ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ» (МГТУ ГА)**

Кафедра вычислительных машин, комплексов, систем и сетей
Н.И. Черкасова, Л.А. Надейкина

ОПЕРАЦИОННЫЕ СИСТЕМЫ

ПОСОБИЕ
по выполнению лабораторной работы

**«Разработка приложений с использованием
прерываний и служебных функций ОС»**

*для студентов II курса
направления 09.03.01 (230100)
очной формы обучения*

Москва - 2015

ББК 6Ф7.3
Ч-48

Рецензент канд. техн. наук, доц. Л.А. Вайнейкис

Черкасова Н.И., Надейкина Л.А.

Ч-48 Операционные системы: пособие по выполнению лабораторной работы «Разработка приложений с использованием прерываний и служебных функций ОС». – М.: МГТУ ГА, 2015. – 24 с.

Данное пособие издается в соответствии с рабочей программой учебной дисциплины «Операционные системы» по Учебному плану для студентов II курса направления 09.03.01 (230100) очной формы обучения.

Рассмотрено и одобрено на заседаниях кафедры 19.05.15 г. и методического совета 19.05.15 г.

Подписано в печать 18.06.2015 г.		
Печать офсетная	Формат 60x84/16	1,14 уч.-изд. л.
1,40 усл.печ.л.	Заказ № 2035/	Тираж 80 экз.

Московский государственный технический университет ГА
125993 Москва, Кронштадтский бульвар, д.20
Редакционно-издательский отдел
125493 Москва, ул. Пулковская, д.6а

Содержание

Разработка приложений с использованием прерываний и служебных функций ОС

1.Цель лабораторной работы

Целью лабораторной работы является освоение:

1. структуры и состава векторов прерываний;
2. структуры и состава служебных функций;
3. приемов работы с векторами прерываний и служебными функциями MS DOS;
4. структуры и состава функций языка программирования C++, обслуживающих прерывания;
5. приемов работы с использованием обслуживающих средств MS DOS (служебных функций) и BIOS;
6. составление приложений, обслуживающих прерывания.

2. Содержание отчёта

Отчет по лабораторной работе должен включать:

1. цель лабораторной работы;
2. конкретный вариант задания на выполнение;
3. тексты программ;
4. схемы алгоритмов;
5. результаты выполнения программ.

3. Краткие теоретические сведения

3.1.Механизм прерываний

Для обработки событий, происходящих асинхронно по отношению к выполнению программы, лучше всего подходит механизм прерываний. Прерывание можно рассматривать как некоторое особое событие в системе, требующее моментальной реакции.

Например, хорошо спроектированные системы повышенной надежности используют прерывание по аварии в питающей сети для выполнения процедур записи содержимого регистров и оперативной памяти на носитель с тем, чтобы после восстановления питания можно было продолжить работу с того же места.

Механизм прерываний поддерживается аппаратными средствами компьютера и программными средствами операционной системы. Существуют два основных способа, с помощью которых шины выполняют прерывания: *векторный (vectored)* и *опрашиваемый (polled).* В обоих способах процессору предоставляется информация об уровне приоритета прерывания на шине подключения внешних устройств. В случае векторных прерываний в процессор

передается также информация о начальном адресе программы обработки возникшего прерывания — *обработчика прерываний.*

Поэтому прерывание рассматривается не просто как таковое, с ним связывают число, называемое номером типа прерывания или просто номером прерывания. С каждым номером прерывания связывается то или иное событие. Система умеет распознавать, какое прерывание, с каким номером оно произошло, и запускает соответствующую этому номеру процедуру.

В зависимости от источника прерывания делятся на три больших класса:

1. внешние;
2. внутренние;
3. программные.

3.2. Прерывания в реальном режиме

Программы могут сами вызывать прерывания с заданным номером. Для этого они используют команду INT. Это так называемые программные прерывания. Программные прерывания не являются асинхронными, так как вызываются из программы. Программные прерывания удобно использовать для организации доступа к отдельным, общим для всех программ модулям. Например, программные модули операционной системы доступны прикладным программам именно через прерывания, и нет необходимости при вызове этих модулей знать их текущий адрес в памяти. Прикладные программы могут сами устанавливать свои обработчики прерываний для их последующего использования другими программами. Для этого встраиваемые обработчики прерываний должны быть резидентными в памяти.

Аппаратные прерывания вызываются физическими устройствами и приходят асинхронно. Эти прерывания информируют систему о событиях, связанных с работой устройств, например о том, что завершилась печать символа на принтере и необходимо выдать следующий символ, или о том, что требуемый сектор диска уже прочитан, и его содержимое доступно программе.

Использование прерываний при работе с медленными внешними устройствами позволяют совместить ввод/вывод с обработкой данных в центральном процессоре, и в результате повышает общую производительность системы.

Некоторые прерывания (первые в порядке номеров) зарезервированы для использования самим центральным процессором на случай каких-либо особых событий вроде попытки деления на ноль, переполнения и т.п.

