1. Многоуровневые операционные системы. Достоинства и недостатки.

многоуровневые (ОС с кольцевой структурой). Система включала 6 уровней. Уровень 0 занимался распределением времени процессора, переключая процессы при возникновении прерывания или при срабатывании таймера. Над уровнем 0 система состояла из последовательных процессов, каждый из которых можно было запрограммировать, не заботясь о том, что на одном процессоре запущено несколько процессов. Другими словами, уровень 0 обеспечивал базовую многозадачность процессора.

Уровень 1 управлял памятью. Он выделял процессам пространство в оперативной памяти и на магнитном барабане объемом 512 К слов для тех частей процессов (страниц), которые не помешались в оперативной памяти. Процессы более высоких уровней не заботились о том, находятся ли они в данный момент в памяти или нэ барабане. Программное обеспечение уровня 1 обеспечивало попадание страниц в оперативную память по мере необходимости.

Уровень	Функция
5	Оператор
4	Программы пользователя
3	Управление вводом-выводом
2	Связь оператор-процесс
1	Управление памятью и барабаном
0	Распределение процессора и многозадачность

Уровень 2 управлял связью между консолью оператора и процессами. Таким образом, все процессы выше этого уровня имели свою собственную консоль оператора. Уровень 3 управлял устройствами ввода-вывода и буферизовал потоки информации к ним и от них. Любой процесс выше уровня 3, вместо того чтобы работать с конкретными устройствами, с их разнообразными особенностями, мог обращаться к абстрактным устройствам ввода-вывода, обладающим удобными для пользователя характеристиками. На уровне 4 работали пользовательские программы, которым не надо было заботиться ни о процессах, ни о памяти, ни о консоли, ни об управлении устройствами ввода-вывода. Процесс системного оператора размещался на уровне 5.

Микроядерная архитектура – функциональная независимость и универсальность системы. Ядро должно работать с программой пользователя. Здесь имеется ядро и функциональные блоки. Ядро здесь компактное и только передает блокам информацию по синхронизации и отслеживает «общение» между процессами, а дальше блоки работают самостоятельно

2. Причины приостановки процессов. Принципы выбора процесса для приостановки

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.

3. TSS структура, назначение. Участие в направлении процессми.

TSS(task status segment)- это специальный системный сегмент (состояния задачи), (содержит состояние регистров процессора и регистра флагов).Когда задача(процесс) неактивна в нём хранится вся информация про значения всех РОНов и системных регистров.В TSS находится вся информация, необходимая для восстановления задачи после прерывания(переключение задач предусматривает сохранение TSS для старой задачи в соответствующие регистры). Место расположение TSS для каждой задачи это GDT.

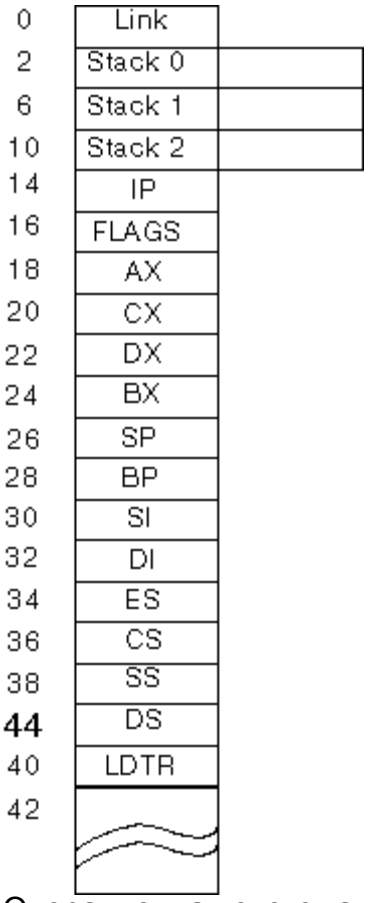

Рис. 14. Формат сегмента состояния задачи TSS.

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

Поле доступа дескриптора TSS

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.

4. Назначение КЕШ памяти (Назначение MMU, TLB)

КЭШ – способ организации совместного функционирования двух типов запоминающих устройств(ЗУ), отличающихся временем доступа и стоимосью хранения данных, который позволяет уменьшить среднее время доступа к данным за счет динамического копирования в быстрое ЗУ наиболее часто используемой информации из «медленного» ЗУ

Большинство систем виртуальной памяти используют технику, называемую **страничной организацией памяти** (paging), которую мы сейчас опишем. На любом компьютере существует множество адресов в памяти, к которым может обратиться программа. Когда программа использует следующую инструкцию

MOV REG.1000

она делает это для того, чтобы скопировать содержимое памяти по адресу 1000 в регистр REG (или наоборот, в зависимости от компьютера). Адреса могут формироваться с использованием индексации, базовых регистров, сегментных регистров и другими путями.

Эти программно формируемые адреса, называемые **виртуальными адресами,** формируют **виртуальное адресное пространство.** На компьютерах без виртуальной памяти виртуальные адреса подаются непосредственно на шину памяти и вызывают для чтения или записи слово в физической памяти с тем же самым адресом. Когда используется виртуальная память, виртуальные адреса не передаются напрямую шиной памяти. Вместо этого они передаются **диспетчеру памяти** (MMU — Memory Management Unit), который отображает виртуальные адреса на физические адреса памяти, как продемонстрировано на рис. 4.9.

Пространство виртуальных адресов разделено на единицы, называемые **страницами.** Соответствующие единицы в физической памяти называются **страничными блоками** (page frame). Страницы и их блоки имеют всегда одинаковый размер. В этом примере они равны 4 Кбайт, но в реальных системах использовались размеры страниц от 512 байт до *64* Кбайт. Имея *64* Кбайт виртуального адресного пространства и 32 Кбайт физической памяти, мы получаем 16 виртуальных страниц и 8 страничных блоков. Передача данных между ОЗУ и диском всегда происходит в страницах. Когда программа пытается получить доступ к адресу 0, например, используя команду MOV REG.O виртуальный адрес 0 передается диспетчеру памяти (MMU). Диспетчер

памяти видит, что этот виртуальный адрес попадает на страницу 0 (от 0 до 4095), которая отображается страничным блоком 2 (от 8192 до 12287). Диспетчер переводит логич. В физич.

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.

При больших объёмах памяти, таблица описателей может занимать очень много, поэтому нецелесообразно хранить всю таблицу. Организовывается иерархия таблиц (обычно 2-3 уровня). То есть первая часть в виртуальном адресе указывает на строку в таблице верхнего уровня, вторая часть – в таблице второго у ровня и, наконец, последняя часть это смещение.

Каждая строка в таблице страниц содержит запись-описатель страницы. Обычно такой описатель содержит следующие поля: номер страничного блока на который отображена данная страницы, бит присутствия (отображена ли она в данный момент), бит защиты (какие операции можно производить над этой страницей), биты изменения и обращения (позволяет определить “грязная” или “чистая” данная страница). Стоит заметить, что адрес страницы на диске, не содержится в описателе, поскольку о нем должна знать ОС, а не аппаратура.

Для ускорения поиска в таблицах, используют аппаратные средства, а именно Translation Lookaside Buffer - TLB. Это ассоциативная память, в которую копируется часть, наиболее интенсивно используемых страниц (а такое множество можно выделить, исходя из принципа локальности). Сначала соответствующая страница ищется в TLB и только в случае если ее там нет, начинается поиск в таблицах страниц.

5. Различие в применении флажка, семафора, монитора

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;
3. **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.