1 Состав операционной системы. Назначение отдельных частей.

Операционная система состоит из СПО, которое разделяется на обрабатывающее и управляющее.
Обрабатывающие(драйвера, компиляторы, редакторы) – изменяют инфу поступающую им на вход.
Управляющие проги(не изменяют информацию, но обеспечивают основные фунцкии):

А)управление заданиями – отслеживает прохождение задания от входа до выхода.

Б)Блок управления задачами или процессами – следит за выполнением задачи связянной с ресурсами системы.

В)Блок управления памятью – выполняет операции по использованию памяти всеми программами в соответствии с организацией памяти(физическая реализация памяти в системе)

Г)Управление данными (файловая система)

Д)Управление внешними устройствами.

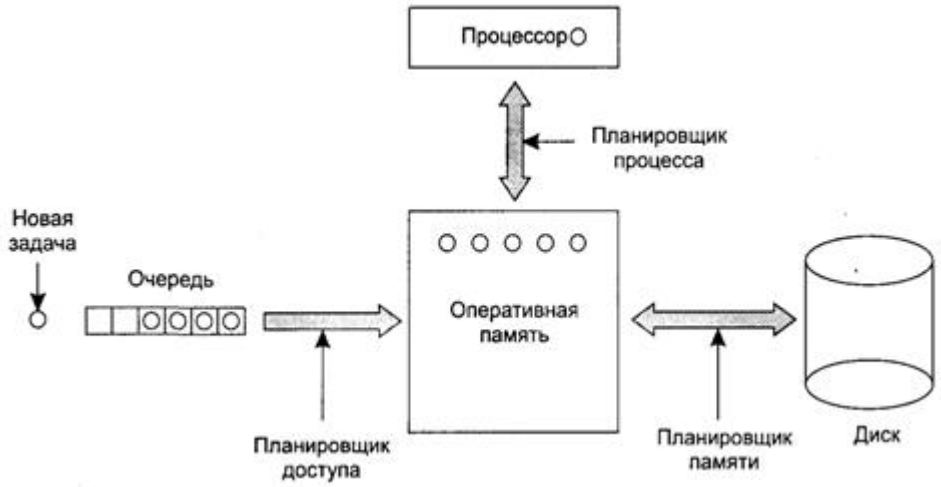
Современные ОС имеют сложную структуру, каждый элемент которой выполняет определенные функции по управлению компьютером.

- Управление файловой системой. Процесс работы компьютера сводится к обмену файлами между устройствами. В ОС имеются программные модули, управляющие файловой системой.
- Командный процессор. В состав ОС входит специальная программа – командный процессор, – которая запрашивает у пользователя команды (запуск программы, копирование, удаление и т.д.) и выполняет их.
- Драйверы устройств. В состав ОС входят драйверы устройств, специальные программы, которые обеспечивают управление работой устройств и согласование информационного обмена с другими устройствами, позволяют производить настройку параметров устройств. Каждому устройству соответствует свой драйвер. Программы (драйверы) для поддержки наиболее распространенных устройств входят в Windows, а для остальных устройств поставляются вместе с этими устройствами или контроллерами. Можно самому установить или переустановить драйвер.
- Графический интерфейс. Для упрощения работы пользователя в состав ОС входят программные модули, создающие графический пользовательский интерфейс. Пользователь может вводить команды с помощью мыши, а не с клавиатуры.
- Сервисные программы. (утилиты). Позволяют обслуживать диски (проверять, сжимать, дефрагментировать), выполнять операции с файлами (архивировать и т.д.), работать в сетях и т.д.
- Справочная система. Позволяет оперативно получить необходимую информацию о функционировании ОС.

2 Этапы прохождения задачи через ВС. Модификация задания и его отображение в вычислительной среде.

Задание – это внешняя еденица работы системы, для которой система ресурсов не выделяет.

Описана трехуровневая модель.



1 уровень. Задание (см. определения выше) кто-то формирует. Программист или система. В первом случае, если задание правильно оформлено (синтаксис), оно накапливается в очереди входных заданий.

Для выбора заданий из очереди может использоваться алгоритм, в котором устанавливается приоритет коротких задач перед длинными. Впускной/входной планировщик волен придержать некоторые задания во входной очереди, а пропустить задание, поступившее позже остальных.

Т.е. на 1 уровне происходит фильтрация заданий, но ресурсы к ним не определяются. В задании должна быть указана программа и данные. В результате, имеем очередь входных заданий.