Обработчики прерываний могут сами вызывать программные прерывания, например, для получения доступа к сервису BIOS или DOS .

В таблице 1 приведены значения некоторых наиболее важных векторов прерываний:

Таблица 1 – вектора прерывания

Номер	Описание
0	Ошибка деления. Вызывается автоматически после выполнения команд DIV или IDIV, если в результате деления происходит переполнение (например, при делении на 0). Для процессора 8086 при этом адрес возврата указывает на следующую после команды деления команду, а в процессоре 80286 - на первый байт команды, вызвавшей прерывание.
1	Прерывание пошагового режима. Вырабатывается после выполнения каждой машинной команды, если в слове флагов установлен бит пошаговой трассировки TF. Это прерывание не вырабатывается после выполнения команды MOV в сегментные регистры или после загрузки сегментных регистров командой POP.
2	Аппаратное немаскируемое прерывание. Вырабатывается при ошибке четности в оперативной памяти и при запросе прерывания от сопроцессора.
3	Прерывание для трассировки. Это прерывание генерируется при выполнении однобайтовой машинной команды с кодом CCh
4	Переполнение. Генерируется машинной командой INTO, если установлен флаг OF. Если флаг не установлен, то команда INTO выполняется как NOP.
5	Печать копии экрана. Генерируется при нажатии на клавиатуре клавиши PrtScr.
6	Неопределенный код операции или длина команды больше 10 байт (для процессора 80286).
7	Особый случай отсутствия математического сопроцессора (процессор 80286).
8	IRQ0 - прерывание интервального таймера, возникает 18,2 раза в секунду.
9	IRQ1 - прерывание от клавиатуры. Генерируется при нажатии и отжатии клавиши. Используется для чтения данных от клавиатуры.
A	IRQ2 - используется для каскадирования аппаратных

	прерываний в машинах класса AT.
B	IRQ3 - прерывание асинхронного порта COM2.
C	IRQ4 - прерывание асинхронного порта COM1.
D	IRQ5 - прерывание от контроллера жесткого диска для XT.
E	IRQ6 - прерывание генерируется контроллером флоппи-диска после завершения операции.
F	IRQ7 - прерывание принтера. Генерируется принтером, когда он готов к выполнению очередной операции.
10	Обслуживание видеоадаптера.
11	Определение конфигурации устройств в системе.
12	Определение размера оперативной памяти в системе.
13	Обслуживание дисковой системы.
14	Последовательный ввод/вывод.
15	Расширенный сервис для AT-компьютеров.
16	Обслуживание клавиатуры.
17	Обслуживание принтера.
18	Запуск BASIC в ПЗУ, если он есть.
19	Загрузка операционной системы.
1A	Обслуживание часов.
1B	Обработчик прерывания Ctrl-Break.
1C	Прерывание возникает 18.2 раза в секунду, вызывается программно обработчиком прерывания таймера.
1D	Адрес видеотаблицы для контроллера видеоадаптера 6845.
1E	Указатель на таблицу параметров дискеты.
1F	Указатель на графическую таблицу для символов с кодами ASCII 128-255.
20-5F	Используется DOS или зарезервировано для DOS.
60-67	Прерывания, зарезервированные для пользователя.
68-6F	Не используются.
70	IRQ8 - прерывание от часов реального времени.
71	IRQ9 - прерывание от контроллера EGA.
72	IRQ10 - зарезервировано.

73	IRQ11 - зарезервировано.
74	IRQ12 - зарезервировано.
75	IRQ13 - прерывание от математического сопроцессора.
76	IRQ14 - прерывание от контроллера жесткого диска.
77	IRQ15 - зарезервировано.
78 - 7F	Не используются.

3.2.1. Прерывания MS-DOS

Прерывание с номером 21H - стандартное средство обращения к большинству функций MS-DOS. В некоторых случаях для уточнения вида операций используется регистр AX. Каждое прерывание или обращение к функции использует значения в различных регистрах для передачи или возвращения специальной информации.

Таблица 2 - Прерывания MS-DOS

Прерывание	Описание
Int20H	ОБЩЕЕ ПРЕРЫВАНИЕ ПРОГРАММЫ.
Int21H	ДИСПЕТЧЕР ФУНКЦИЙ MS-DOS.
Int22H	АДРЕС ПРЕРЫВАНИЯ ПРОГРАММЫ
Int23H	АДРЕС Ctrl-Break.
Int24H	ССЫЛКА НА УПРАВЛЯЮЩУЮ ЗАПИСЬ КРИТИЧЕСКОЙ ОШИБКИ.
Int25H	АБСОЛЮТНОЕ ЧТЕНИЕ ДИСКА.
Int26H.	АБСОЛЮТНАЯ ЗАПИСЬ НА ДИСК.
Int27H	ЗАВЕРШЕНИЕ ПРОГРАММЫ, ОСТАВЛЯЮЩЕЕ ЕЁ РЕЗИДЕНТНОЙ
Int30H-3FH	РЕЗЕРВИРУЮТСЯ ОПЕРАЦИОННОЙ СИСТЕМОЙ

3.3.Прерывания в защищенном режиме

Обработка прерываний в защищенном режиме отличается от обработки в реальном режиме, потому что:

1. В защищенном режиме изменено распределение номеров векторов прерываний;

2. Принципиально иным является механизм обработки прерываний.

