БИЛЕТ № 2

1)Структура ядра ОС.

ЯДРО ОПЕРАЦИОННОЙ СИСТЕМЫ - комплекс программ и микропрограмм (резидент) для операций с процессами.

Операции, связанные с процессами, входят в базовое обеспечение машины. Они включаются в комплекс используемых средств с помощью программ и микропрограмм, совокупность которых составляет ядро ОС. Таким образом, все операции, связанные с процессами, осуществляются под управлением ядра ОС, которое представляет лишь небольшую часть кода ОС в целом. Поскольку эти программы часто используются, то резидентно размещаются в ОП. Другие же части ОС перемещаются в ОП по мере необходимости. Ядро ОС скрывает от пользователя частные особенности физической машины, предоставляя ему все необходимое для организации вычислений:

• само понятие процесса, а значит и операции над ним, механизмы ны-деления времени физическим процессам;

• примитивы синхронизации, рсали-юнанные ядром, которые скрывают от пользователя физические механизмы псресэановки контекста при реализации операций, связанных с прерываниями.

Выполнение программ ядра может осуществляться диумя способами:

1. Вызовом примигива управления процессами (создание, уничтожение, сипкронизация и т.д.); эти примитивы реализованы в виде обращений к супервизору.
2. Прерыванием: программы обработки прерываний сосгиияют часгь дра. т.к они нсгюсрсди пенно связаны с операциями синхронизации и и.юлиронаны ос шлсшнх уровней управления.

Таким образом, во всех случаях вход в ядро предусматривает сохранение слова состояния (при переключении контекста в PSW) и регистров процессора (в РСВ), который вызываетсупервизор или обрабатывает прерывание.

Ядро ОС содержит программы для реализации следующих функций:

• обработка прерываний;

• создание и уничтожение процессов;

• переключение процессов из состояния в состояние;

• диспетчеризацию заданий, процессов и ресурсов;

• приостановка и активизация процессов;

• синхронизация процессов;

• организация взаимодействия между процессами;

• манипулирование РСВ;

• поддержка операций ввода/вывода; / " поддержка распределения и перераспределения памяти;

• поддержка работы файловой системы;

. • поддержка механизма вызова— возврата при обращении к процедурам;

• поддержка определенных функций по ведению учета работы машины;

Одна из самых важных функций, реализованная в ядре — обработка прерываний
Какие программы находятся в ядре ОС (Виды программ) ядро супервизора, включающие только те программы, которые отвечают за реакцию системы.

Какая системная программа готовит команду начать ввод-вывод. Обработчик прерываний.

Какая часть ОС обрабатывает сигналы прерываний Ядро.

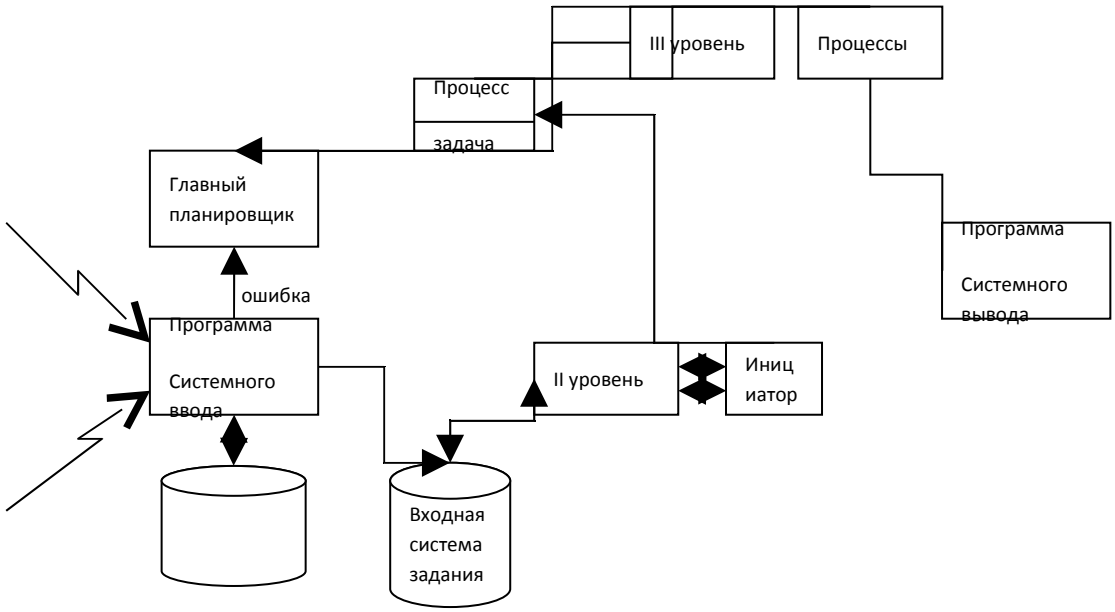
Выполнение программ ядра может осуществляться двумя способами:

