

Содержание

Лабораторная работа №13 Инструментарий операционной системы для сбора информации о компонентах ПЭВМ

Лабораторная работа №14 Программирование внутренних устройств компьютера. Изучение интерфейса PCI

Лабораторная работа №15 Исследование S.M.A.R.T.-атрибутов жестких дисков

Лабораторная работа №16 Работа с интерфейсом ввода-вывода ПЭВМ

Лабораторная работа №17 Анализ трафика между периферийными устройствами и ЭВМ

Лабораторная работа №18 Основы работы с кодами клавиш стандартной клавиатуры

Лабораторная работа №19 Определение основных характеристик видеоадаптера

Лабораторная работа №20 Программное извлечение флеш-диска

Лабораторная работа №21 Исследование характеристик привода оптических дисков

ЛАБОРАТОРНАЯ РАБОТА N1

Инструментарии операционной системы для сбора информации о компонентах ПЭВМ

1. Цель работы

Практическое ознакомление со структурой ПЭВМ, методами определения ее конфигурации, ресурсами, выделяемые компонентам ПЭВМ.

2. Рекомендуемая литература

- 2.1. Гук М. Аппаратные интерфейсы ПК. Энциклопедия.– СПб.: Питер, 2002, с.454-465.
- 2.2. Мюллер С. Модернизация и ремонт ПК, 16-е изд.: Пер. с англ. – М.: Вильямс, 2006, с.400-411.
- 2.3. Несвижский В. Программирование аппаратных средств в Windows. – СПб.: БХВ-Петербург, 2004, с.799-815.
- 2.4. Юров, В. И. Assembler [Text] : спец. справочник / Юров В. И. - 2-е изд. - СПб. : Питер, 2004. – с. 67-72.
- 2.5. CPUID – Идентификация CPU – Club155.ru [Электронный ресурс]. – Режим доступа: <http://www.club155.ru/x86cmd/CPUID> , свободный. – Загл. с экрана. – 17.02.2017.
- 2.6. CPUID Specification [Электронный документ] / Advanced Micro Devices – Режим доступа: <https://support.amd.com/TechDocs/25481.pdf>, свободный. – 20.02.2017.
- 2.7. Intel 64 and IA-32 Architectures Developer's Manual: Vol. 2A [Электронный документ] / Intel Corporation – Режим доступа: <http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-2a-manual.html> , свободный. – 20.02.2017.
- 2.8. About WMI (Windows) [Электронный документ]. – Режим доступа: [https://msdn.microsoft.com/en-us/library/aa384642\(VS.85\).aspx](https://msdn.microsoft.com/en-us/library/aa384642(VS.85).aspx) . – 20.02.2017.
- 2.9. WMIC – использование инструментария управления Windows (WMI) в командной строке [Электронный документ]. – Режим доступа: <http://ab57.ru/cmdlist/wmic.html> . – 20.02.2017.
- 2.10. Финогенов К.Г. Использование языка Ассемблера: Учебное пособие для вузов. – М.: Горячая линия–Телеком, 2004. –с. 309-320.

3. Подготовка к работе

- 3.1. Изучить методические указания и рекомендованную литературу.
- 3.2. Подготовить ответы на контрольные вопросы.

4. Контрольные вопросы

- 4.1. Какими средствами можно определить аппаратную конфигурацию ПЭВМ?
- 4.2. Какие ресурсы могут быть выделены периферийным устройствам?
- 4.3. Каким устройствам выделяются стандартные ресурсы, а каким динамические?
- 4.4. В случае обнаружения конфликта по прерыванию при подключении нового устройства, какой номер прерывания ему лучше выдать? Как это сделать?
- 4.5. Какие прерывание и адреса портов ввода-вывода обычно выдаются первому последовательному порту?
- 4.6. Как получить список видеорежимов?
- 4.7. Как определить производителя видеоадаптера?
- 4.8. Как определить производителя и модель микропроцессора, установленного в ПЭВМ?
- 4.9. Для чего предназначена команда CPUID?
- 4.10. Как использовать команду CPUID? Какой алгоритм работы с командой CPUID?
- 4.11. Какую информацию можно получить, используя программу WMIC?
- 4.12. Формат запроса через WMIC. Приведите примеры запроса к WMI через приложение WMIC.
- 4.13. Для чего нужен WMI?
- 4.14. Как определить поддержку процессором команд MMX и SIMD?