3.3.1. Классификация прерываний в защищенном режиме

Прерывания и исключения можно разделить на несколько групп:

1. сбой;
2. ловушка;
3. аварийное завершение.

Это деление производится в соответствии со следующими признаками:

1. какая информация сохраняется о месте возникновения прерывания (исключения;
2. возможно ли возобновление прерванной программы.

Исходя из этих признаков, можно дать следующие характеристики вышеперечисленным группам:

1. Сбой (ошибка) — прерывание или исключение, при возникновении которого в стек записываются значения регистров cs: ip, указывающие на команду, вызвавшую данное прерывание. Это позволяет, получив доступ к сегменту кода, исправить ошибочную команду в обработчике прерывания и, вернув управление программе, фактически осуществить ее рестарт (в реальном режиме при возникновении прерывания в стеке всегда запоминается адрес команды, следующей за той, которая вызвала это прерывание).
2. Ловушка — прерывание или исключение, при возникновении которого в стек записываются значения регистров cs: ip, указывающие на команду, следующую за командой, вызвавшей данное прерывание. Механизм ловушек похож на механизм прерываний в реальном режиме. Если прерывание типа ловушки возникло в команде передачи управления jmp, то содержимое пары cs: ip в стеке будет отражать результат этого перехода, то есть соответствовать команде назначения.
3. Аварийное завершение — прерывание, при котором информация о месте его возникновения недоступна или неполна, и поэтому рестарт практически невозможен, если только данная ситуация не была запланирована заранее.

Соответствующие программы-обработчики ошибок, ловушек и аварий будут отличаться алгоритмами работы. Микропроцессор жестко определяет, какие прерывания являются ошибками, ловушками и авариями.

Некоторые прерывания при своем возникновении дополнительно генерируют и записывают в стек так называемый код ошибки. Этот код может впоследствии использоваться для установления источника прерывания. Код ошибки записывается в стек вслед за содержимым регистров eflags, cs и eip

3.3.2. Таблица дескрипторов прерываний IDT

Вместо таблицы векторов прерываний система прерываний имеет дело с таблицей дескрипторов прерываний (IDT, interrupt descriptor table) Таблица IDT связывает каждый вектор прерывания с дескриптором процедуры или задачи, которая будет обрабатывать это прерывание. Элементами таблицы IDT являются дескрипторы. Формат таблицы IDT подобен формату GDT и LDT (Global Descriptor Table, Local Descriptor Table.).

Дескрипторы в таблице прерываний обычно называются шлюзами (иногда коммутаторами). Шлюзы предназначены для указания точки входа в программу обработки прерывания. В дескрипторной таблице прерываний IDT могут содержаться шлюзы трех типов:

1. шлюз ловушки;
2. шлюз прерывания;
3. шлюз задачи.

Физически микропроцессор отличает шлюзы по содержимому полей.
Особенности таблицы IDT:

1. нулевой дескриптор используется для описания шлюза для программы обработки исключительной ситуации 0 (ошибка деления), (в таблице GDT отсутствует);
2. дескрипторы в таблице IDT строго упорядочены в соответствии с номерами прерываний, в таблицах GDT и LDT порядок описания дескрипторов роли не играет, хотя и допускается наличие некоторых соглашений по их упорядоченности;
3. размерность таблицы IDT — не более 256 элементов размером по восемь байт, по числу возможных источников прерываний.

В отдельных случаях есть смысл описывать все 256 дескрипторов этой таблицы, формируя для неиспользуемых номеров прерываний шлюзы-заглушки. Это позволит корректно обрабатывать все прерывания, даже если они и не планируются к использованию в данной задаче. Если этого не сделать, то при незапланированном прерывании с номером, превышающим пределы IDT для данной задачи, будет возникать исключительная ситуация общей защиты (с номером 13 (ODh)). Адрес и длина таблицы IDT содержатся в регистре idtr.
Заметим, что в реальном режиме регистр idtr также содержит адрес таблицы прерываний, но при этом каждый вектор занимает 4 байта и содержит 32-разрядный адрес в формате CS:IP. Начальное значение этого регистра равно нулю, но в него можно занести и другое значение.

Возможно произвольное размещение в памяти этой таблицы не только в защищенном режиме, но и реальном. В защищенном режиме произвести загрузку регистра idtr может только код с максимальным уровнем привилегий.
Доступ к таблице IDT со стороны пользовательских (прикладных) программ невозможен.