2 уровень. Если в системе есть свободные ресурсы (см. определение выше), то всплывает планировщик второго уровня. Он выбирает из очереди задание, определяет, можно ли его решить с помощью ресурсов, что есть в системе. Если можно, то вызывает *инициализатор/инициатор*, который как только определит ресурсы и расшифрует задание, сформирует блок управления задачей/процессом (*PCB*).

Так исторически сложилось, что сначала был TCB (task control block), а потом появился программный режим работы, и появились процессы и PCB.

Таким образом, задание превращается в задачу/процесс. Возможна ситуация, когда процессов слишком много и они все в памяти не помещаются, тогда некоторые из них будут выгружены на диск. Второй уровень планирования определяет, какие процессы можно хранить в памяти, а какие — на диске. Этим занимается *планировщик памяти*.

Для оптимизации эффективности системы планировщик памяти должен ре шить, сколько и каких процессов может одновременно находиться в памяти. Кол-во процессов, одновременно находящихся в памяти, называется *степенью многозадачности.*

Планировщик памяти периодически просматривает процессы, находящиеся на диске, чтобы решить, какой из них переместить в память. Среди критериев, используемых планировщиком, есть следующие:

1. Сколько времени прошло с тех пор, как процесс был выгружен на диск или загружен с диска?
2. Сколько времени процесс уже использовал процессор?
3. Каков размер процесса (маленькие процессы не мешают)?
4. Какова важность процесса?

3 уровень. Как только процессор освобожден (либо естественно либо с вытеснением), срабатывает планировщик 3-го (верхнего уровня, он же *планировщик процессора*) и из готовых процессов/задач он должен выбрать процесс/задачу для выполнения и занять время выполнение в процессоре.

4 уровень. Собственно, его можно не выделять отдельно. На последнем этапе результаты выполнения могут быть сохранены или выведены на внешний носитель.

Если в системе есть ресурсы то всплывает планировщик второго уровня (выбор из очереди задания; определить можно ли его решить с помощью ресурсов, что есть в системе; вызвать инициализатор который создает блок управления задачей.)

Признаком подкачки нового задания является либо освобождение опер. памяти либо резкое уменьшение эффективной работы вычислительной установки.

Как только процессор освобожден либо естественно либо с вытеснением сработывает планировщик 3-го (верхнего уровня) и из готовых задач он должен выбрать зад. Для выполнения и занять время выполнение в процессоре.

Особенности формирования исполнительного адреса в различных видах организации памяти. Участие GDT u LDT.

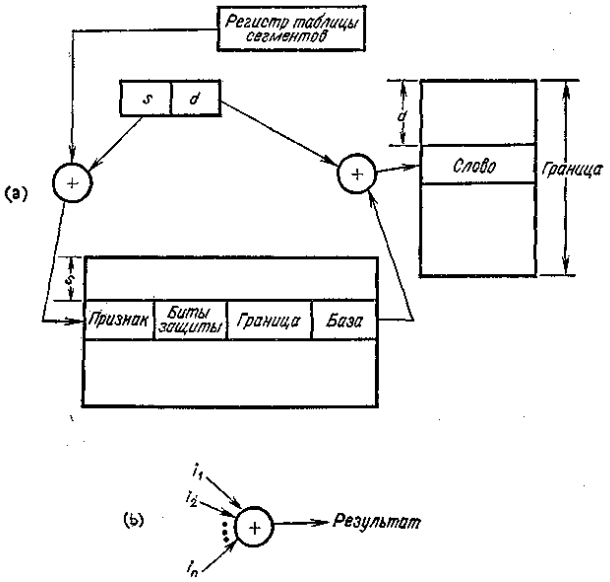
В системе с сегментацией всякий адрес представляет собой пару *Is, d],* где *s* — имя сегмента, *ad* — смещение. Каждому заданию или процессу соответствует всегда присутствующая в памяти *таблица сегментов,* в которой каждому сегменту данного задания соответствует одна запись. С помощью этой таблицы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

признак (или *бит присутствия),* показывающий, присутствует ли s-й сегмент в данный момент в памяти;

базовый адрес s-го сегмента;

граница, указывающая количество ячеек, занимаемых данным сегментом;

4) биты защиты (не обязательные), используемые для контроля способа доступа.



