адресов в физическом адресном пространстве. Размер виртуальной памяти может превышать размер физической памяти засчет вторичной памяти.

**23. Назначение и функции подсистемы управления памятью**

Для подзадач ОС ресурс памяти ограничен. Данные, которые не могут быть размещены в ОЗУ, располагаются во вторичной памяти, роль которой выполняют дисковые накопители . Распределение памяти в ОС осуществляет подсистема управления памятью, которая для многозадачных С обеспечивает: 1)выполнение задач объем которых > ОП 2)выполнение частично загруженных в ОП задач 3)размещение в ОП сразу нескольких задач 4) размещение задач в произвольном месте ОЗУ 5)размещение задачи в различных частях ОЗУ 6)совместное использование несколькими задачами одних и тех же областей ОЗУ

Функции системы управления памятью 1)учет свободной и занятой памяти 2)принятие решения о распределении памяти 3) выделение памяти процессу и учет информации о ее использовании 4) освобождение памяти и корректировка ее состояния.

При распределении ОЗУ возможны 2 метода 1) статистический (память назначается до выполнения программы) 2)динамический (во время выполнения программы)

**24. Статическое распределения памяти.**

При статическом распределении вся необходимая оперативная память выделяется процессу в момент его порождения. При этом память выделяется единым блоком необходимой длины, начало которого определяется базовым адресом. Программа пишется в адресах относительно начала блока, а физический адрес команды или операнда при выполнении программы формируется как сумма базового адреса блока и относительного адреса в блоке. Значение базового адреса устанавливается при загрузке программы в оперативную память. Так как в разных программах используются блоки разной длины, то при таком подходе возникает проблема фрагментации памяти, то есть возникают свободные участки памяти, которые невозможно без предварительного преобразования использовать для вычислительного процесса.

**25. Динамическое распределение памяти: дисциплины диспетчеризации. Уплотнение.**

При динамическом распределении памяти каждой программе в начальный момент выделяется лишь часть от всей необходимой ей памяти, а остальная часть выделяется по мере возникновения реальной потребности в ней.Такой подход базируется на следующих предпосылках.

Во-первых, при каждом конкретном исполнении в зависимости от исходных данных некоторые части программы (до 25% ее длины) вообще не используются. Следует стремиться к тому, чтобы эти фрагменты кода не загружались в ОП.

Во-вторых, исполнение программы характеризуется так называемым принципом локальности ссылок. Он подразумевает, что при исполнении программы в течение некоторого относительно малого интервала времени происходит обращение к памяти в пределах ограниченного диапазона адресов (как по коду программы, так и по данным). Следовательно, на протяжении этого времени нет необходимости хранить в ОП другие блоки программы.

При этом системные средства должны отслеживать возникновение требований на обращение к тем частям программы, которые в данный момент отсутствуют в ОЗУ, выделять этой программе необходимый блок памяти и помещать туда из внешнего ЗУ требуемую часть программы. Для этого может потребоваться предварительное перемещение некоторых блоков информации из ОЗУ во внешнюю память. Данные перемещения должны быть скрыты от пользователя и в наименьшей степени замедлять работу его программы.

Перемещение блоков информации из ОЗУ во внешнюю память с целью освобождения места для новой информации происходит обычно по одному из следующих алгоритмов:

LRU (least recently used) - наиболее давно не использовавшийся;

FIFO - самый давний по пребыванию в ОЗУ;

Random - случайным образом.

Динамическое распределение памяти тесно переплетается с понятием виртуальной памяти.

**Уплотнение памяти.**

Т.к. задачи появляются в разное время и различны их объемы, возможна сильная фрагментация( объем свободной памяти велик, но нельзя её выделить). Организуется уплотнение памяти – все вычисления приостанавливаются и диспетчер памяти корректирует списки, перемещая все разделы например в начало памяти.

*Недостаток*: время на уплотнение

*Преимущество:* можно выделить необходимый размер

**26. Сегментная организация памяти. Дескриптор сегмента. Трансляция адресов, основанная на сегментации.**

Понятие о сегментированной модели памяти

 Память для программы делится на непрерывные области памяти, называемые сегментами.