- вызовом примитива управления процессами (создание, уничтожение, синхронизация и т.д.); эти примитивы реализованы в виде обращений к супервизору;

- прерыванием: программы обработки прерываний составляют часть ядра, т.к. они непосредственно связаны с операциями синхронизации и изолированы от высших уровней управления

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

2) Процедура ввода задания в ВС. Участие системных программ. Особенности реализации в разных ОС.
Конспект:



Программа системного ввода – планировщик 1 уровня
Командный интерпритатор берет входное задание
В том случае, если система считает, что есть ресурс для активизации задания, то активизируется планировщик 2-го уровня, то он выбирает задание с наивысшим приритетом. В том случае, если система считает, что есть ресурс для активизации задания, то активизируется планировщик. Инициатор проверяет наличие ресурсов для выполнения задания. Если ресурсов нет, то задание сбрасывается. Если всё в порядке, то образуется задача или процесс. Как только сформирован ТСВ система должна его обязательно выполнить. При формировании РСВ определяется программа подчиненная задаче, и данные, которые должна выполнить программа. Как только освобождается процессор, всплывает планировщик 3-го уровня, который обрабатывает ТСВ или PSB, ищет наиболее приоритетный процесс, который можно запустить
Доп. Інфа:
Система управления заданиями управляет прохождением заданий в ВС и выполняет функции:

1.Предоставление языковых средств управления работами в вычислительной системе.

2.Ввод и интерпретация заданий/команд.

3.Выделение и освобождение необходимых ресурсов.

4.Планирование заданий на выполнение.

5.Сбор и предоставление информации о состоянии заданий.

Существует три основных уровня планирования:

1.Планирование на верхнем уровне или планирование заданий.

На этом уровне осуществляется выбор заданий пользователем для выполнения и их запуск. Выбранные задания становятся готовыми процессами. Эту работу выполняет системный компонент - планировщик заданий.

2.Планирование на нижнем уровне или диспетчирование процессов.

Здесь осуществляется выбор готового процесса для выполнения, то есть предоставления ему ЦП. Выбранный процесс становится активным. Эту работу выполняет системный компонент - диспетчер.

3.Планирование на промежуточном уровне.

На данном уровне определяется, каким процессам будет разрешено состязаться за захват ЦП, то есть быть готовыми, и какие процессы будут кратковременно приостановлены (задержаны) для оптимизации загрузки системы. Промежуточное планирование управляет текущей производительностью вычислительной системы.

Эффективное планирование заданий и процессов является сложной проблемой, поскольку должно учитываться много противоречивых требований, таких как:

- справедливость;

- максимальная пропускная способность;

- приемлемое время ответа для максимального числа интерактивных пользователей;

- предсказуемость (задание должно выполняться примерно за одно время независимо от загрузки вычислительной системы);

- минимум накладных расходов на выполнение планирования;

- сбалансированность использования ресурсов;

- исключение бесконечного откладывания;

- учет приоритетов;

- отдавать предпочтение процессам, занимающие ключевые ресурсы;

- плавно деградировать при увеличении нагрузок.

Для того чтобы реализовать перечисленные требования, механизм планирования должен знать и учитывать следующие факторы:

- является ли процесс обменным (активно использующим операции ввода/вывода) или вычислительным (активно использующим процессор);

-является ли процесс пакетным или диалоговым;

- уровень реактивности интерактивного процесса;

-приоритетность процесса;

- частота прерываний по отсутствию нужной страницы;

- частота прерывания с низкого приоритета на высокий;

- длительность периода ожиданий ЦП процесса;

- суммарное использование времени ЦП и оценочное время, необходимое для завершения.

Для детального выделения действий, выполняемых при планировании, рассмотрим схему прохождения работ через ВС. При этом будем отслеживать прохождение задания с момента его ввода до полного завершения (вывода), фиксируя моменты изменения формы представления задания в ВС (Mi) и интервалы времени преобразования задания из одного состояния в другое (рис 4.2.)