5. Задание на выполнение работы

- 5.1. Создать и отладить программу получения информации о процессоре согласно варианту задания из таблицы 1.1. Программа должна содержать проверку поддержки процессором команды CPUID и выводить информацию о количестве поддерживаемых функций CPUID и производителе процессора.

Вариант задания соответствует целой части суммы деления последней цифры зачетной книжки на два с единицей. Пример: послед-

ная цифра зачетной книжки 7, тогда номер варианта находится из выражения $N=7/2+1$, в этом случае номер варианта равен 4.

Таблица 1.1

Варианты заданий	
№ варианта	Выводимый параметр процессора
1	сигнатура процессора (в двоичном виде)
2	номер разработки (stepping)
3	модель
4	семейство
5	полное наименование

5.2. Запустите утилиту *Сведения о системе*.

Для этого наберите в поисковой строке меню Пуск или в приложении Выполнить (открывается комбинацией клавиш **Win+R**)

MSinfo32.exe.

Ознакомьтесь со структурой окна программы. Системная информация на панели обзора разделена на четыре категорий: корневой узел *Сведения о системе* и три основных подузла:

- аппаратные ресурсы;
- компоненты;
- программная среда.

Определить конфликтующие по используемым ресурсам устройства. В случае наличия таких устройств в отчет запишите информацию по одному из конфликтов для каждого ресурса: порт ввода-вывода, адрес памяти, прерывание IRQ.

5.3. Запустите утилиту *Диспетчер устройств*.

В ОС Windows Vista/7/8/10 «Диспетчер устройств» можно найти в меню Пуск / Панель управления / Оборудование и звук. Диспетчер устройств можно также открыть набрав в поисковой строке меню Пуск или в приложении Выполнить

Devmgmt.msc.

Ознакомьтесь с окном программы. Раскройте несколько веток дерева устройств и выберите устройства. Просмотрите свойства устройств, выбрав одноимённый пункт в контекстном меню данных устройств.

5.4. Запустите утилиту *Средство диагностики DirectX*.

Для этого наберите в поисковой строке меню Пуск или в приложении Выполнить

Dxdiag.exe.

Запрос проверки цифровых подписей драйверов можно игнорировать.

Ознакомьтесь со структурой окна программы и просмотрите содержимое всех вкладок окна программы.

5.5. Используя командную строку, ознакомьтесь с возможностями приложения **wmic.exe** вызвав справку и сформировав несколько запросов:

wmic /? — ознакомиться с командами *WMIC*.

wmic COMPUTERSYSTEM get /value — получить информацию о системе.

wmic BIOS get /value / more — получить информацию о *BIOS*.

Используя приложение **wmic.exe** сформируйте запросы отображения серийных номеров накопителей информации, системной платы и оперативной памяти. Занесите в отчет результаты сформированных запросов.

5.6. Используя вышеуказанные системные утилиты и средства WMI, заполните соответствующие столбцы таблиц 1.2 – 1.4.

6. Отчёт должен содержать:

6.1 Титульный лист (с названием ВУЗа, кафедры, номера лабораторной работы и её названия, фамилии И.О. студента, подготовившего отчёт).

6.2 Цель работы.

6.3 Листинг программы и результат её работы согласно заданию 5.1.

6.4 Информацию о наличие конфликтов в использованном ПЭВМ.

6.5 Информацию о серийных номерах согласно заданию 5.5.

6.6 Информацию об аппаратно-программной части использованного ПЭВМ, занесенную в таблицы 1.2 – 1.4.