Сегменты - это логические элементы программы.

Сама программа может обращаться только к данным, которые находятся в этих сегментах.

 Сегмент представляет собой независимый, поддерживаемый на аппаратном уровне блок памяти.

 Сегментация - механизм адресации, обеспечивающий существование нескольких независимых адресных пространств как в пределах одной задачи, так и в системе в целом для защиты задач от взаимного влияния.

 Замечание. Программист может либо самостоятельно разбивать программу на фрагменты (сегменты), либо автоматизировать этот процесс и возложить его на систему программирования.

 Для микропроцессоров Intel принят особый подход к управлению памятью. Каждая программа в общем случае может состоять из любого количества сегментов, но непосредственный доступ она имеет только к 3 основным сегментам: кода, данных и стека и к дополнительным сегментам данных (всего 3).

 Операционная система (! а не сама программа) размещает сегменты программы в ОП по определенным физическим адресам, а значения этих адресов записывает в определенные места, в зависимости от режима работы микропроцессора:

       в реальном режиме адреса помещаются непосредственно в сегментные регистры (cs, ds, ss, es, gs, fs);

       в защищенном режиме - в специальную системную дескрипторную таблицу (Элементом дескрипторной таблицы является дескриптор сегмента. Каждый сегмент имеет дескриптор сегмента -8 байт. Существует три дескрипторные таблицы. Адрес каждой таблицы записывается в специальный системный регистр).

 Для доступа к данным внутри сегмента обращение производится относительно начала сегмента линейно, т.е. начиная с 0 и заканчивая адресом, равным размеру сегмента. Этот адрес называется смещением (offset).

 Таким образом, для обращения к конкретному физическому адресу ОП необходимо определить адрес начала сегмента и смещение внутри сегмента.

Физический адрес принято записывать парой этих значений, разделенных двоеточием

segment : offset

Например, 0040:001Ch; 0000:041Ch; 0020:021Ch; 0041:000Ch.

 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.

Достоинства:

1)     общий объем виртуальной памяти превосходит объем физической памяти

2)     возможность размещать в памяти как можно больше задач (до определенного предела) ⇒ увеличивает загрузку системы и более эффективно используются ресурсы системы

 Недостатки:

1)     увеличивается время на доступ к искомой ячейке памяти, т.к. должны вначале прочитать дескриптор сегмента, а потом уже, используя его данные, можно вычислить физический адрес (для уменьшения этих потерь используется кэширование - дескрипторы, с которыми работа идет в данный момент размещаются в сверхоперативной памяти - в специальных регистрах процессора);

2)     фрагментация;

3)     потери памяти на размещение дескрипторных таблиц

4)     потери процессорного времени на обработку дескрипторных таблиц.

 Сегментированная модель памяти поддерживается и в реальном, и в защищенном режимах работы микропроцессора.

**27. Сегментно-страничная организация памяти. Преимущества и недостатки.**

Виртуальное адресное пространство процесса разбивается на сегменты, а каждый сегмент разбивается на страницы фиксированного размера. Перемещение данных между основной и внешней памятью осуществляется страницами. Однако разбиение на сегменты позволяет определять разные права доступа к разным частям кода и данных программы. С каждым процессом

связана одна таблица сегментов и несколько (по одной на сегмент) таблиц страниц. Начальный адрес таблицы сегментов извлекается из регистра процессора. Первая компонента виртуального адреса содержит номер сегмента, который служит указателем в таблице сегментов на таблицу страниц. Вторая компонента виртуального адреса (смещение относительно начала сегмента) состоит из двух полей: номера виртуальной страницы, который служит указателем в таблице страниц на физическую страницу и смещения относительно начала страницы.

*Достоинства:* разбиение программы на сегменты позволит размещать сегменты целиком в памяти, причем все страницы сегмента загружаются в память. Это уменьшает количество страничных прерываний. Страницы исполняемого сегмента могут располагаться не обязательно радом в памяти.

*Недостатки*: большая задержка доступа к памяти (в 3 раза больше), данный способ требует значительных затрат вычислительных ресурсов.