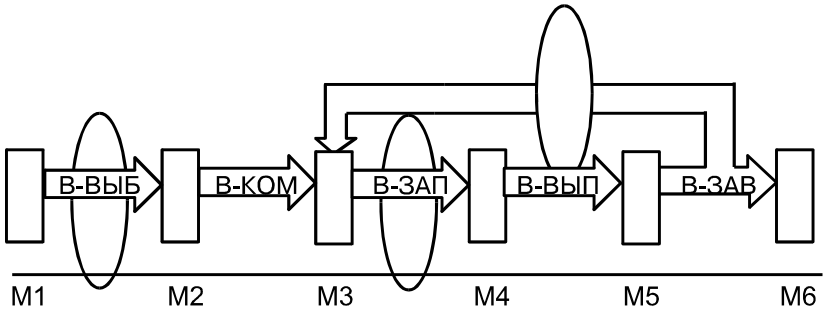


Рис. 4.2

Где:

- М1 — момент запроса: момент фиксации системой заявки/заявок на обслуживание;
- В-ВЫБ — время выборки: время предварительной обработки входного потока заявок;
- М2 — момент выборки: момент принятия решения об активизации заявок или отсрочке их выполнения;
- В-КОМ — время компиляции: время подготовки исходного описания задания-заявки и преобразования его в форму, необходимую для выполнения в параллельной системе;
- М3 — момент компоновки: момент, когда компиляция закончена, и входные задания переводятся в ранг "процессов", для которых ВС должна выделить ресурсы;
- В-ЗАП — время запуска: время, в течение которого выполняются действия, требуемые для запуска (инициирования) готовых процессов, — выделение ресурсов и перевод процессов в активное состояние;
- М4 — момент запуска: момент инициирования (активизации) задачи-процесса после распределения ресурсов (устройств, данных, памяти), который выполняется во время В-ЗАП.
- В-ВЫП — время выполнения: период времени, когда задания в системе активны, то есть выполняются.
- М5 — момент завершения: момент фиксации завершения задания (естественного или аварийного);
- В-ЗАВ — время завершения: время, в течение которого диспетчер ОС определяет, выполнено ли задание или его нужно продолжать выполнять. В последнем случае он ставит задание обратно в очередь на выполнение;
- М6 — момент выхода: момент, когда ОС считает задание полностью выполненным или устанавливает, что система не имеет возможности его дальнейшего выполнения, тогда оно удаляется из системы;
- Эллипс означает выполнение действий планировщика системы.
- заявок;
- требуется новый тип планировщика (назовем его ПТ — планировщик транспортный, рис. 4.17) для распараллеливания заданий, синхронизации процессов по данным — обеспечения поступления требуемых данных и поддержки связей между вычислительными узлами при реализации связи по данным;
- обработанные задания (или их распараллеленные модули) транслятором ОС и ПТ ставятся в новую очередь — буфер БУФ;
- к ПП добавляется функция *адаптирования*, при выполнении которой задания распределяются соответственно особенностям данной системы (например: специальные схемы для систем гиперкуб, транспьютерных систем или дополнительная схема для неоднородной среды).
- к ПН добавляется функция балансирования в случае реконфигурации (отказа некоторых элементов) системы.