Таблица 1.2

Сведения о системе

Характеристика	Значение
Изготовитель процессора	
Модель процессора	
Количество ядер процессора	
Частота ядра процессора	
Объем кэш памяти L1	
Объем кэш памяти L2	
Объем кэш памяти L3	
Сокет процессора	
Изготовитель системной платы	
Серия системной платы (Product)	
Модель системной платы	
Изготовитель видеоадаптера	
Модель видеоадаптера	
Объем видеопамяти	
Объем оперативной памяти	
Количество планок оперативной памяти	
Изготовитель оперативной памяти	
Поддерживаемая скорость передачи данных оперативной памятью	
Описание BIOS	
Версия операционной системы	
Версия DirectX	
Сетевое имя компьютера	

Таблица 1.3

Сведения о диске

Характеристика	Значение
Изготовитель	
Модель	
Объём	
Количество цилиндров	
Количество секторов	
Количество разделов на диске	

Таблица 1.4

Сведения о периферийных устройствах

Тип периферийного устройства	Название устройства	Выделяемые устройству ресурсы		
		Порты ввода/вывода	Прерывание	Ячейки памяти
Клавиатура				
Видеоадаптер				
Сетевой адаптер				
Последовательный порт				
Контроллер IDE ATA/ATAPI или SATA				
Хост-контроллер USB				

Примечание: в случае наличия в компьютере нескольких устройств одной категории в таблицу записывается информация обо всех устройствах.

7. Общие сведения

Персональные ЭВМ имеют модульную структуру для обеспечения создания из базовых модулей необходимую для пользователя конфигурацию машины. Обычно в любой ПЭВМ имеются следующие узлы:

- процессор;
- материнская (системная) плата;
- оперативная память;
- видеоадаптер;
- жесткий диск;
- корпус системного блока с блоком питания;
- монитор;
- клавиатура;
- мышь.

Определить конфигурацию ПЭВМ можно различными способами:

- опросить компоненты ПЭВМ программным способом;
- просмотреть информацию в BIOS;
- воспользоваться служебными программами операционной системы («Сведения о системе» MSinfo32.exe, «Диспетчер устройств» Devmgmt.msc, «Средство диагностики DirectX» Dxdiag.exe, «Консольная утилита для вызова объектов и методов WMI» Wmic.exe, «Средство оценки производительности» WinSat.exe);

- запустить специализированные диагностические и тестовые программы (AIDA64 от FinalWire Ltd, Sandra от SiSoftware, Speccy от Piriform и др.).

Современные операционные системы семейства Windows имеют достаточно много средств для программного получения информации об аппаратной части компьютера. В дальнейшем рассмотрим только два таких инструмента.

Программное получение информации о процессоре

Для программной идентификации большинство современных процессоров поддерживает команду CPUID. Данная команда впервые появилась в процессоре Pentium и предоставляет программному обеспечению информацию о производителе, семействе, модели и поколении процессора, наборе поддерживаемых процессором функций и другую информацию.

Команда CPUID не имеет операндов, однако использует регистр EAX для указания номера функции (входное значение). Для разных моделей процессоров задокументированы различные наборы допустимых входных значений EAX. В общем случае, их можно разделить на стандартные (поддерживаемые всеми производителями) и расширенные (так или иначе отличающиеся для процессоров разных моделей и производителей).

Вся информация, даже текстовая, возвращается в регистрах EAX, EBX, ECX и EDX. В зависимости от запрашиваемой информации (входного значения в EAX) назначение регистров будет разным.

Если входное значение EAX=0, то после выполнения команды CPUID регистр EAX будет содержать максимальное значение, понимаемое командой CPUID (только стандартные функции), а в регистрах EBX, EDX и ECX будет находиться строка идентификации производителя процессора (EBX содержит первых 4 символа, EDX содержит следующие 4 символа и ECX содержит последние 4 символа). В таблице 1.5 приведены значения, выдаваемые наиболее распространенными моделями микропроцессоров (МП).

Таблица 1.5

Содержимое регистров после выполнения команды CPUID (EAX = 0)

Производитель	EAX	EBX:EDX:ECX	
		ASCII-строка	HEX-значения
Intel	x	GenuineIntel	756E6547:49656E69:6C65746E
AMD	1	AuthenticAMD	68747541:69746E65:444D4163
Cyrix	1	CyrixInstead	69727943:736E4978:64616574
Centaur	1	CentaurHauls	746E6543:48727561:736C7561
SiS	1	SiS SiS SiS	20536953:20536953:20536953

После выполнения команды CPUID с входным значением EAX=1, биты регистра EAX будут иметь следующие назначения (т.н. сигнатура процессора):

EAX[3:0] – (Stepping) номер разработки МП;

EAX[7:4] – (Base Model) модель МП,

EAX[11:8] – (Base Family) номер семейства МП;

EAX[13:12] – (Type) тип МП (00b – стандартный, 01b – OverDrive, 10b – Dual, 11b – зарезервировано; процессоры Pentium OverDrive for Pentium возвращают 00b в поле тип);

EAX[15:14] – зарезервировано;

EAX[19:16] – (Extended Model) расширение поля модели МП;

EAX[27:20] – (Extended Family) расширение поля семейства МП;

EAX[31:28] – зарезервировано.