3.3.3.Обобщенная схема обработки прерывания в защищенном режиме

При возникновении прерывания от источника с номером N микропроцессор, находясь в защищенном режиме, выполняет следующие действия:
1. Определяет местонахождение таблицы IDT, адрес и размер которой содержится в регистре idtr;
2. Складывает значение адреса, по которому размещена IDT, и значение N*8, по данному смещению в таблице IDT должен находиться 8-байтовый дескриптор, определяющий местоположение процедуры обработки прерывания;
3. Переключается на процедуру обработки прерывания.

3.4. Прерывания в мультипрограммной ОС.

3.4.1.Диспетчеризация и приоритезация прерываний в ОС

Операционная система должна играть активную роль в организации обработки прерываний. Прерывания выполняют очень полезную для вычислительной системы функцию — они позволяют реагировать на асинхронные по отношению к вычислительному процессу события. В то же время прерывания создают дополнительные трудности для ОС в организации вычислительного процесса. Эти трудности связаны *с* непредвиденными переходами управления от одной процедуры к другой, возникающими в результате прерываний от контроллеров внешних устройств. Возможно, также, возникновение в непредвиденные моменты времени исключений, связанных с ошибками во время выполнения инструкций. Усложняют задачу планирования вычислительных работ и запросы на выполнение системных функций (системные вызовы) от пользовательских приложений, выполняемые с помощью программных прерываний. Сами модули ОС также часто вызывают друг друга с помощью программных прерываний, еще больше запутывая картину вычислительного процесса.

Операционная система не может терять контроль над ходом выполнения системных процедур, вызываемых по прерываниям. Она должна упорядочивать их во времени так же, как планировщик упорядочивает многочисленные пользовательские потоки. Кроме того, сам планировщик потоков является системной процедурой, вызываемой по прерываниям (аппаратным — от таймера или контроллера устройства ввода-вывода, или программным — от приложения или модуля ОС). Поэтому правильное планирование процедур, вызываемых по прерываниям, является необходимым условием правильного планирования пользовательских потоков. В противном случае в системе могут возникать, такие ситуации, когда операционная система будет длительное время заниматься не требующей мгновенной реакции задачей в то время, когда

будут простаивать и тормозить работу многочисленные приложения. Проблемы несколько смягчает то обстоятельство, что во многих случаях обработка прерывания связана с выполнением операций ввода-вывода и поэтому имеет очень небольшую продолжительность. Тем не менее, ОС всегда должна контролировать ситуацию и выполнять критичную работу своевременно

Для упорядочения работы обработчиков прерываний в операционных системах применяется тот же механизм, что и для упорядочения работы пользовательских процессов — механизм приоритетных очередей.. Все источники прерываний обычно делятся на несколько классов, причем каждому классу присваивается приоритет. В операционной системе выделяется программный модуль, который занимается диспетчеризацией обработчиков прерываний. Этот модуль в разных ОС называется по-разному, но для определенности будем его называть *диспетчером прерываний.*

Программные прерывания отличаются от двух аппаратных классов тем, что они по своей сути не являются «истинными» прерываниями. Программное прерывание возникает при выполнении особой команды процессора, выполнение которой имитирует прерывание, то есть переход на новую последовательность инструкций. Причины использования программных прерываний вместо обычных инструкций вызова процедур будут изложены ниже, после рассмотрения механизма прерываний.

Прерываниям приписывается приоритет, с помощью которого они ранжируются по степени важности и срочности. О прерываниях, имеющих одинаковое значение приоритета, говорят, что они относятся к одному *уровню приоритета* прерываний.

Прерывания обычно обрабатываются модулями операционной системы, так как действия, выполняемые по прерыванию, относятся к управлению разделяемыми ресурсами вычислительной системы — принтером, диском, таймером, процессором и т. п. Процедуры, вызываемые по прерываниям, обычно называют *обработчиками прерываний,* или *процедурами обслуживания прерываний (Interrupt Service Routine, ISR).* Аппаратные прерывания обрабатываются драйверами соответствующих внешних устройств, исключения — специальными модулями ядра, а программные прерывания — процедурами ОС, обслуживающими системные вызовы. Кроме этих модулей в операционной системе может находиться так называемый диспетчер прерываний, который координирует работу отдельных обработчиков прерываний.

4.Программирование приложений, обслуживающих прерывания

4.1. Структура и состав функций языка программирования C++, обслуживающих прерывания

Как известно, C++, обеспечивает возможность создания собственных программ, обслуживающих прерывания. Рассмотрим основные функции языка C++ для работы с прерываниями. Заголовочные файлы: dos.h.

4.1.1.Модификатор **interrupt**

Модификатор **interrupt** объявляет функцию, как программу обработки прерываний.

4.1.2.Функция **getvect()**

Синтаксис: void interrupt (*getvect(int interruptno))().

Параметры: int interruptno – номер прерывания в диапазоне от 0 до 255, используемый как индекс в массиве дальних указателей или векторов, запомненных в младших адресах памяти.

Описание: Возвращает вектор прерывания в виде дальнего указателя на функцию обработки прерывания.

4.1.3.Функция **int86()**

Синтаксис: int int86(int intno, union REGS* inregs, union REGS* outregs).