В большинстве современных микропроцессорах – плоская модель памяти – один сегмент разбитый на страницы.

Сегментно-страничная и страничная организация памяти позволяет легко организовать совместное использование одних и тех же данных и программного кода разными задачами. Для этого различные логические блоки памяти разных процессов отображают в один и тот же блок физической памяти, где размещается разделяемый фрагмент кода или данных.

Оверлей – структурная разбивка исполняемого кода на чисти которые подгружаются ОС в оперативн память только при необходимости.

**28. Страничная организация памяти. Дескриптор страниц.**

При этом распределении виртуального прост-во каждого процесса делится на одинаковые фиксированные для данной системы. Вся ОП машины делится на части этих размеров – физические страницы. Имеется таблица распределения страниц – загружается в спец.регистр процессора. И на них организуется работа.

Может храниться управляющая инфо – в таблице страниц. Она:

-признак невыгружаемости страниц

-признак модификации страниц

-признак обращения к странице(счетчик обращений)

-вся управл.инфо используется механизмом страничного модифицирования.

При каждом обращении к памяти происходит чтение из табл.страниц инфо о Вирт.странице. Если данная страница находится в памяти, то производится преобр.вирт.адреса в физический. Если она выгружена, то – стр.прерыв. Наполняющийся процесс переводится в состояние ожидания и активизируется в другой, готовый к исполнению процесс. Параллельно программа обработки страничного прерывания находит на диске нужную Вирт.страницу и попытки загрузить её. Если в памяти свободная страница, то загрузка выполняется немедленно.

Если своб.страниц нет, то что-то выгружается. Выбор выгружения страниц:

- выгр.может дольше всего не использ.страница

- первая попавшаяся

- страница, к-й в последнее время было мало обращений.

Рабочее множество страниц - определяется для каждого процесса и представляет собой перечень наиболее часто используемых страниц, к-е должны постоянно находиться в памяти. В зависимости от признака модификации и выбора страниц, её версия может быть записана или просто уничтожена.

Страничное распределение памяти может быть организовано в упрощ.варианте без выгрузки на диск. В этом случае все Вирт.стр-цы процессов постоянно находятся в памяти. Такой метод не предоставляет Вирт.памяти, но позволяет избежать фрагментации, засчет того, что программа может загружаться в несмежной области и при загрезке страниц не образуется остатков.

Для страницы существует дескриптор страницы. Основные поля: поле номера страницы (адрес физической страницы), бит присутствия, бит обращения (R), бит модификации (M) – предназначен для отображения что данную страницу изменяли и ОС следует переписать ее при замещении, биты уровня доступа – бит привилегий доступа, бит контроля доступа(ноль – только чтение).

**29. Виртуальная память. Архитектура**

Виртуальная память – то чего нет в реальности. Информация с которой работает процесс должна располагаться в оперативной памяти. В схемах виртуальной памяти у процесса создается иллюзии, что вся необх инф находится в основной памяти. При этом программа не ограничена объемом физической памяти. Задействуются большие виртуальные пространства. В памяти могут размещаться программы не целиком, а частично за счет гибкого перераспределения памяти между программами. Характерный размер виртуального адресного пространства для 32 разряд – 4 Гб. Каждый процесс получает свое адресное виртуальное пространство таких размеров, что при этом не может адресоваться чужое адресное пространство. Обеспечивается защита данных программ и ос. Аппаратура компьют принимает участие в трансляции адресов. Главный ее вклад в схему виртуальн памяти - генерация исключительных ситуаций при отсутствии в памяти нужных страниц.

Страничная виртуальная память.

Физич память делится на кадры. Адрес – номер и смещение внутри страниц. Процесс может выполняться если его текущие страницы находятся в его ОП. В случае если нужная страница в памяти отсутствует, она переписывается (подкачивается) из внешней памяти в любой свободный страничный кадр. Перед тем, как страница подкачив возникает страничное прерывание. Выполнение команды прерывается и начинается процесс подкачки. После чего продолжается выполнение команды. При отсутствии свободных страничных кадров, одна из страниц вытесняется во внешнюю память. Сегментно-страничная виртуальная память. Неперекрывающиеся сегменты организуются в едином виртуальном адресном простран.