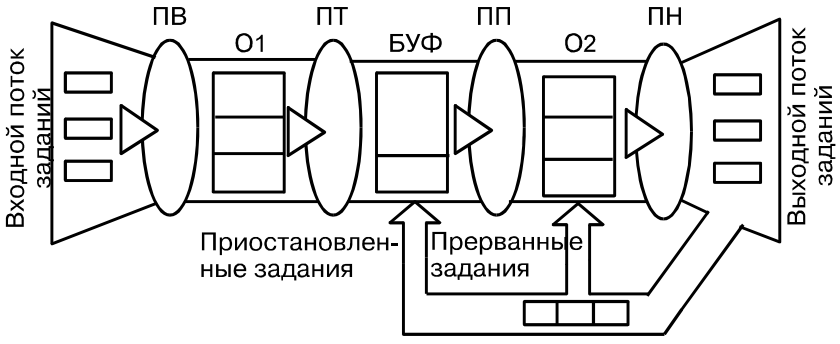


Рис. 4.17. Схема прохождения заданий через ВС

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

Табл. 4.1

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

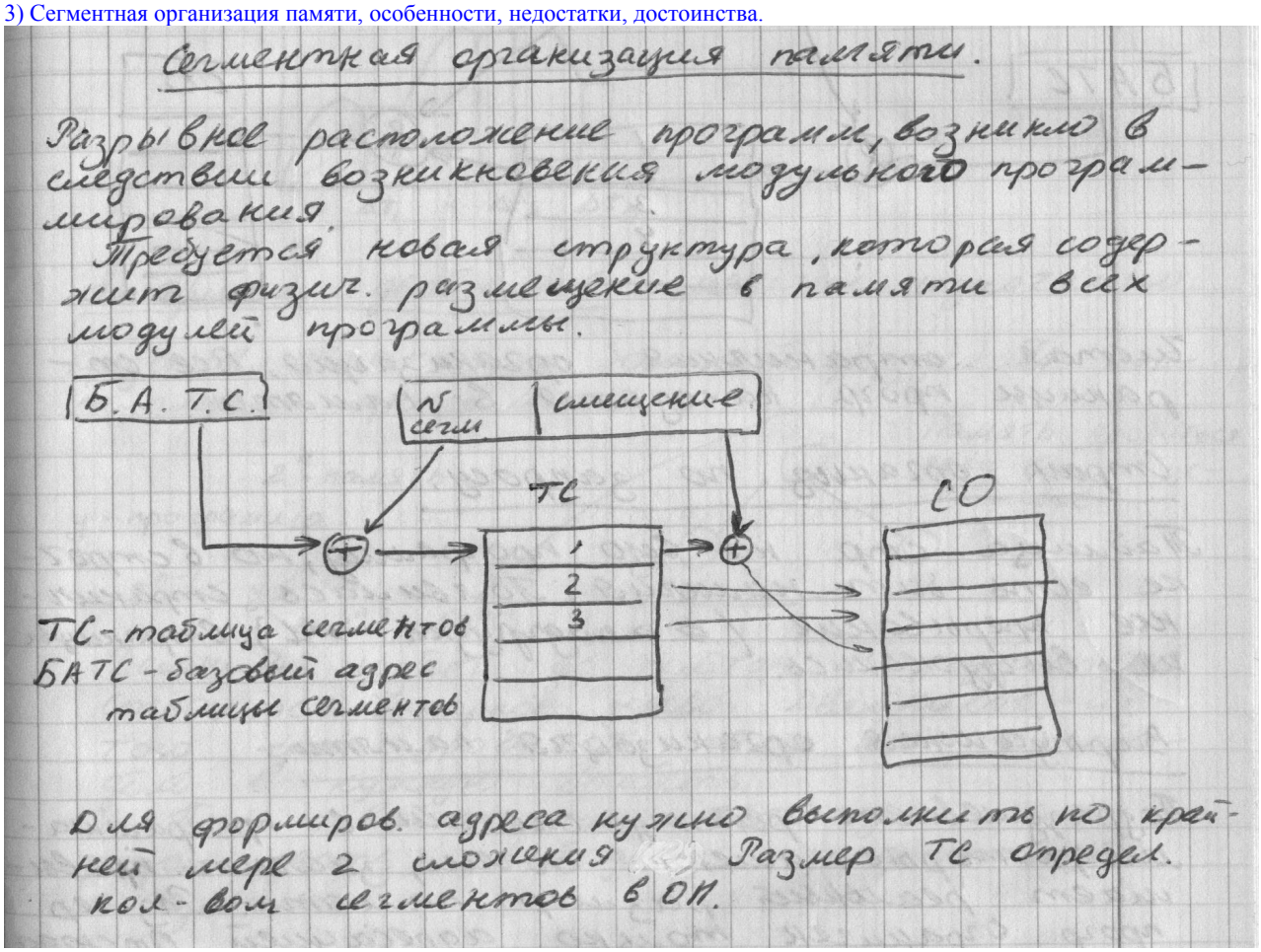
Доп. Инфа

Сколько входных очередей заданий формирует система ?

Процессы при поступлении в систему находятся в подготовленном состоянии и накапливаются во входных очередях заданий. Следует различать процессы, поступающие в систему и требующие безусловного выполнения в соответствии с одной из дисциплин обслуживания, и процессы, находящиеся в подготовленном состоянии после загрузки операционной системы. Процессы первого вида, как правило, принадлежат пользователям, а второго вида — ОС. Такие процессы находятся в подготовленном состоянии до тех пор, пока для выполнения функций системы они не будут активизированы. Пользовательские процессы активизируются или делается попытка их активизации при наличии в системе ресурсов. Попытка активизации процесса производится системой при наличии в системе оперативной памяти, достаточной для размещении программы, подчиненной процессу.