Параметры: int intno – номер программного прерывания; union REGS* inregs – указатель на структуру, членам этой структуры следует присваивать значения, которые данная функция копирует в регистры процессора до выполнения программного прерывания; union REGS* outregs – содержимое регистров процессора, копируемое функцией после возврата из прерывания.

Описание: Выполняет программное прерывание. Возвращает значение регистра AX после возврата из прерывания.

4.1.4.Функция int **int86x** (расширенный интерфейс)

Синтаксис: int int86x(int intno, union REGS* inregs, union REGS* outregs, struct SREGS* segregs).

Параметры: struct SREGS* segregs – указатель на структуру, члены ds и es которой установлены равными адресным значениям сегментов, копируемым в регистры процессора DS и ES перед выполнением программного прерывания. После возврата из прерывания данная функция устанавливает члены структуры ds и es равными значениям регистров DS ES и восстанавливает исходное значение в регистре DS. Значение регистра ES не восстанавливается.

Описание: Аналогично функции int86(), но содержит четвертый параметр, адресующий структуру, определяющую значение сегментных регистров, копируемых в регистры DS и ES перед выполнением программного прерывания.

4.1.5.Функция **int intdos()**

Синтаксис: int intdos(union REGS* inregs, union REGS* outregs)

Параметры: Те же, что и для функции int86(), за исключением номера программного прерывания.

Описание: Вызывает функцию DOS, выполняя программное прерывание 0x21. Возвращает значение регистра процессора AX после возврата из функции.

4.1.6.Функция **setvect()**

Синтаксис: setvect(int interruptno, void interrupt(*ISR) ()).

Параметры: int interruptno – номер прерывания; void interrupt(*ISR(Interrupt Service Routine)) – указатель на функцию обслуживания прерывания.
Описание: Устанавливает вектор прерывания равным адресу указываемой подпрограммы обслуживания прерывания. Следует отметить необходимость возвращения функции обработки прерывания в конце программы. В приложении представлен пример программы, замещающий стандартный вектор прерывания 9листинг 1).

4.2..Программирование на нескольких языках

Ассемблер наиболее эффективно обеспечивает возможность создания собственных программ, обслуживающих прерывания.

Современные компиляторы позволяют совместно использовать преимущества языка высокого уровня и эффективность ассемблерных программ, допуская написание частей одной программы на разных языках программирования. В частности, большинство компиляторов С поддерживают три основных способа совместного использования кода на С++ и ассемблере и их комбинации:

1. экспорт объектного модуля, созданного на С, для использования ассемблерной программой, при таком подходе на С программируется интерфейс с пользователем;
2. импорт объектного модуля, созданного на ассемблере, для использования с программой на С (этот подход позволяет наиболее полно использовать преимущества совместного программирования на С и ассемблере, причем, такая организация программы сложна с точки зрения программиста);
3. использование встроенного ассемблера – написание на ассемблере части исходного кода непосредственно внутри исходного текста программы на языке С (подход имеет некоторые ограничения при использовании ассемблера, такая организация программы наиболее проста с точки зрения программиста).

Наиболее гибким, с точки зрения предоставляемых возможностей, является второй способ (или комбинация первого и второго способов). Однако, его использование сложно с точки зрения программиста. При его реализации критическая часть программы пишется на языке ассемблера и компилируется в объектный код. При написании могут использоваться все преимущества ассемблера: гибкое описание сегментов и структур данных, низкоуровневая оптимизация под конкретное семейство процессоров и т.п. Однако для того, чтобы результирующий объектный модуль мог быть подключен к модулю основной программы, скомпилированной из языка С, необходимо учесть массу формальностей, касающихся именования переменных, подпрограмм и ячеек памяти, правил стыковки сегментов и т.п., причем нужно принимать во внимание специфику использования различных компиляторов.

В силу перечисленных сложностей 1 и 2 подходов наиболее часто на практике применяют третий способ, программируя на встроенном ассемблере.

Встроенный ассемблер большинства С-компиляторов не позволяет явно описывать сегменты и процедуры. Он имеет различные ограничения, в частности, при определении ячеек памяти и использовании регистров процессора, однако, с учетом ограничений, позволяет получать код близкий по эффективности к полноценному ассемблеру. Основными достоинствами использования встроенного ассемблера по сравнению с двумя другими подходами, являются простота использования с точки зрения программиста и хорошая переносимость исходных текстов программ между различными компиляторами.

Подробное описание особенностей работы и ограничений при использовании встроенного ассемблера нужно искать в руководствах программиста и технической документации на каждый конкретный компилятор.

Для написания кода на встроенном ассемблере в Borland C++ 3.1 используется конструкция:

```
asm{
//Текст программы на языке ассемблера
}.
```

Встроенный ассемблер BC++ 3.1 не допускает модификации сегментных регистров, кроме ES, имеются существенные ограничения при объявлении поименованных ячеек памяти и вызове подпрограмм. При использовании команд перехода метки, на которые осуществляется переход, должны быть описаны вне конструкции asm{...}, существуют и другие ограничения. Комментарии оформляются по правилам С. Однако, внутри конструкций asm{...} допустимы обращения по именам к переменным, объявленным в программе по правилам языка С.