Семейство МП (Family) объединяет несколько выпускаемых процессоров в одну группу, обладающую некоторыми общими свойствами программного и аппаратного обеспечения. Модель (Model) выделяет один экземпляр из семейства МП. Номер разработки (Stepping) определяет конкретную версию определённой модели.

Семейство МП определяется как сумма чисел в Base Family и Extended Family, однако если значение Base Family меньше Fh, то Extended Family не используется, а в качестве семейства МП используется только поле Base Family. Например, если Base Family = Fh, а Extended Family = 1h, то семейство МП будет 10h.

Модель МП формируется объединением двух чисел в одной записи слева на право: сначала записывается Extended Model, а затем Base Model. Если Base Model меньше чем Fh, то Extended Model не используется, а модель МП формируется только из Base Model. Рассмотрим пример: Base Model = 8h, Extended Model = Eh, тогда модель МП будет E8h. Программно реализовать объединение двух полей можно выделив из сигнатуры соответствующие поля в отдельные регистры, выполнить смещение на 4 разряда влево поля Extended Model с

заполнением нулями справа и сложить содержимое используемых регистров.

Регистры EBX и ECX при входном значении EAX=1 возвращают информацию о характеристиках процессора.

В регистре EDX будет содержаться информация о некоторых свойствах и возможностях микропроцессора, например, о поддержке команд 3DNow!, MMX, SIMD. Установленный флаг свойств указывает на то, что соответствующее свойство (возможность, функция) данной моделью микропроцессора поддерживается. Назначение битов регистра EDX можно найти в рекомендованной литературе [2.5].

Команда CPUID с входным значением EAX = 2 предназначена для получения информации об объеме и типе КЭШ памяти микропроцессора. После выполнения этой команды в регистрах EAX, EBX, EDX, ECX содержится соответствующая информация, причем, в младших 8-ми битах регистра EAX (регистр AL) содержится число, указывающее на то, сколько раз подряд необходимо выполнить команду CPUID со входным EAX = 2, чтобы получить достоверную информацию о микропроцессоре (в случае AL = 1, команда должна выполняться однократно).

Команда CPUID с входным значением EAX = 3 предназначена для чтения т.н. идентификационного номера микропроцессора – уникального 96-разрядного номера. После выполнения этой команды регистр ECX содержит младшие 32 бита идентификационного номера, а регистр EDX – средние 32 бита. Старшие 32 бита идентификационного номера – это данные, выдаваемые в регистре EAX командой CPUID с входным EDX = 1. Микропроцессоры, начиная с Pentium 4, не выводят идентификационные номера.

Для некоторых процессоров-клонов (AMD, Cyrix, IDT...) определены т.н. нестандартные (расширенные) функции команды CPUID. Они вызываются при входных значениях в EAX больших 7FFFFFFh. Полное описание этих функций вы сможете найти в технических руководствах фирм-производителей по конкретным процессорам, а здесь приведем только функцию для получения полного наименования процессора.

Команда CPUID с входными значениями в EAX=80000002h, 80000003h, 80000004h возвращает последовательно в регистрах EAX, EBX, ECX, EDX 48-ми значную ASCII строку, содержащую полное наименование процессора. Если строка короче 48 символов, то она будет дополняться пробелами. Эта строка обычно используется BIOS для вывода на экран ПК при начальной загрузке.

Перед использованием команды **CPUID** необходимо убедиться, что данная команда поддерживается процессором. Для этого необходимо попытаться изменить 21-й бит (флаг идентификации **ID**) расширенного регистра **EFLAGS**. Если бит успешно поменяется, то инструкция **CPUID** процессором поддерживается, если регистр флагов остался без изменений, то процессор команду **CPUID** не поддерживает.