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

3) Сегментная организация памяти, особенности, недостатки, достоинства.



Сегментная организация памяти – модульное программирование. Каждый модуль требует непрерывной области памяти.

Возникает проблема защиты, требуется таблица сегментов, которая определяет место размещения сегментов, участвует в формировании исполнительного адреса.

Сегментная и сегментно-страничная организация памяти

Существуют две другие схемы организации управления памятью: сегментная и сегментно-страничная. *Сегменты*, в отличие от *страниц*, могут иметь переменный размер. Идея сегментации изложена во введении. При сегментной организации виртуальный адрес является двумерным как для программиста, так и для операционной системы, и состоит из двух полей – номера *сегмента* и смещения внутри *сегмента*. Подчеркнем, **что в отличие от страничной организации, где линейный адрес преобразован в двумерный операционной системой для удобства отображения, здесь двумерность адреса является следствием представления пользователя о процессе не в виде линейного массива байтов, а как набор сегментов переменного размера (данные, код, стек...)**.

Программисты, пишущие на языках низкого уровня, должны иметь представление о сегментной организации, явным образом меняя значения сегментных регистров (это хорошо видно по текстам программ, написанных на Ассемблере). Логическое *адресное пространство* – набор *сегментов*. Каждый *сегмент* имеет имя, размер и другие параметры (уровень привилегий, разрешенные виды обращений, флаги присутствия). В отличие от страничной схемы, где пользователь задает только один адрес, который разбивается на номер *страницы* и смещение прозрачным для программиста образом, в сегментной схеме пользователь специфицирует каждый адрес двумя величинами: именем *сегмента* и смещением.

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

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

В системах, где *сегменты* поддерживаются аппаратно, эти параметры обычно хранятся в таблице дескрипторов *сегментов*, а программа обращается к этим дескрипторам по номерам-селекторам. При этом в контекст каждого процесса входит набор сегментных регистров, содержащих селекторы текущих *сегментов* кода, стека, данных и т. д. и определяющих, какие

сегменты будут использоваться при разных видах обращений к памяти. Это позволяет процессору уже на аппаратном уровне определять допустимость обращений к памяти, упрощая реализацию защиты информации от повреждения и несанкционированного доступа.

Аппаратная поддержка *сегментов* распространена мало (главным образом на процессорах Intel). В большинстве ОС сегментация реализуется на уровне, не зависящем от аппаратуры.

Хранить в памяти *сегменты* большого размера целиком так же неудобно, как и хранить процесс непрерывным блоком. Напрашивается идея разбиения *сегментов* на *страницы*. При сегментно-страничной организации памяти происходит двухуровневая *трансляция* виртуального адреса в физический. В этом случае логический адрес состоит из трех полей: номера *сегмента* *логической памяти*, номера *страницы* внутри *сегмента* и смещения внутри *страницы*. Соответственно, используются две таблицы отображения – таблица *сегментов*, связывающая номер *сегмента* с таблицей *страниц*, и отдельная таблица *страниц* для каждого *сегмента*.

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.