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

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

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

В самом простом и наиболее распространенном случае страничной организации памяти (или paging) как логическое адресное пространство, так и физическое представляются состоящими из наборов блоков или страниц одинакового размера. При этом образуются логические страницы (page), а соответствующие единицы в физической памяти называют страничными кадрами (page frames). Страницы (и страничные кадры) имеют фиксированную длину, обычно являющуюся степенью числа 2, и не могут перекрываться. Каждый кадр содержит одну страницу данных. При такой организации внешняя фрагментация отсутствует, а потери из-за внутренней фрагментации, поскольку процесс занимает целое число страниц, ограничены частью последней страницы процесса. Логический адрес в страничной системе – упорядоченная пара (p,d), где p – номер страницы в виртуальной памяти, а d – смещение в рамках страницы p, на которой размещается адресуемый элемент. Заметим, что разбиение адресного пространства на страницы осуществляется вычислительной

системой незаметно для программиста. Поэтому адрес является двумерным лишь с точки зрения операционной системы, а с точки зрения программиста адресное пространство процесса остается линейным.
Описываемая схема позволяет загрузить процесс, даже если нет непрерывной области кадров, достаточной для размещения процесса целиком. Но одного базового регистра для осуществления трансляции адреса в данной схеме недостаточно. Система отображения логических адресов в физические сводится к системе отображения логических страниц в физические и представляет собой таблицу страниц, которая хранится в оперативной памяти. Иногда говорят, что таблица страниц – это кусочно-линейная функция отображения, заданная в табличном виде.
Интерпретация логического адреса показана на рис. 8.7. Если выполняемый процесс обращается к логическому адресу v = (p,d), механизм отображения ищет номер страницы p в таблице страниц и определяет, что эта страница находится в страничном кадре p', формируя реальный адрес из p' и d.

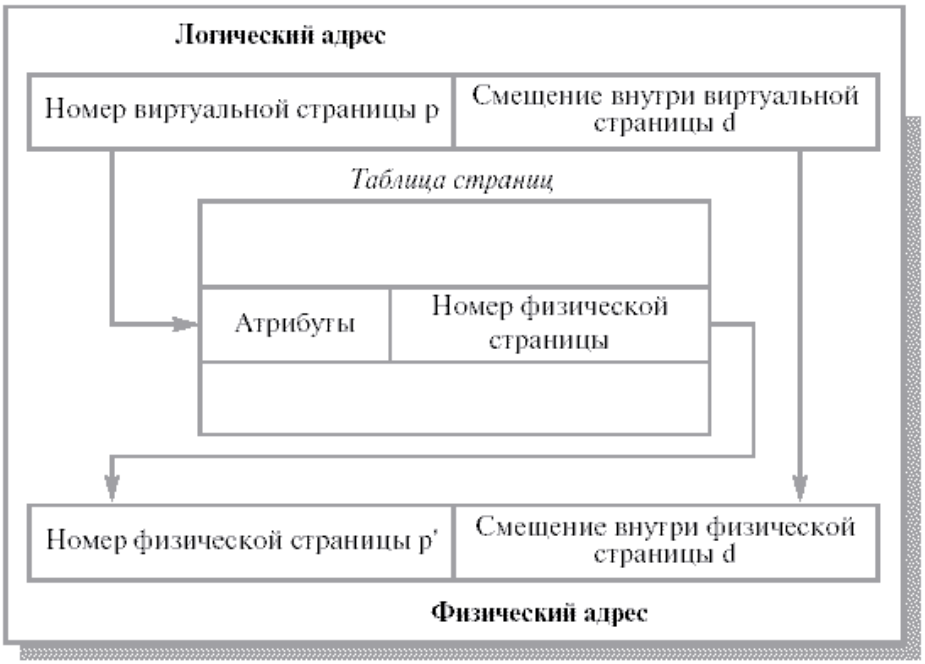


Рис. 8.7. Связь логического и физического адресов при страничной организации памяти
Таблица страниц (page table) адресуется при помощи специального регистра процессора и позволяет определить номер кадра по логическому адресу. Помимо этой основной задачи, при помощи атрибутов, записанных в строке таблицы страниц, можно организовать контроль доступа к конкретной странице и ее защиту.
Отметим еще раз различие точек зрения пользователя и системы на используемую память. С точки зрения пользователя, его память – единое непрерывное пространство, содержащее только одну программу. Реальное отображение скрыто от пользователя и контролируется ОС. Заметим, что процессу пользователя чужая память недоступна. Он не имеет возможности адресовать память за пределами своей таблицы страниц, которая включает только его собственные страницы.
Для управления физической памятью ОС поддерживает структуру таблицы кадров. Она имеет одну запись на каждый физический кадр, показывающий его состояние.
Отображение адресов должно быть осуществлено корректно даже в сложных случаях и обычно реализуется аппаратно. Для ссылки на таблицу процессов используется специальный регистр. При переключении процессов необходимо найти таблицу страниц нового процесса, указатель на которую входит в контекст процесса.