Если проверка прошла успешно, необходимо определить максимальное количество поддерживаемых стандартных функций команды **CPUID** и производителя микропроцессора.

Пример 32-х разрядного приложения на Ассемблере, выводящего в консоль информацию о количестве поддерживаемых функций команды **CPUID** процессором приведен ниже.

Создание 32-х разрядных приложений на Ассемблере

Операционная система Windows в зависимости от версии позволяет запускать 16-ти, 32-х и 64-х разрядные приложения, однако 16-ти разрядные приложения поддерживаются только 32-х разрядными версиями Windows. Этот момент необходимо учитывать при создании приложения на языке Ассемблера. Современные пакеты языка Ассемблер позволяют создавать как 16-ти разрядные, так и 32-х и 64-х разрядные приложения. В дальнейшем будет рассмотрено создание 32-х разрядного приложения с использованием пакета Ассемблера **Turbo Assembler** версии 5.3 от компании Borland.

Отличительной особенностью создания 32-х разрядных приложений под Windows является отсутствие доступа к портам ввода-вывода и программным прерываниям BIOS и DOS. Взамен операционная система предоставляет свои программные средства, которые называются Win API. Win API включает в себя набор функций, находящихся в библиотеках **kernel32.dll**, **user32.dll**, **gdi32.dll**, **advapi32.dll** и др. Для того чтобы использовать некоторую функцию этих библиотек необходимо загрузить нужную библиотеку в свой исходный модуль (например, директивой **INCLUDELIB**) и указать какие функции будут внешними с помощью **EXTRN**.

Вызов API функции осуществляется командой **call**, а параметры функции передаются заранее через стек. Порядок передачи параметров в функцию определяется моделью вызова, которая указывается после задания модели памяти. В случае использования модели **STDCALL** параметры передаются слева направо и снизу вверх. Например, вызов функции с параметрами

функция (параметр 1, параметр 2, параметр 3)

на языке Ассемблера будет выглядеть следующим образом

PUSH параметр 3

PUSH параметр 2

PUSH параметр 1

CALL функция

Однако допустимо указывать передаваемые параметры в самой команде **call**:

CALL функция, параметр 1, параметр 2, параметр 3

Для создания самого простого консольного приложения понадобятся три функции из библиотеки *Kernel32.lib*, поэтому далее рассмотрим их подробнее.

Приложение в ОС Windows может создавать собственные окна или использовать уже существующее, если необходимо вывести информацию в существующую консоль. Для получения дескриптора существующего консольного окна можно использовать функцию **GetStdHandle**, в качестве аргумента которой указывается одна из трех констант

STD_INPUT_HANDLE equ -10 ;для ввода

STD_OUTPUT_HANDLE equ -11 ;для вывода

STD_ERROR_HANDLE equ -12 ;для сообщения об ошибке

Функция возвращает дескриптор в регистр EAX.

Для вывода текстовой информации в консоль используется API-функция **WriteConsole**, общий формат которой имеет следующий вид:

WriteConsole (*hConsoleOutput*, *lpBuffer*, *nNumberOfCharsToWrite*,
lpNumberOfCharsWritten, *lpReserved*),

где *hConsoleOutput* – дескриптор буфера вывода консоли;

lpBuffer – указатель на буфер, где находится выводимый текст;

nNumberOfCharsToWrite – количество выводимых символов;

lpNumberOfCharsWritten – указывает на переменную, куда будет помещено количество действительно выведенных символов;

lpReserved – резервный параметр, должен быть равен нулю

Все функции Win API, которые работают со строковыми данными, могут обрабатывать как ANSI кодировку, так и UNICODE. Добавление к имени функции суффикс «A» означает, что строковые данные должны иметь кодировку в стандарте ANSI. В дальнейшем будем использовать именно эту кодировку.

Каждая программа в конце своего выполнения обязательно должна вызвать функцию **ExitProcess** для удаления приложения из памяти.

В 32-х разрядных приложениях Windows используется так называемая плоская (**flat**) модель памяти, в которой базовые линейные адреса сегментов команд и данных равны 0, а размер достигает 4 Гбайт.