Существуют две другие схемы организации управления памятью: сегментная и сегментно-страничная. *Сегменты*, в отличие от *страниц*, могут иметь переменный размер. Идея сегментации изложена во введении. При сегментной организации виртуальный адрес является двумерным как для программиста, так и для операционной системы, и состоит из двух полей – номера *сегмента* и смещения внутри *сегмента*. Подчеркнем, что в отличие от страничной организации, где линейный адрес преобразован в двумерный операционной системой для удобства отображения, здесь двумерность адреса является следствием представления пользователя о процессе не в виде линейного массива байтов, а как набор сегментов переменного размера (данные, код, стек...). Программисты, пишущие на языках низкого уровня, должны иметь представление о сегментной организации, явным образом меняя значения сегментных регистров (это хорошо видно по текстам программ, написанных на Ассемблере). Логическое *адресное пространство* – набор *сегментов*. Каждый *сегмент* имеет имя, размер и другие параметры (уровень привилегий, разрешенные виды обращений, флаги присутствия). В отличие от страничной схемы, где пользователь задает только один адрес, который разбивается на номер *страницы* и смещение прозрачным для программиста образом, в сегментной схеме пользователь специфицирует каждый адрес двумя величинами: именем *сегмента* и смещением. Каждый *сегмент* – линейная последовательность адресов, начинающаяся с 0. Максимальный размер *сегмента* определяется разрядностью процессора (при 32-разрядной адресации это 2^{32} байт или 4 Гбайт). Размер *сегмента* может меняться динамически (например, *сегмент* стека). В элементе таблицы *сегментов* помимо физического адреса начала *сегмента* обычно содержится и длина *сегмента*. Если размер смещения в виртуальном адресе выходит за пределы размера *сегмента*, возникает исключительная ситуация. Логический адрес – упорядоченная пара v=(s,d), номер *сегмента* и смещение внутри *сегмента*. В системах, где *сегменты* поддерживаются аппаратно, эти параметры обычно хранятся в таблице дескрипторов *сегментов*, а программа обращается к этим дескрипторам по номерам-селекторам. При этом в контекст каждого процесса входит набор сегментных регистров, содержащих селекторы текущих *сегментов* кода, стека, данных и т. д. и определяющих, какие *сегменты* будут использоваться при разных видах обращений к памяти. Это позволяет процессору уже на аппаратном уровне определять допустимость обращений к памяти, упрощая реализацию защиты информации от повреждения и несанкционированного доступа. Аппаратная поддержка *сегментов* распространена мало (главным образом на процессорах Intel). В большинстве ОС сегментация реализуется на уровне, не зависящем от аппаратуры. Хранить в памяти *сегменты* большого размера целиком так же неудобно, как и хранить процесс непрерывным блоком. Напрашивается идея разбиения *сегментов* на *страницы*. При сегментно-страничной организации памяти происходит двухуровневая *трансляция* виртуального адреса в физический. В этом случае логический адрес состоит из трех полей: номера *сегмента* *логической памяти*, номера *страницы* внутри *сегмента* и смещения внутри *страницы*. Соответственно, используются две таблицы отображения – таблица *сегментов*, связывающая номер *сегмента* с таблицей *страниц*, и отдельная таблица *страниц* для каждого *сегмента*. 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

4) Особенности управления файлами в ОС с FAT, недостатки, достоинства.
Ответ:

Доп.инфа:

Чтобы прочитать файл, программа, работающая в системе MS-DOS, должна сначала сделать системный вызов open, чтобы получить дескриптор файла. Системному вызову open в качестве одного из входных аргументов следует указать путь к файлу, который может быть как абсолютным, так и относительным (относительно текущего каталога). Файловая система открывает каталоги, перечисленные

в пути, один за другим, пока не обнаруживает последний каталог, который считывается в оперативную память. Затем в считанном каталоге ищется описатель файла, который требуется открыть.

Хотя каталоги в файловой системе MS-DOS переменного размера, используемые каталоговые записи, как и в CP/M, имеют фиксированный размер 32 байт. Формат описателя файла системы MS-DOS показан на рис. 6.29. В нем содержится имя файла, его атрибуты, дата и время создания, номер начального блока и точный размер файла. Имена файлов короче 8 + 3 символов выравниваются по левому краю полей и дополняются пробелами, каждое поле отдельно. Поле *Attributes* (атрибуты) представляет собой новое поле, содержащее биты, указывающие, что для файла разрешено только чтение, что файл должен быть заархивирован, что файл является системным или скрытым. Запись в файл, для которого разрешено только чтение, не разрешается. Таким образом осуществляется защита файлов от случайной записи или удаления. Бит *archived* (архивный) не устанавливается и не проверяется операционной системой MS-DOS. Он зарезервирован в описателе для архивирующих программ уровня пользователя, сбрасывающих этот бит при создании резервной копии файла, в то время как программы, модифицирующие файл, должны устанавливать этот бит. Таким образом архивирующая программа может определить, какие файлы подлежат архивации. Бит *hidden* (скрытый файл) позволяет избежать отображения файла в перечне файлов каталога. Основное его назначение заключается в том, чтобы скрыть от неопытных пользователей файлы, назначение которых им неизвестно. Наконец, бит *system* (системный) также скрывает файлы и защищает их от случайного удаления командой *del*. Этот бит установлен у основных компонентов системы MS-DOS.

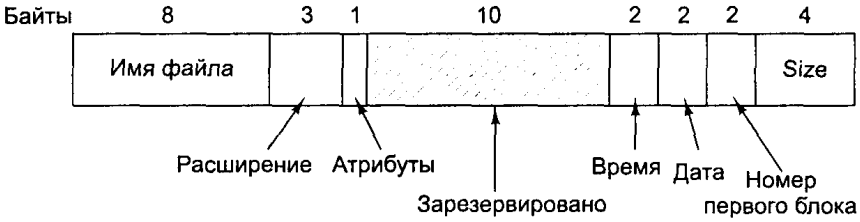


Рис. 6.29. Формат каталоговой записи в системе MS-DOS

В зависимости от количества блоков на диске в сист MS-DOS применяется три версии файл сист FAT: -12, -16, -32.