В качестве примера использования встроенного ассемблера представлена программа, позволяющая осуществлять вывод на экран монитора в текстовом режиме строки текста заданным цветом в точку с указанными координатами (листинг 2).

4.3. Обработка ошибок

Перед обращением программы к прерыванию, необходимо загрузить перед вызовом прерывания все необходимые операнды в регистры процессора. Если выполнение операции невозможно по каким-то причинам (неправильные операнды, устройство неработоспособно и т.д.), то для большинства функций устанавливается признак ошибки - флаг переноса, регистр AX при этом содержит код ошибки.

Таблица 3 - Коды ошибок, возвращаемые программе через регистр AX

№	Код ошибки

1	Неправильный код функции
2	Файл не найден
3	Путь не найден
4	Слишком много открытых файлов
5	Доступ запрещен
6	Неправильный идентификатор файла
7	Разрушен блок управления памятью
8	Недостаточно памяти
9	Неправильный адрес блока памяти
10	Неправильная среда
11	Неправильный формат
12	Неправильный код доступа
13	Неправильные данные
14	Зарезервировано
15	Ошибка при указании дисковода
16	Невозможно удалить текущий каталог
17	Другое устройство
18	Больше нет подходящих файлов

Для получения дополнительной информации об ошибках введена функция 59h прерывания INT 21h.

Функция выдает подробную информацию об ошибке, произошедшей непосредственно перед обращением к обработчику int24h или к одной из системных функций 2FH-62H. Перед обращением к функции 59H в регистр BX помещается значение 00H. В результате работы функции в регистре AX возвращается код ошибки. Также возвращаются три вида данных:

1. в регистре BH - класс ошибки;
2. врегистре BL - рекомендуемые действия;
3. в регистре CH - местоположение ошибки.

При вызове этой функции регистр BX должен содержать индикатор уровня анализа ошибок, который должен быть равен 0. Кроме расширенного кода ошибки, возвращаемого в регистре AX, программа может получить класс ошибки (регистр BH), код предполагаемых действий (регистр BL), локализацию ошибки, т.е. место, где произошла ошибка (регистр CH).

К сожалению, эта функция разрушает содержимое регистров CL, DX, SI, DI, BP, DS, ES. Программа, использующая функцию 59h, должна позаботиться о сохранении содержимого этих регистров.

Расширенный код ошибки, возвращаемый в регистре AX, может принимать значения, указанные в приводимой ниже таблице. Коды от 1 до 18 эквивалентны представленным выше и второй раз не приводятся.

Таблица 4 - Расширенные коды ошибок, возвращаемые программе через регистр AX

№	Код ошибки
19	Запись на защищенный от записи диск
20	Задан неизвестный идентификатор устройства
21	Дисковод не готов
22	Неизвестная команда
23	Ошибка циклического кода проверки
24	Неправильная длина структуры запроса
25	Ошибка поиска
26	Неизвестен тип среды носителя данных
27	Сектор не найден
28	Кончилась бумага в принтере
29	Ошибка записи
30	Ошибка чтения
31	Общая ошибка
32	Нарушение разделения файла
33	Нарушение блокировки файла
34	Неправильная замена диска
35	FCB недоступен (слишком много блоков FCB)
36	Переполнился буфер разделения
37	Зарезервировано
38	Не завершена операция "Конец файла"
39-49	Зарезервировано
50	Сетевая функция не поддерживается
51	Удаленный компьютер "не слышит"
52	Дублирование имени в сети
53	Сетевое имя не найдено
54	Сеть занята

55	Сетевое устройство больше не существует
56	Превышен лимит команды сетевой BIOS
57	Ошибка в аппаратуре сетевого адаптера
58	Неправильный ответ из сети
59	Непредусмотренная ошибка сети
60	Несовместимый удаленный адаптер
61	Заполнена очередь печати
62	Для печатаемого файла недостаточно места
63	Печатающийся файл был удален
64	Сетевое имя было удалено
65	Доступ запрещен
66	Неправильный тип сетевого устройства
67	Сетевое имя не найдено
68	Превышен лимит сетевого имени
69	Превышен лимит сеанса сетевой BIOS
70	Временная пауза

Таблица 5 - Класс ошибки, передаваемый в регистре BH,

№	Класс ошибки
1	Недостаточно ресурсов: блоков FCB, памяти и т.д.
2	Временная ситуация
3	Нет прав доступа
4	Внутренняя ошибка DOS
5	Ошибка аппаратуры
6	Системная ошибка DOS (нет CONFIG.SYS и т.п.)
7	Ошибка в прикладной программе
8	Файл или объект не найден
9	Неправильный формат файла или объекта
10	Файл или объект заблокирован
11	Ошибка носителя данных
12	Файл или объект уже существует

13	Прочие ошибки

Таблица 6 - Код предполагаемых действий, передаваемый через регистр BL

№	Код действий
1	Повторить операцию позже
2	Повторить предыдущую операцию после небольшой паузы
3	Если пользователь вводил какие-то данные для DOS, следует попросить его ввести эти данные еще раз
4	Аварийно завершить работу прикладной программы с выполнением всех обычных завершающих действий
5	Немедленный выход из программы без выполнения завершающих действий
6	Следует игнорировать ошибку
7	Повторить операцию после того, как пользователь выполнит требуемые действия

Таблица 6 - Коды локализации в регистре CH.

