1. **Особенности ОС с микроядерной архитектурой. Достоинства и недостатки.**

Микроядерная архитектура является альтернативой классическому способу построения операционной системы. Под классической архитектурой в данном случае понимается структурная организация ОС, в соответствии с которой все основные функции операционной системы, составляющие многослойное ядро, выполняются в привилегированном режиме.
Суть микроядерной архитектуры состоит в следующем. В привилегированном режиме остается работать только очень небольшая часть ОС, называемая микроядром. Микроядро защищено от остальных частей ОС и приложений. В состав микроядра обычно входят машинно-зависимые модули, а также модули, выполняющие базовые (но не все) функции ядра по управлению процессами, обработке прерываний, управлению виртуальной памятью, пересылке сообщений и управлению устройствами ввода-вывода, связанные с загрузкой или чтением регистров устройств. Все остальные более высокоуровневые функции ядра оформляются в виде приложений, работающих в пользовательском режиме.
Менеджеры ресурсов, вынесенные в пользовательский режим, называются серверами ОС, то есть модулями, основным назначением которых является обслуживание запросов локальных приложений и других модулей ОС. Для реализации микроядерной архитектуры необходимым условием является наличие в операционной системе удобного и эффективного способа вызова процедур одного процесса из другого. Поддержка такого механизма и является одной из главных задач микроядра. Работа микроядерной операционной системы соответствует известной модели клиент-сервер, в которой роль транспортных средств выполняет микроядро.
Преимущества и недостатки микроядерной архитектуры.
Операционные системы, основанные на концепции микроядра, в высокой степени удовлетворяют большинству требований, предъявляемых к современным ОС, обладая переносимостью (весь машинно-зависимый код изолирован в микроядре, поэтому для переноса системы на новый процессор требуется меньше изменений и все они логически сгруппированы вместе), расширяемостью, присущей микроядерной ОС в очень высокой степени, надежностью и создавая хорошие предпосылки для поддержки распределенных приложений. За эти достоинства приходится платить снижением производительности, и это является основным недостатком микроядерной архитектуры. При классической организации ОС выполнение системного вызова сопровождается двумя переключениями режимов, а при микроядерной организации - четырьмя.Таким образом, операционная система на основе микроядра при прочих равных условиях всегда будет менее производительной, чем ОС с классическим ядром. Именно по этой причине микроядерный подход не получил такого широкого распространения, которое ему предрекали.

2. **Особенности перехода процесса из одного состояния в другое**

Во время своего существования процесс может находиться в четырех состояниях: ***подготовленном, готовом, активном и заблокированном*** (рис.2.10). В каждый конкретный момент времени процесс находится в одном из этих состояний, которое фиксируется в блоке управления процессом (PCB).

Процесс находится в **подготовленном** состоянии, если он находится в системе (как правило на внешнем носителе информации), но ему не выделены ресурсы системы.

Процесс находится в **готовом** состоянии или активизирован, если ему уже выделены ресурсы (программа, необходимая для выполнения, находится в оперативной памяти), однако ему не выделено время процессора для выполнения (процессор занят другим процессом).

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.

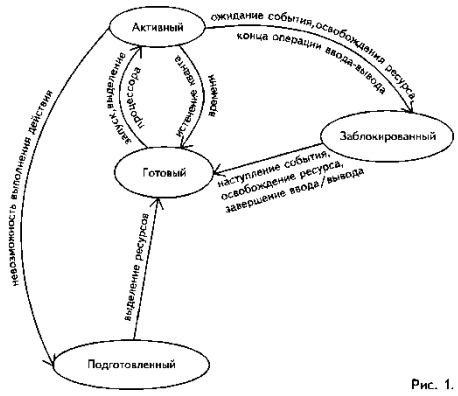
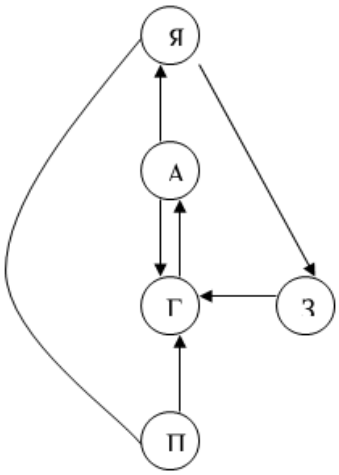


Рис. 1.

Системные задачи формируются и активизируются при загрузке системы.
Подготовленный процесс – еще не задача, не готова к выполнению, но имеет ресурсы(не все)
Если процесс нашей задачи пытается активизировать действия, связанные с О.С., то задача входит в заблокированное состояние.
Заблокирован до тех пор, пока системная оп-ция, которая была активизирована, не выполнится и пришлет сообщение о выполнении (событие)



Если во время выполнения задачи оказалось, что задачу не возможно продолжать выполнять из-за отсутствия ресурса, то она приостанавливается и переходит в «режим ядра». Если ресурсы найдены, то задача переходит в состояние заблокированной .