Таблица страниц. Номер записи в таблице страниц соот номеру виртуальной страницы. Из этой записи можно получить номер кадра для данной страницы. Биты страницы модификации устанавлив, если страница была изменена в главной памяти и необходима ее перезапись во внешней памяти. Бит ссылки полагает отсеять малоиспользов страницы, вид кэширования помогает кэшировать страницу в более быстрой памяти.

Многоуровневые таблицы страниц. Из-за того, что каждому процессу необход выделить свою таблицу страниц, а размеры ее очень велики, таблица разбивается на фрагменты. 32 разрядный адрес делится на 10 разрядное поле.

**30. Защита адресного пространства задач в многозадачных ОС**

*Модель без защиты –* системное и пользовательское адресные пространства не защищены друг от друга, используется два сегмента памяти: для кода и для данных; при этом от системы не требуется никакого управления памятью, не требуется MMU (memory management unit – специальное аппаратное устройство для поддержки управления виртуальной памятью).

*Модель защиты система/пользователь* – системное адресное пространство защищено от адресного пространства пользователя, системные и пользовательские процессы выполняются в общем виртуальном адресном пространстве, при этом требуется MMU. Защита обеспечивается страничным механизмом защиты. Различаются системные и пользовательские страницы. Пользовательские приложения никак не защищены друг от друга. Процессор находится в режиме супервизора, если текущий сегмент имеет уровень 0, 1 или 2. Если уровень сегмента – 3, то процессор находится в пользовательском режиме. В этой модели необходимы четыре сегмента – два сегмента на уровне 0 (для кода и данных) и два сегмента на уровне 3. Механизм страничной защиты не добавляет накладных расходов, т.к. защита проверяется одновременно с преобразованием адреса, которое выполняет MMU; при этом ОС не нуждается в управлении памятью.

*Модель защиты пользователь/пользователь* – к *модели* *система/пользователь* добавляется защита между пользовательскими процессами; требуется MMU. Как и в предыдущей модели, используется механизм страничной защиты. Все страницы помечаются как привилегированные, за исключением страниц текущего процесса, которые помечаются как пользовательские. Таким образом, выполняющийся поток не может обратиться за пределы своего адресного пространства. ОС отвечает за обновление флага привилегированности для конкретной страницы в таблице страниц при переключении процесса. Как и в предыдущей модели используются четыре сегмента.

*Модель защиты виртуальной памяти* – каждый процесс выполняется в своей собственной виртуальной памяти, требуется MMU. У каждого процесса имеются свои собственные сегменты и, следовательно, своя таблица описателей. ОС несет ответственность за поддержку таблиц описателей. Адресуемое пространство может превышать размеры физической памяти, если используется страничная организация памяти совместно с подкачкой. Однако в системах реального времени подкачка обычно не применяется из-за ее непредсказуемости. Для решения этой проблемы доступная память разбивается на фиксированное число логических адресных пространств равного размера. Число одновременно выполняющихся процессов в системе становится ограниченным.

**31. Физические принципы организации ввода/вывода в ОС.**

Устройства ввода-вывода делятся на два типа: *блок-ориентированные* устройства и *байт-ориентированные* устройства. Блок-ориентированные устройства хранят информацию в блоках фиксированного размера, каждый из которых имеет свой собственный адрес. Самое распространенное блок-ориентированное устройство - диск. Байт-ориентированные устройства не адресуемы и не позволяют производить операцию поиска, они генерируют или потребляют последовательность байтов. Примерами являются терминалы, строчные принтеры, сетевые адаптеры. Однако некоторые внешние устройства не относятся ни к одному классу, например, часы, которые, с одной стороны, не адресуемы, а с другой стороны, не порождают потока байтов. Это устройство только выдает сигнал прерывания в некоторые моменты времени.

Внешнее устройство обычно состоит из механического и электронного компонента. Электронный компонент называется контроллером устройства или адаптером. Механический компонент представляет собственно устройство. Некоторые контроллеры могут управлять несколькими устройствами. Если интерфейс между контроллером и устройством стандартизован, то независимые производители могут выпускать совместимые как контроллеры, так и устройства.