FAT-12 (макс размер диска 64 Мбайт, раздела - 16)

FAT-16 (макс размер диска 8 Гбайт, раздела - 2)

FAT-32 (макс размер диска 8 Тбайт, раздела - 2)

5) Виды прерываний и особенности их обработки.

5) Виды прерываний и особенности их обработки.

Прерывание — это нарушение последовательности выполнения действий (команд), т.е. после текущего действия (команды) выполняется не следующее (команда), а некоторое другое действие.

Классы прерываний

По своему назначению, причине возникновения прерывания делятся на различные **классы**. Традиционно выделяют следующие классы:

1. *Прерывания от схем контроля машины*. Возникают при обнаружении сбоев в работе аппаратуры, например, при несовпадении четности в микросхемах памяти.
2. *Внешние прерывания*. Возбуждаются сигналами запросов на прерывание от различных внешних устройств: таймера, клавиатуры, другого процессора и пр.

3. *Прерывания по вводу/выводу*. Инициируются аппаратурой ввода/вывода при изменении состояния каналов ввода/вывода, а также при завершении операций ввода/вывода.
4. *Прерывания по обращению к супервизору*. Вызываются при выполнении процессором **команды обращения к супервизору** (вызов функции операционной системы). Обычно такая команда инициируется выполняемым процессом при необходимости получения дополнительных ресурсов либо при взаимодействии с устройствами ввода/вывода.
5. *Программные прерывания*. Возникают при выполнении **команды вызова прерывания** либо при обнаружении ошибки в выполняемой команде, например, при арифметическом переполнении.

В последнее время принято прерывания 4 и 5 классов объединять в один класс программных прерываний, причем, в зависимости от источника, вызвавшего прерывание, среди них выделяют такие подтипы:

- прерывание вызванное исполнением процессором *команды перехода к подпрограмме обработки прерывания*
- прерывания, возникающие в результате *исключительной* (аварийной) *ситуации* в процессоре (деление на "0", переполнение и т.д.).

В связи с многообразием различных ВС и их постоянным развитием меняется и организация системы прерываний. Так, с появлением виртуальной памяти, появился класс страничных прерываний, который можно отнести и к классу исключительных ситуаций в процессоре; в системах с кэш-памятью существуют прерывания подкачки страниц в кэш-память и т.д.

Во время выполнения обработчика одного из прерываний возможно поступление другого сигнала прерывания, поэтому система должна использовать определенную *дисциплину обслуживания заявок* на прерывания. Обычно различным классам либо отдельным прерываниям присваиваются различные *абсолютные приоритеты*, т.о. при поступлении запроса с высшим приоритетом текущий обработчик снимается, а его место занимает обработчик вновь поступившего прерывания. Количество уровней вложенности прерываний называют **глубиной системы прерываний**. Обработчики прерываний либо дисциплина обслуживания заявок на прерывание должны иметь также специальные средства для случая, когда при обработке прерывания возникает запрос на обработку прерывания от того же источника.

((((((((((((((((((((((((С ШПОР

В системе присутствует 2 приоритетных уровня:

- Приоритет сигналов
  Используется полноупорядоченная схема сигналов (контроллер, его приоритет определяется путём запуска команды смены прерывания) либо частичноупорядоченная (когда прерываний много – все они разбиваются на классы, а потом производиться сканирование в полносвязной системе).

- Приоритет прерывания программ
  Более важный уровень, на нём выбирается программа с наивысшим приоритетом.

Дополнительные классы прерываний:

- Прерывания по таймеру
- Прерывания по кэш-промаху
- Страничное прерывание
- Прерывание по нажатию клавиши

))

РЕАЛИЗАЦИЯ МЕХАНИЗМА ПРЕРЫВАНИЙ

В КОМПЬЮТЕРАХ ТИПА IBM PC

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

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

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

Операционная система не контролирует содержимое таблицы векторов, хотя и имеются средства для чтения и изменения векторов. При помощи функций MS-DOS значение векторов можно изменить, поместив нужный адрес

непосредственно в таблицу векторов прерываний. В этом случае контроль за содержимым векторов и их соответствием размещению обработчиков в памяти полностью ложится на программиста, а последствия в случае ошибки могут быть самыми непредсказуемыми.

