БИЛЕТ № 25

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.

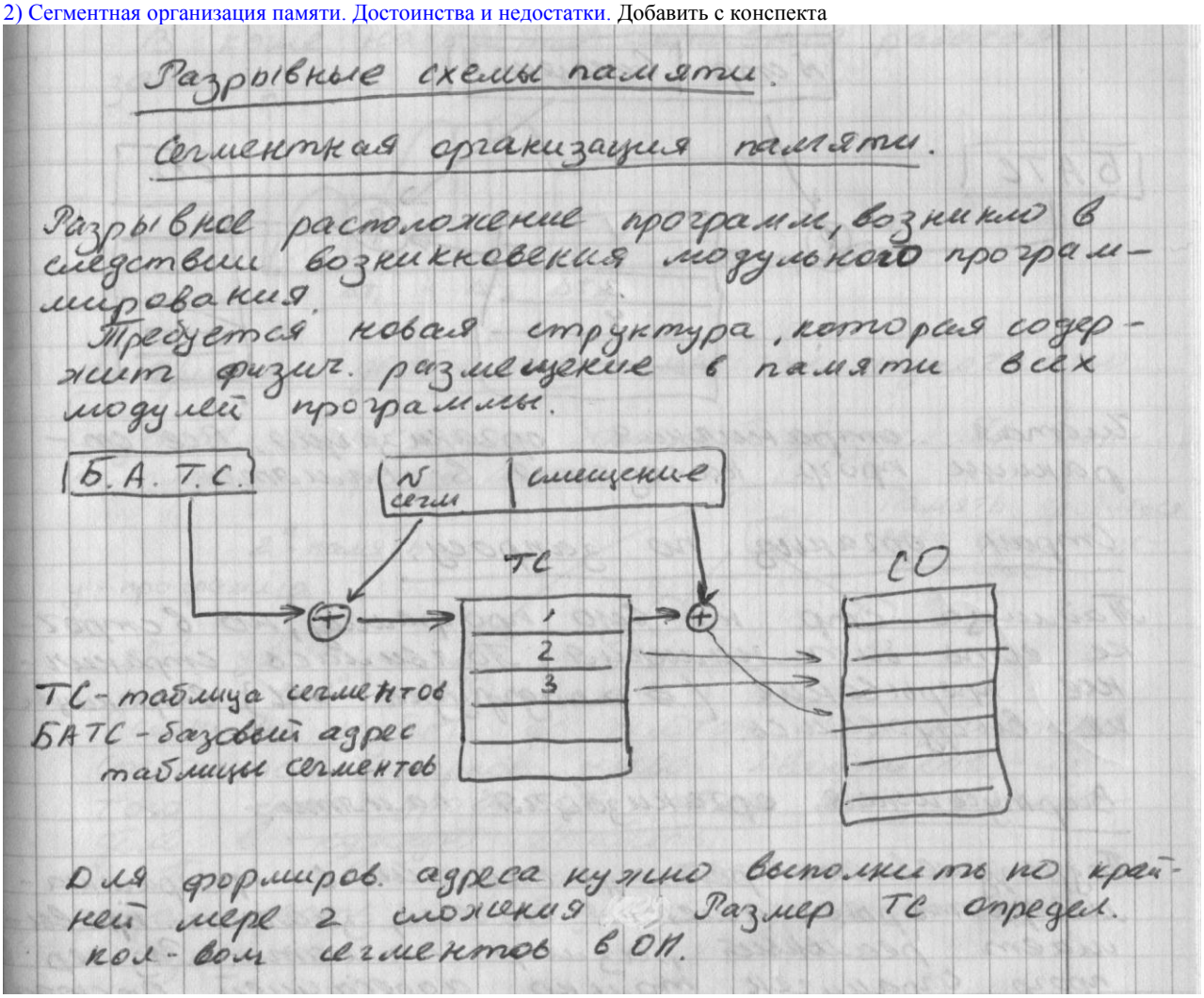
Выполнение программ ядра может осуществляться двумя способами:

- 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) Сегментная организация памяти. Достоинства и недостатки. Добавить с конспекта



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

3) Особенности организации файловой системы FAT. Достоинства и недостатки.

Связный список при помощи таблицы в памяти

Оба недостатка предыдущей схемы организации файлов в виде связных списков могут быть устранены, если указатели на следующие блоки хранить не прямо в блоках, а в отдельной таблице, загружаемой в память. На рис. 6.11 показан внешний вид такой таблицы для файлов с рис. 6.10. На обоих рисунках показаны два файла. Файл *A* использует блоки диска *4*, *7*, *2*, *10* и *12*, а файл *B* использует блоки диска *6*, *3*, *11* и *14*. С помощью таблицы, показанной на рис. 6.11, мы можем начать с блока *4* и следовать по цепочке до конца файла. То же может быть сделано для второго файла, если начать с блока *6*. Обе цепочки завершаются специальным маркером (например –1), не являющимся допустимым номером блока. Такая таблица, загружаемая в оперативную память, называется **FAT-таблицей** (File Allocation Table — таблица размещения файлов).