№	Коды локализации
1	Локализация ошибки не может быть определена (система не знает, где произошла ошибка).
2	Ошибка произошла в блочном устройстве (диск или магнитная лента).
3	Ошибка связана с сетью.
4	Ошибка произошла в символьном устройстве, например, в принтере.
5	Ошибка связана с оперативной памятью.

Если программа составлена на языке ассемблера, то после обращения к DOS через прерывание следует проверить состояние флага переноса:

```
int     21h
jc      error
```

Программы, составленные на языке Си для передачи параметров используются структуры REGS, WORDREGS, BYTEREGS, SREGS. Они описаны в файле dos.h, Значение флага переноса записывается в переменную cflag,

определенную в структуре WORDREGS. Эта структура входит в объединение REGS.

```
union REGS {
    struct WORDREGS x;
    struct BYTEREGS h;}
```

Код ошибки при этом содержится в переменной outregs.x.ax.

В листинге 3 приведен пример программы, которая стирает каталог с именем DIR в текущем каталоге и, в случае ошибки, выводит расширенную информацию об ошибке, класс ошибки, код предполагаемых действий и код локализации ошибки:

5. Контрольные вопросы

1. Понятие прерывания.
2. Понятие вектора прерывания.
3. Основные типы прерываний.
4. Прерывания реального и защищенного режима работы.
5. Прерывания мультипрограммного режима.
6. Обработка ошибок.
7. Векторы прерывания MS DOS и BIOS.
8. Служебные функции MS DOS.
9. Функции языка программирования C++, обслуживающие прерывания.

Литература.

1. В.Г.Олифер, Н.А. Олифер. Сетевые операционные системы. - Спб: Питер, 2003г.
2..Подбельский В.В. Язык Си++. – М: Финансы и статистика, 1996 г.
3. Т.Сван. Программирование для Windows в Borland C++.- М: Бином, 1995 г.
4.Э. Таненбаум Современные операционные системы.Питер, 2006 г.
5.Черкасова Н.И. Тексты лекций по дисциплине «Операционные системы» для студентов специальности 230101 дневного обучения, часть1,- М., РИО МГТУ ГА, 2010 г.
6.Черкасова Н.И. Тексты лекций по дисциплине «Операционные системы» для студентов специальности 220100 дневного обучения, часть 2,- М., РИО МГТУ ГА, 2003 г.

Приложение.

Листинг 1.

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>
void interrupt get_out(...); /* interrupt прототип*/
void interrupt (*oldfunc)(...);
```

```
        /*Установление нового вектора прерывания для прерывания 5H – вызова
служебной программы BIOS печати с экрана*/
        int count=0;
        main()
        {int old_count=0;
        oldfunc = getvect(5);
        setvect(5,get_out);
        puts("Press <Shift><PrtSc> 3 to end");
        while (count<3)
        if (count !=old_count)
        { puts("Press <Shift><PrtSc");
        old_count= count;}
setvect(5,oldfunc);
return 0;}
void interrupt get_out(...) { count }
```

Листинг 2.

```
#define clBlack 0 //Описания констант, кодирующих цвет
#define clBlue 1
#define clGreen 2
#define clCyan 3
#define clRed 4
#define clMagenta 5
#define clBrown 6
#define clLightGray 7
#define clDarkGray 8
#define clLightBlue 9
#define clLightGreen 10
#define clLightCyan 11
#define clLightRed 12
#define clLightMagenta 13
#define clYellow 14
#define clWhite 15
#define clBlink 1 //и наличие
#define clNoBlink 0 //признака мерцания
//Для видеорежима 80x50
#define MAX_WIDTH 80 //Количество знакомест в строке
#define MAX_HEIGHT 50 //Количество строк на экране
unsigned char text_attribute; //Глобальная переменная – аттрибуты
void set_color(char color, char bgcolor, char blink){
//Задает цвет текста, цвет фона и признак мерцания -
//формирует байт атрибутов, модифицируя глобальную text_attribute
asm{
```