Нельзя сразу перейти из активизированного в заблокированный «режим ядра» - это шлюз.

Если система считает, что ко-во активизированных процессов у нее слишком большее, она может выбрать некоторые процессы по своему усмотрению. Решает она по показателю эффективности работы системы.

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.

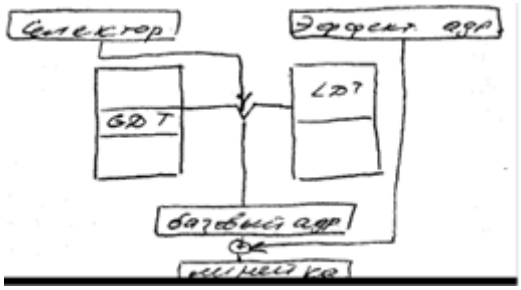
Когда процесс переходит из активного состояния в готовое. ? - Если активный процесс не выполнился за выделенный ему квант времени, то он переходит снова в готовое состояние.

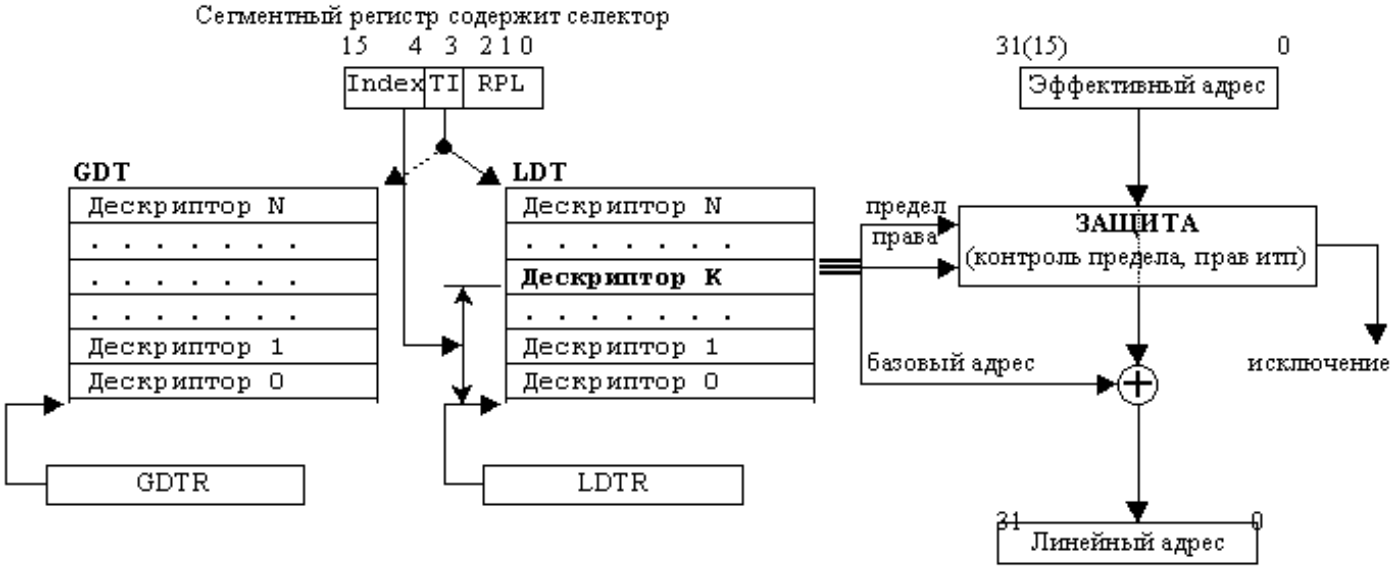
*Условие перехода процесса из активного состояния в заблокированное.?*Процесс, которому выделяется время процессора, переводиться в активное состояние или состояние выполнения.

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.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.

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).

3. Особенности формирования эффективного адреса в защищенном режиме.





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

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.

4. Принципы борьбы с тупиками

Последствия тупиков равносильны ситуации бесконечного откладывания. Тупики и ситуация бесконечного откладывания — это ситуации равносильные "потери процесса".

Можно выделить 4 стратегии борьбы с тупиками:

Предотвращение тупика. Требуется создать условия, исключающие возможность возникновения тупиковых ситуаций. Однако этот метод часто приводит к нерациональному использованию ресурсов.
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.
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 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.

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.

5. Особенности применения сигналов

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

1. Завершить выполнение процесса
2. Игнорировать сигнал
3. Остановить процесс
4. 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:

1. Непосредственно перед возвратом процесса из режима ядра в режим задачи при обработке системного вызова
2. Перед переходом процесса в состояние сна, если он имеет приоритет, допускающий прерывание сигнала
3. Сразу же после пробуждения с приоритетом, допускающим прерывание сигнала
4. 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:

1. 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

2. Не допускается