Активизация обработчика прерывания может произойти в результате возникновения следующих ситуаций:

- возникновение сигнала внутреннего прерывания внутри микросхемы процессора (**внутренние** или **логические прерывания**)
- поступление в процессор сигнала запроса на прерывание от внешнего (по отношению к процессору) устройства (**аппаратные прерывания**)
- выполнение процессором команды INT вызова процедуры обработки прерывания (**программные прерывания**)

2.9.1. Внутренние прерывания

Для обслуживания внутренних прерываний микропроцессоры i8086 использовали 5 первых векторов прерываний (00h-04h). Прерывания с номерами 05h-31h фирма Intel зарезервировала для использования в дальнейших разработках, однако фирма IBM при создании IBM PC использовала эти вектора по своему усмотрению. В последующих реализациях процессоров эти же вектора были задействованы для обработки новых внутренних прерываний, в результате чего стали возможны конфликты. Поскольку все добавленные прерывания используются в защищенном режиме процессора, то для избежания конфликтов при переходе в защищенный режим контроллер прерываний перепрограммируется таким образом, что при поступлении сигнала на линию IRQ0 вызывается процедура обработки с номером, отличным от 08h (обычно 50h или 60h). Для последующих линий IRQ вызываются процедуры с последующими (относительно базового для IRQ0) номерами.

Сигналы внутренних прерываний возникают при нарушениях в работе самого микропроцессора либо при ошибках в выполняемых командах и формируются внутри схемы микропроцессора при возникновении одной из ситуаций, указанных в Табл. 2.2.

2.9.2. Аппаратные прерывания

Аппаратные прерывания инициируются различными устройствами компьютера, внешними по отношению к процессору (системный таймер, клавиатура, контроллер диска и пр.). Эти устройства формируют сигналы запросов на прерывания (IRQ), поступающие на **контроллер прерываний**.

. Немаскируемые прерывания

Существует одно аппаратное прерывание, отличное от всех остальных. Это **немаскируемое прерывание** *(Non-Maskable Interrupt, NMI).* Его отличие состоит в том, что хоть инициатором его и является внешнее устройство, сигнал запроса поступает в процессор, минуя контроллер прерываний. Поэтому такой запрос на прерывание невозможно замаскировать. Кроме этого, запрос по линии NMI вызывает немедленную реакцию процессора и имеет максимальный приоритет в системе, хотя обработчик NMI может быть прерван любым маскируемым прерыванием, если их разрешить командой STI. Это позволяет использовать данное прерывание в особо критических ситуациях: обычно оно используется при падении напряжения питания и при ошибках памяти, однако на некоторых моделях IBM — совместимых компьютеров NMI используется также для обслуживания сопроцессора, клавиатуры, каналов ввода/вывода, дисковых контроллеров, часов реального времени, таймеров, контроллеров прямого доступа к памяти и пр.

В процессорах i80286 и выше в случае, если во время обработки NMI возникнет повторный запрос на немаскируемое прерывание, он не прервет выполнение текущего обработчика, а запомнится и будет обработан после завершения текущего обработчика. Если повторных запросов во время выполнения обработчика будет несколько, то сохранится только первый, последующие будут проигнорированы. При возврате из обработчика командой IRET процессор не переходит на выполнение следующей команды, а пытается выполнить команду по тому же адресу, что привел к возникновению критической ситуации.

2.9.2.3. Программные прерывания

Программные прерывания инициируются специальной командой процессору INT n, где n указывает номер прерывания, обработчик которого необходимо вызвать. При этом в стек заносится содержимое регистров флагов, указателя команд и сегмента кода, а флаг разрешения прерывания сбрасывается (аналогично тому, как при обработке аппаратных прерываний).

Далее в процессор загружается новое PSW, значение которого содержится в векторе прерывания с номером n, и управление переходит к обработчику прерывания. Требования к обработчику программного прерывания такие же, как и к обработчику аппаратного прерывания, за исключением того, что команда завершения аппаратного прерывания (EOI) не требуется. Обработка прерывания завершается командой IRET, восстанавливающей из стека старое PSW.

Обработка программного прерывания является более мягким видом прерывания по отношению к прерываемой программе, нежели обработка аппаратного прерывания, т.к. аппаратное прерывание выполняется по требованию самой прерываемой программы, и она ожидает от этого какого-либо результата. Аппаратное же прерывание получает управление безо всякого ведома выполняющейся программы и зачастую не оказывает на нее никакого влияния.