```
push AX //Сохраняем содержимое используемого регистра AX
mov AL, bgcolor //Формируем атрибут цвета фона сдвигом 3 битного
shl AL, 4 //кода фона в старшую тетраду байта атрибутов
mov AH, blink //Если blink
cmp AH, 0 //не равен 0
je no_blink //то
or AL, 0x80} //установить _____старший бит – признак мерцания
no_blink:
asm{
mov AH, color //Устанавливаем атрибуты цвета текста,
add AL, AH //добавляя значение color к содержимому AL
mov text_attribute, AL //Сформированный в AL байт атрибутов помещаем
//в глобальную переменную text_attribute
pop AX}} //Восстанавливаем AX
void string2screen(char x, char y, char *string){
//Выводит на экран строку string, начиная с позиции с координатами
//(x, y). При выводе символы отображаются с атрибутом text_attribute
asm{
push AX //Сохраняем
push BX //значения
push DI //регистров,
push SI //используемых в
push ES //функции
mov AX, 0xB800 //Загружаем в ES адрес
mov ES, AX //текстового видеобуфера: B8000h
//Рассчитаем начальное смещение в видеобуфере DI<-(MAX_WIDTH*y+x)*2
mov AL, y //AL=y
mov AH, MAX_WIDTH //AH=MAX_WIDTH
mul AH //AX=AL*AH: (MAX_WIDTH*y)
xor BX, BX //BX=0 (Результат команды эквивалентен mov BX, 0)
mov BL, x //BL=x
add AX, BX //AX=AX+BL: (MAX_WIDTH*y+x)
shl AX, 1 //AX=AX*2: (MAX_WIDTH*y+x)*2
mov DI, AX //DI<-(MAX_WIDTH*y+x)*2
mov AH, text_attribute //Загружаем в AH байт атрибутов
mov SI, string }//SI хранит адрес начала строки string
cycle:
asm{
mov AL, [SI] //Разыменование: загрузка в AL байта по адресу SI,
//т.е. ASCII-код очередного символа строки
cmp AL, 0 //ЕСЛИ достигнут завершающий строку 0
je end_of_string //ТО на выход
mov ES:[DI], AX //ИНАЧЕ вывести очередной символ в видеобуфер
```

```
inc SI //Инкремент смещения очередного символа строки
add DI, 2 //Увеличиваем смещение видеобуфера
jmp cycle} //Цикл итерирует до вывода всех символов строки
end_of_string:
asm{
pop ES //Восстанавливаем
pop SI //значения
pop DI //сохраненных
pop BX //регистров
pop AX}} //Команды pop вызываются в порядке обратном push
void clear_screen(void){
//Очищает экран, заполняя его пробелами с текущим цветом фона
asm{
push AX //Сохраняем
push DI //значения
push ES //используемых
push CX //регистров
mov AX, 0xB800 //Загружаем адрес
mov ES, AX //текстового видеобуфера в ES
mov AL, ' ' //В AL заносим код символа ”пробел”
mov AH, text_attribute //В AH сохраняем значение байта атрибутов
mov DI, 0 //Начальное смещение в видеобуфере
mov CX, MAX_WIDTH*MAX_HEIGHT} //Количество символов,
//помещающихся на экране в текущем видеорежиме помещаем в счетчик цикла
cycle:
asm{
mov ES:[DI], AX //Отправляем очередной ”пробел” в видеобуфер
add DI, 2 //Увеличиваем смещение в видеобуфере
loop cycle} //Цикл итерирует CX раз
asm{
pop CX //Восстанавливаем
pop ES //значения
pop DI //используемых
pop AX}} //регистров
void main(void){
set_color(clYellow, clBlue, clBlink ); //Установим цвет фона - синий,
//цвет символа – желтый,
//атрибут мерцания установлен
clear_screen(); //Очистим экран
string2screen(10,10,"Hello world!");} //Вывод на экран в заданную позицию
//текстовой строки
```

Листинг 3.

```
#include <dos.h>
#include <stdio.h>
union REGS inregs, outregs;
struct SREGS segregs;
void main(void);
void main(void) {
char _far *dir_name = "DIR";
// Стираем каталог с именем DIR. Для этого вызываемфункцию 0x3A
inregs.h.ah = 0x3a;
      segregs.ds = FP_SEG(dir_name);
      inregs.x.dx = FP_OFF(dir_name);
      intdosx(&inregs, &outregs, &segregs);
// Если после выполнения прерывания установлен
// флаг переноса, выводим сообщение об ошибке.
      if(outregs.x.cflag != 0) {
        printf( "Ошибка при удалении каталога: %d",
        outregs.x.ax);
// Получаем расширенную информацию об ошибке
// с помощью функции 0x59 прерывания INT 21h.
        inregs.h.ah = 0x59;
        inregs.x.bx = 0;
// Сохраняем регистры в стеке, т.к. их содержимое
// изменится
      _asm {
          push ds
          push es
          push si
          push di}
      intdosx(&inregs, &outregs, &segregs);
      _asm {
           pop di
           pop si
           pop es
           pop ds}
// Выводим расширенную информацию об ошибке.
      printf("\nРасширенный код ошибки:   %d"
          "\nКласс ошибки:            %d"
          "\nПредполагаемые действия:  %d"
          "\nЛокализация ошибки:       %d",
          outregs.x.ax,
          outregs.h.bh,
          outregs.h.bl,
          outregs.h.ch);}}
```