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

памяти, что описана в соответствующих GDT и LDT. Программа не ничего не знает и не заботится о том, где именно располагается тот или иной сегмент в физической памяти. Сегментация в микропроцессорах Intel основана на использовании двух таблиц: **GDT** (Global Descriptor Table) и **LDT** (Local Descriptor Table). В GDT содержатся дескрипторы сегментов, которые используются системой. LDT может создаваться для каждого процесса и содержит сегменты, которые он использует.

В элементе таблицы сегментов, помимо физического адреса начала сегмента (если виртуальный сегмент содержится в основной памяти), содержится размер сегмента.

Если размер смещения в виртуальном адресе выходит за пределы размера сегмента, возникает прерывание.

Для того что бы загрузить дескриптор используется **селектор**, следующего вида:

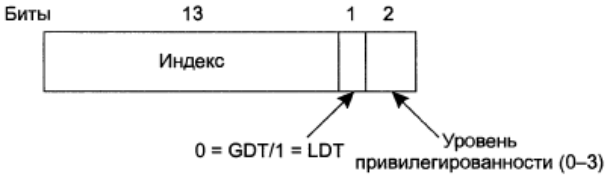
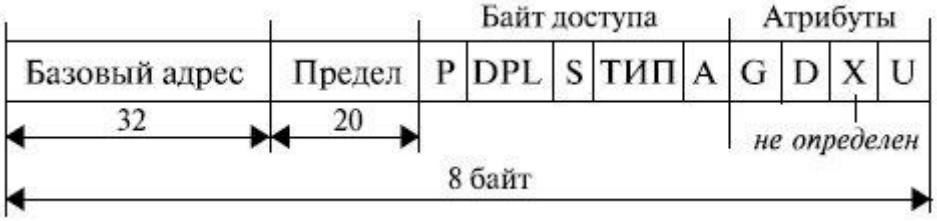


Рис. 4.23. Селектор в системе Pentium

Второй бит которого, показывает в какой из таблиц находится дескриптор (в локальной или глобальной), а следующие 13 - номер дескриптора в таблице.

Дескриптор сегмента имеет следующую структуру:



Базовый адрес – позволяет определить начальный адрес сегмента в любой точке адресного пространства в 2^32 байт (4 Гбайт).

Поле предела (limit) – указывает длину сегмента - 1

G (Granularity – гранулярность) – единица измерения длины сегмента (0 – байты, 1 – страницы (4 кб))

Бит размерности (Default size) – определяет длину адресов и операндов, используемых в команде по умолчанию (0 – 16 разрядов, 1 – 32 разряда)

Байт доступа определяет основные правила обращения с сегментом.

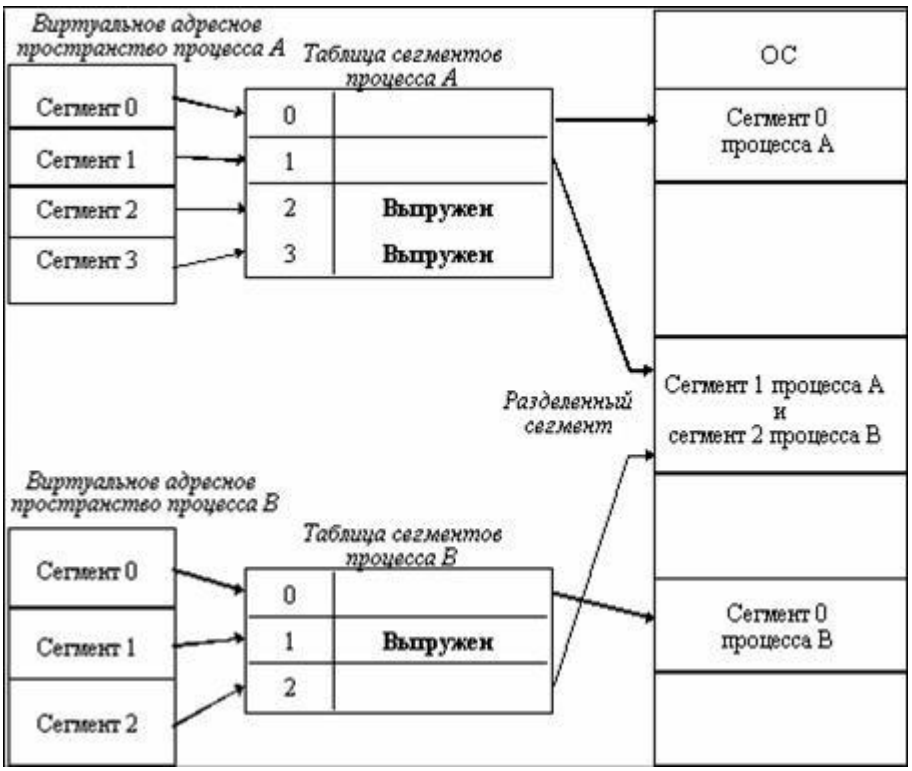
Бит присутствия P (Present) показывает возможность доступа к сегменту (если он равен 1 – сегмент находится в ОП).