Операционная система обычно имеет дело не с устройством, а с контроллером. Контроллер, как правило, выполняет простые функции, например, преобразует поток бит в блоки, состоящие из байт, и осуществляют контроль и исправление ошибок. Каждый контроллер имеет несколько регистров, которые используются для взаимодействия с центральным процессором. В некоторых компьютерах эти регистры являются частью физического адресного пространства. В таких компьютерах нет специальных операций ввода-вывода. В других компьютерах адреса регистров ввода-вывода, называемых часто портами, образуют собственное адресное пространство за счет введения специальных операций ввода-вывода (например, команд IN и OUT в процессорах i86).

ОС выполняет ввод-вывод, записывая команды в регистры контроллера. Например, контроллер гибкого диска IBM PC принимает 15 команд, таких как READ, WRITE, SEEK, FORMAT и т.д. Когда команда принята, процессор оставляет контроллер и занимается другой работой. При завершении команды контроллер организует прерывание для того, чтобы передать управление процессором операционной системе, которая должна проверить результаты операции. Процессор получает результаты и статус устройства, читая информацию из регистров контроллера.

**32. Классификация устройств в/в. Контроллеры устройств в/в.**

*Классификация устройств ввода-вывода*

В общем случае две категории – блочные и символьные

*Блочное* – хранит информацию в виде блоков фиксированного размера , каждый блок имеет адрес и может быть прочитан независимо от остальных.

*Символьные* – предоставляемый или принимаемый поток символов без блочной структуры, здесь нет адресации, следовательно нет операции поиска (например мышь).

Любое устройство ввода-вывода состоит из двух частей – *механическая* и *электронная* (контроллер).

*Контроллер* – микросхема, физически управляющая устройствами. Контроллер принимает команды ОС и дешифрует их для устройства. Интерфейс между контроллером и устройством – интерфейс низкого уровня, как правило к контроллеру может подключатся 1,2,4,8 устройств однотипных. Для связи с каждым контроллером существует определенное количество регистров (т.н. “управляющих”)

**33. Сетевые операционные системы и операционные системы суперкомпьютеров.**

Сетевые ОС: Сетевые и распределенные.

ОС, позволяющие пользователю работать в сети, называются сетевыми. Набор ОС отдельных компьютеров составляет сеть. Включают согласованный набор коммуникационных протоколов для взаимодействия процессов на различных компьютерах и для разделения ресурсов между пользователями сети. В сетевой ОС пользователь всегда знает, что имеет дело с сетевым ресурсом. В распределенных ОС сетевые ресурсы – это единая виртуальная машина.

**Функциональные компоненты сетевой ОС:** серверная часть – та часть, которая обеспечивает доступ к вашей системе.клиентская часть – средства запроса доступа к удаленным ресурсам и услугам.транспортное средство**Функции клиентской части:** способность отличить запрос к удаленному файлу от локального.преобразование формата запроса к ресурсам, прием запросов от приложений на доступ в сети и получение ответов от серверной части своей и другой системы.Сетевая служба – совокупность клиентской и серверной частей ОС, представляет доступ к конкретному типу ресурса компьютера. Службы делятся на административные и пользовательские.Узлы в сети бывают одного из трех видов:одноранговый узел – совмещает функции клиента и сервераклиентсерверВ одноранговых сетях все узлы равны в возможности доступа друг к другу. Сети с выделенными серверами – ОС оптимизированы для работы в качестве сервера и клиента. **Отличие серверных ОС**поддержка мощных платформподдержка большого числа одновременно выполняемых процессов и соединений.наличие средств централизованного администрирования сети.более широкий набор сетевых служб **ОС суперкомпьютеров**Философское определение – компьютер, мощность которого всего на порядок меньше, чем нужно для решения поставленных задач.Экономическое определение – система, цена которой выше 1-2 млн $.Для их ОС характерно:высокая степень отказоустойчивости.отсутствие функциональной избыточностиналичие специальных средств диагностики и мониторинга работы процессора и связей.