TITLE CPUID

;32-х разрядное приложение получения информации о процессоре,
;используя команду CPUID.
;Программа выводит в текущую консоль количество поддерживаемых
;процессором стандартных функций команды CPUID.

.586 ;Расширенная система команд.

;Плоская модель памяти и стандартная модель вызова подпрограмм.

.MODEL FLAT, STDCALL

;Директивы компоновщику для подключения библиотек.

INCLUDELIB import32.lib ;Работа с библиотекой ОС Kernel32.

;Внешние процедуры

EXTRN GetStdHandle: PROC ;Получить дескриптор окна.

EXTRN WriteConsoleA: PROC ;Вывести в консоль.

EXTRN ExitProcess: PROC ;Завершить процесс.

;Константы

Std_Output_Handle EQU -11 ;Консольное окно для вывода данных.

.DATA ;Открыть сегмент данных.

Handl_Out DD ? ;Дескриптор окна вывода данных.

Lens DW ? ;Количество выведенных символов.

Not_Supp DB 'CUID not supported', 13, 10, '\$'

Not_Supp_L = \$ - Not_Supp - 1 ;Длина строки Not_Supp без знака \$.

Supp DB 'CUID supported', 13, 10, '\$'

Supp_L = \$ - Supp - 1

Str1 DB 'Number function: ', '\$'

Str1_L = \$ - Str1 - 1

Number DD ?

Number_Str DB 8 dup (0)

Number_Str_L = 8

;-----

.CODE ;Открыть сегмент кодов.

Preobr PROC

mov EAX, Number

mov EBX, 10

mov word ptr Number_Str, 0 ;Очистить переменную.

```

mov word ptr Number_Str[2], 0h
mov word ptr Number_Str[4], 0h
mov word ptr Number_Str[6], 0h
xor ECX, ECX
m1: xor EDX, EDX
div EBX
push EDX
inc ECX
cmp EAX, 0
jne m1
xor ESI, ESI
m2: pop EAX
add EAX, 30h
mov Number_Str[ESI], AL
inc ESI
loop m2
ret
Preobr ENDP

```

Start:

```

;Получить дескриптор окна вывода.
call GetStdHandle, Std_Output_Handle
mov Handl_Out, EAX
;-----Проверка поддержки команды CPUID процессором-----
pushfd ;Получение регистра флагов через стек.
pop EAX
mov EBX, EAX
xor EAX, 200000h ;Изменение 21-ого бита в регистре флагов.
push EAX ;Сохранение и снова получение
popfd ;регистра флагов.
pushfd
pop EAX
cmp EAX, EBX ;Сравнение регистров до и после изменения.
jne CPUIDSupp ;Если не изменился, то CPUID не поддерживается.
;Вывести строку Not_supp.
call WriteConsoleA, Handl_Out, offset Not_Supp, Not_Supp_L, offset Lens, 0
jmp @exit
CPUIDSupp:
;Вывести строку Supp.
call WriteConsoleA, Handl_Out, offset Supp, Supp_L, offset Lens, 0

```

```

;Получение информации о количестве поддерживаемых функций CPUID
mov EAX, 0
cpuid
mov Number, EAX           ;Количество поддерживаемых функций.
call Preobr
;Добавление в конец строки служебных кодов 13 и 10.
mov word ptr Number_Str[6], 0D0Ah
;Вывести строку Str1.
call WriteConsoleA, Handl_Out, offset Str1, Str1_L, offset Lens, 0
;Вывести строку Number_Str.
call WriteConsoleA, Handl_Out, offset Number_Str, Number_Str_L, offset Lens, 0
@exit: call ExitProcess   ;Завершить программу.
END Start                ;Конец исходного модуля.

```

Трансляция исходного модуля производится программой **tasm32.exe**, общий формат вызова которой из командной строки выглядит следующим образом:

tasm32 </ключи> <имя файла>.

Подробную информацию о возможных ключах программы **tasm32.exe** можно получить, вызвав справку командой **tasm32 /?**

Компоновка объектных модулей производится вызовом компоновщика **tlink32.exe** из командной строки

tlink32 /Tpe /ap /v <имя файла>.