Двухразрядное поле DPL (Descriptor Privilege Level) указывает один из четырех возможных (от 0 до 3) уровней привилегий дескриптора, определяющий возможность доступа к сегменту со стороны тех или иных программ (уровень 0 соответствует самому высокому уровню привилегий).

Бит обращения A (Accessed) устанавливается в "1" при любом обращении к сегменту.

Поле типа в байте доступа определяет назначение и особенности использования сегмента. Если бит S (System - бит 4 байта доступа) равен 1, то данный дескриптор описывает реальный сегмент памяти. Если S = 0, то этот дескриптор описывает специальный системный объект, который может и не быть сегментом памяти, например, шлюз вызова, используемый при переключении задач, или дескриптор локальной таблицы дескрипторов LDT.

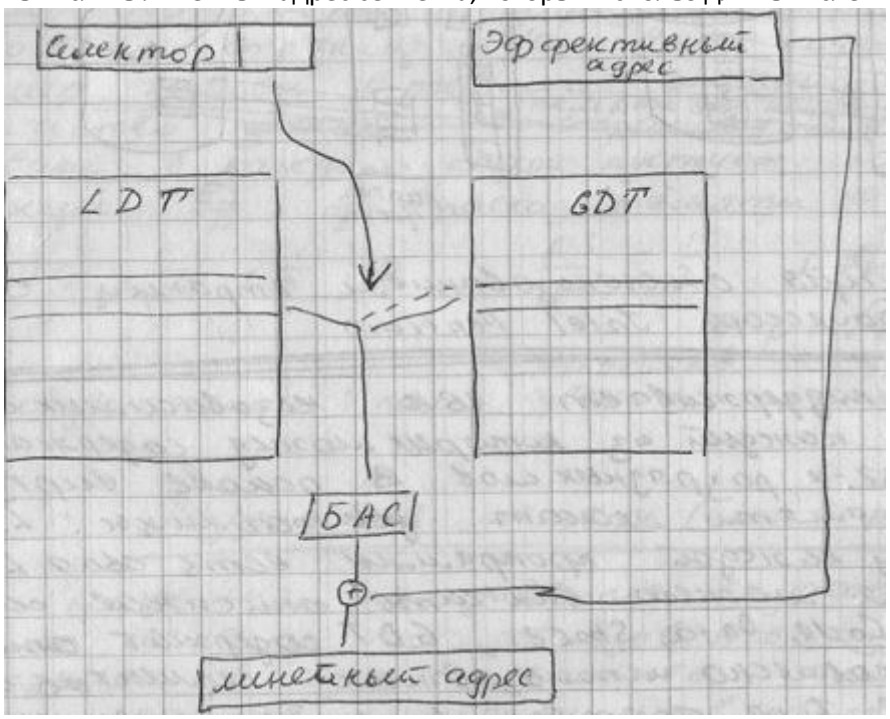
На картинке показано, как программы разбиваются на сегменты. Разные программы могут использовать один сегмент (в котором, например, содержится динамически подключаемая библиотека).