Рис. 6.11. Таблица размещения файлов

Эта схема позволяет использовать для данных весь блок. Кроме того, случай ный доступ при этом становится намного проще. Хотя для получения доступа к какому-либо блоку файла все равно понадобится проследовать по цепочке по всем ссылкам вплоть до ссылки на требуемый блок, однако в данном случае вся цепоч ка ссылок уже хранится в памяти, поэтому для следования по ней не требуются дополнительные дисковые операции. Как и в предыдущем случае, в каталоге до статочно хранить одно целое число (номер начального блока файла) для обеспе чения доступа ко всему файлу.

Основной недостаток этого метода состоит в том, что вся таблица должна по стоянно находиться в памяти. Для 20-гигабайтного диска с блоками размеро 1 Кбайт потребовалась бы таблица из 20 млн записей, по одной для каждого и 20 млн блоков диска. Каждая запись должна состоять как минимум из трех бай тов[1]. Для ускорения поиска размер записей должен быть увеличен до 4 бай Таким образом, таблица будет постоянно занимать 60 или 80 Мбайт оператив ной памяти. Таблица, конечно, может быть размещена в виртуальной памяти, н и в этом случае ее размер оказывается чрезмерно большим, к тому же постоянна выгрузка таблицы на диск и загрузка с диска существенно снизит производитель ность файловых операций.

4) Структура прохождения задания через вычислительную систему.

Создание процесса

Операционной системе необходим способ, позволяющий удостовериться в наличии всех необходимых процессов. В простейших системах, а также системах, разработанных для выполнения одного-единственного приложения (например, контроллер микроволновой печи), можно реализовать такую ситуацию, в которой все процессы, которые когда-либо могут понадобиться, присутствуют в системе при ее загрузке. В универсальных системах необходим способ создания и прерывания процессов по мере необходимости. В этом разделе мы рассмотрим некоторые из возможных способов решения этой проблемы. Ниже перечислены четыре основных события, приводящие к созданию процессов.

1. Инициализация системы.
2. Выполнение изданного работающим процессом системного запроса на создание процесса.
3. Запрос пользователя на создание процесса.
4. Инициирование пакетного задания.

Обычно при загрузке операционной системы создаются несколько процессов. Некоторые из них являются высокоприоритетными процессами, то есть обеспечивающими взаимодействие с пользователем и выполняющими заданную работу. Остальные процессы являются фоновыми, они не связаны с конкретными пользователями, но выполняют особые функции. Например, один фоновый процесс может быть предназначен для обработки приходящей на компьютер почты,

активизируясь только по мере появления писем. Другой фоновый процесс может обрабатывать запросы к web-страницам, расположенным на компьютере, и активизироваться для обслуживания полученного запроса. Фоновые процессы, связанные с электронной почтой, web-страницами, новостями, выводом на печать и т. п., называются **демонами.** В больших системах насчитываются десятки демонов. В UNIX для вывода списка запущенных процессов используется программа ps. В Windows 95/98/Me достаточно нажать CTRL-ALT-DEL, а в Windows 2000 можно воспользоваться диспетчером задач, вызываемым этой же комбинацией трех клавиш.

Процессы могут создаваться не только в момент загрузки системы, но и позже. Например, новый процесс (или несколько) может быть создан по просьбе текущего процесса. Создание новых процессов особенно полезно в тех случаях, когда выполняемую задачу проще всего сформировать как набор связанных, но тем не менее независимых взаимодействующих процессов. Если необходимо организовать выборку большого количества данных из сети для дальнейшей обработки, удобно создать один процесс для выборки данных и размещения их в совместно используемом буфере, в то время как второй процесс будет считывать данные из буфера и обрабатывать их. Эта схема даже ускорит обработку данных, если каждый процесс запустить на отдельном процессоре в случае многопроцессорной системы.

В интерактивных системах пользователь может запустить программу, набрав на клавиатуре команду или дважды щелкнув на значке программы. В обоих случаях результатом будет создание нового процесса и запуск в нем программы. Когда на UNIX работает X Windows, новый процесс получает то окно, в котором был запущен. В Microsoft Windows процесс не имеет собственного окна при запуске, но он может (и должен) создать одно или несколько окон. В обеих системах пользователь может одновременно открыть несколько окон, каждому из которых соответствует свой процесс. Пользователь может переключаться между окнами с помощью мыши и взаимодействовать с процессом, например, вводя данные по мере необходимости.

Последнее событие, приводящее к созданию нового процесса, связано с системами пакетной обработки на больших компьютерах. Пользователи посылают пакетное задание (возможно, с использованием удаленного доступа), а операционная система создает новый процесс и запускает следующее задание из очереди в тот момент, когда освобождаются необходимые ресурсы.

С технической точки зрения во всех перечисленных случаях новый процесс формируется одинаково: текущий процесс выполняет системный запрос на создание нового процесса. В роли текущего процесса может выступать процесс, запущенный пользователем, системный процесс, инициированный клавиатурой или мышью, а также процесс, управляющий пакетами. В любом случае этот процесс всего лишь выполняет системный запрос и создает новый процесс. Системный запрос заставляет операционную систему создать новый процесс, а также прямо или косвенно содержит информацию о программе, которую нужно запустить в этом процессе.

5) Виды прерываний и особенности реагирования на них ОС.

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.