Отрывки из конспекта:

В основе виртуальной памяти лежат 2 таблицы: ЛДТ и ГДТ. У каждой проги есть своя ЛДТ которая может включить описания сегмента. ГДТ содержит описание совместно использованых сегментов включая ОС. Для доступа к любому сегменту система должна загрузить селектор этого сегмента (16 бит). Для поиска ГДТ и ЛДТ используются регистры ГДТР и ЛДТР. ТР – содержит указатель на TSS текущей задачи. В ТСС сохраняется контекст задачи при смене задач.

Дескриптор нахоится в ГДТ или ЛДТ. В соответствии с селектором, а именно индексом требуется вычислить линейный адрес сегмента, который использ для вычисления физического адреса.



TSS имеет динамиическую структуру и минимальный размер 104 байта – все основные регистры процессора. Может сюда добавить регистры прогер. Если стоит признак ЛДТ то сначало надо найти адрес ЛДТ, а потом к ней обращатся. ЛДТ можно найти только через ГДТ.

Особенности управления файлами в ОС CP/M. В какой файловой системе и что использовано от нее

Прямое указание на кластер, где записан файл – с точки зрения надежности – самая приличная система (впервые реализовано в CP/M)

Стоит взглянуть на операционную систему CP/M по нескольким причинам. Во-первых, исторически это была очень важная система, ставшая прямым предшественником системы MS-DOS. Во-вторых, разработчики сегодняшних операционных систем и систем будущего, полагающие, что компьютеру требуется 32 Мбайт памяти, только чтобы загрузить в нее операционную систему, могут поучиться простоте системы, которой вполне хватало 16 Кбайт ОЗУ. В-третьих, в ближайшие десятилетия широкое применение получат встроенные системы. В связи с ограничениями в цене, размерах, весе и потребляемой мощности операционные системы, используемые, например, в часах, видео- и фотокамерах, радиоприемниках и сотовых телефонах, обязательно должны быть компактными, подобно операционной системе CP/M. Конечно, у этих систем не будет 8-дюймовых гибких дисков, но они вполне могут пользоваться электронными дисками во флэш-ОЗУ, на которых несложно организовать файловую систему, подобную CP/M.

Распределение памяти в системе CP/M показано на рис. 6.27. Наверху оперативной памяти (в ОЗУ) располагается базовая система ввода-вывода BIOS, содержащая базовую библиотеку — 17 вызовов ввода-вывода, используемых системой CP/M (в данном разделе мы рассмотрим CP/M 2.2, являвшуюся стандартной версией, когда CP/M была на вершине популярности). Эти системные вызовы предназначены для чтения и записи с гибкого диска, ввода с клавиатуры и вывода на экран.

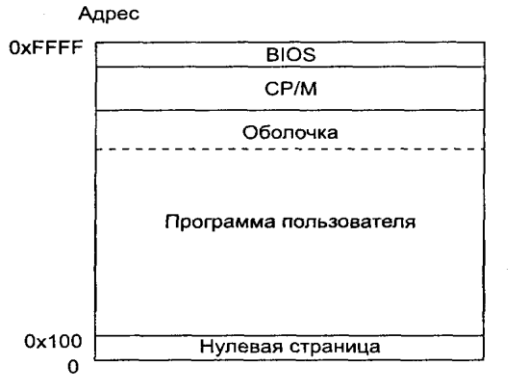


Рис. 6.27. Распределение памяти в системе CP/M

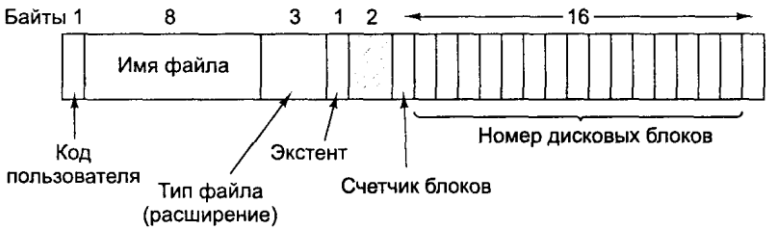


Рис. 6.28. Формат каталоговой записи в системе CP/M

Сразу под BIOS располагается сама операционная система. Для версии CP/M 2.2 ее размер составляет 3584 байт. Невероятно, но факт: вся операционная система занимала менее 4 Кбайт. Под операционной системой размещается оболочка (обработчик командных строк), съедающая еще 2 Кбайт. Остальная память используется для программ пользователя, кроме нижних 256 байт, зарезервированных для векторов аппаратных прерываний, некоторых переменных и буфера для текущей командной строки, в котором к ней могут получить доступ программы пользователя.

Причина, по которой система BIOS отделена от самой операционной системы CP/M (хотя обе системы располагаются в ОЗУ), заключается в переносимости. Операционная система CP/M взаимодействует с аппаратурой только с помощью обращений к BIOS. Для переноса системы CP/M на новую машину нужно всего лишь перенести туда BIOS. Когда это сделано, сама CP/M может быть установлена без изменений.

В файловой системе CP/M всего один каталог, содержащий записи фиксированного размера (32 байт). Размер каталога, фиксированный для данной реализации, может отличаться в других реализациях системы CP/M. В этом каталоге перечисляются все файлы системы. После загрузки система считывает каталог и рассчитывает битовый массив занятых и свободных блоков. Этот битовый массив, размер которого для 180-килобайтного диска составляет всего 23 байта, постоянно хранится в оперативной памяти. После завершения работы операционной системы он не сохраняется на диске. Благодаря такому подходу исчезает необходимость в проверке[1] непротиворечивости файловой системы на диске (вроде той, что выполняет программа *fsck* в UNIX) и сохраняется на диске один блок (в процентном отношении это эквивалентно сохранению 90 Мбайт на современном 16-гигабайтном диске).

Когда пользователь набирает команду, оболочка сначала копирует ее в буфер в нижние 256 байт памяти. Затем она ищет вызываемую программу и загружает ее в память по адресу 256 (над вектором прерываний), после чего передает управление по этому адресу. Программа начинает работу. Она обнаруживает свои параметры в буфере командной строки. Программе разрешается использовать память, занимаемую оболочкой, если ей нужно много памяти. Закончив работу, программа выполняет системный вызов CP/M, сообщая операционной системе, что следует перезагрузить оболочку (если занимаемая ею память использовалась программой) и запустить ее. В двух словах, вот, собственно, и весь рассказ об операционной системе CP/M.

5. Значение приоритетных уровней 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.

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

Контроллер прерываний можно запрограммировать для работы в нескольких режимах:

1. Режим фиксированных приоритетов *(Fixed Priority, Fully Nested Mode).* В этом режиме запросы прерываний имеют жесткие приоритеты от 0 (высший) до 7 (низший) и обрабатываются в соответствии с этими приоритетами. Если запрос с меньшим приоритетом возникает во время обработки запроса с более высоким приоритетом, то он игнорируется. BIOS при включении питания и при перезагрузке программирует контроллер прерываний на работу именно в этом режиме. В дальнейшем, режим при необходимости, можно изменить программным способом.

Табл. 2.2. Внутренние прерывания процессоров i80x86

Исключительная ситуация	Номер прерывания
Деление на 0	00h
Завершение выполнения очередной команды процессора при установленном флаге TF (используется в режиме отладки программ)	01h
Особая ситуация в режиме отладки (80386+)	01h
Выполнение команды INTO при установленном флаге OF (используется при обнаружении переполнения)	04h
Выход проверяемой величины за пределы диапазона при выполнении команды BOUND (80186+)	05h
Попытка выполнения недопустимой операции (например привилегированной команды в реальном режиме) (80286+)	06h
Попытка выполнения команды сопроцессора при отсутствии последнего (80286+)	07h
Возникновение двух исключительных ситуаций в одной команде (80286+)	08h
Попытка сопроцессора получить доступ к памяти за границами сегмента (80286,80386)	09h
Попытка переключиться на TSS, содержащий неверную информацию (80286+)	0Ah
Обращение к сегменту, выгруженному на диск из оперативной памяти (80286+)	0Bh
Переполнение стека (80286+)	0Ch
Отказ общей защиты (80286+)	0Dh
Ошибка обращения к странице виртуальной памяти (80386+)	0Eh
Обращение по неправильно выровненному адресу памяти (486+)	11h

1. Режим с автоматическим сдвигом приоритетов *(Automatic Rotation).* В этом режиме после обработки прерывания данному уровню присваивается минимальный приоритет, а все остальные приоритеты изменяются циклически таким образом, что запросы следующего за только что обработанным уровнем имеют наивысший, а предыдущего — наинизший приоритет. Например, после обработки запроса уровня 6 приоритеты устанавливаются следующим

образом: уровень 7 имеет приоритет 0, уровень 8 — приоритет 1 и т.д., уровень 5 — приоритет 6 и уровень 6 — приоритет 7. Таким образом, в этом режиме всем запросам дается возможность получить обработку.

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