где **/Tpe** – ключ создания 32-х разрядного exe-приложения;

/ap – ключ создания 32-х разрядного консольного приложения;

/v – ключ, передающий в загрузочный файл информацию, используемую при отладке программ.

Средство администрирования WMI и утилита WMIC

Среди инструментов и средств автоматизации в операционной системе Windows особое место занимает технология **Windows Management Instrumentation (WMI)**. Технология **WMI** – это созданная фирмой Microsoft реализация модели управления предприятием на базе Web (Web-Based Enterprise Management, WBEM), которая разработана и принята рабочей группой по управлению распределенными системами (Distributed Management Task Force, DMTF), при участии таких компаний, как BMC Software, Cisco Systems, Intel и Microsoft. Задачей WBEM была разработка таких стандартов для удаленного управления информационной средой предприятия, которые позволили бы управлять всеми физическими и логическими компонентами этой

среды из одной точки и не зависели бы при этом от конкретного оборудования, сетевой инфраструктуры, операционной системы, файловой системы и т. д. Для этого была предложена модель CIM (Common Information Model), которая представляет физическую и логическую структуры компьютерной системы в виде единой расширяемой объектно-ориентированной информационной модели и определяет единые интерфейсы для получения информации о любом компоненте этой модели.

Технология WMI – это глобальная концепция настройки, управления и слежения за работой различных частей корпоративной компьютерной сети под управлением платформы Windows.

Для доступа к WMI можно использовать различные механизмы. Приложения могут обращаться к WMI через WMI COM API. Web-приложения могут использовать средства управления ActiveX для создания сетевых интерфейсов к данным WMI. Системные администраторы могут создавать сценарии, выполняемые в среде Windows Script Host (WSH), или формировать запросы через консольную утилиту WMIC. В данной работе для доступа к средствам WMI будет использоваться консольная утилита WMIC (WMI command-line), которая присутствует в операционной системе начиная с версии Windows XP.

WMIC позволяет выполнять следующие задачи:

- просматривать схемы WMI и запрашивать их классы и экземпляры (обычно с использованием <псевдонимов>, упрощающих работу с WMI);
- работать с локальным компьютером, удаленными компьютерами или выполнять команды сразу для нескольких компьютеров;
- настраивать псевдонимы и форматы вывода в соответствии с имеющимися потребностями;
- создавать и выполнять сценарии на основе WMIC.

Общий формат команды:

wmic [глобальные параметры] <команда>

Подробную информацию о допустимых параметрах и их командах, которые поддерживает WMIC, можно получить, используя следующие команды:

wmic /? - отобразить общую справку.

wmic /?:FULL - отобразить полную справку.

Рассмотрим запрос получения информации о системной плате:

wmic BASEBOARD get /value / more,

где **BASEBOARD** – псевдоним класса системной платы ПЭВМ;

`get` – получение свойств псевдонима;

`/value` – параметр команды `get`, который возвращает значения в виде списка с названием свойства;

| `more` – отображения данных в постраничном режиме.

Приведем ещё несколько примеров запросов WMI:

`wmic OS get OSArchitecture /value` – отобразить архитектуру операционной системы (разрядность Windows);

`wmic DISKDRIVE get Name, Size, Model` – отобразить список физических дисков, содержащих название модели, имя в системе и размер;

`wmic MEMORYCHIP get DeviceLocator ,Capacity` – отобразить сведения о размещении и емкости модулей памяти DIMM.

Основные псевдонимы, используемые для получения информации о конфигурации системы, представлены в таблице 1.6.

Таблица 1.6

Некоторые псевдонимы классов WMI

Псевдоним	Описание
BASEBOARD	Управление системной платой
BIOS	Управление BIOS
CDROM	Управление устройствами чтения компакт-дисков
CPU	Управление ЦП
DISKDRIVE	Управление физическими дисками
LOGICALDISK	Управление локальными запоминающими устройствами
MEMORYCHIP	Информация о микросхемах памяти
OS	Управление установленными операционными системами
SYSTEMSLOT	Управление точками физических соединений, включая порты, гнезда и периферийные устройства, а также специальными точками соединения

Системные ресурсы

Каждое периферийное устройство в зависимости от способа его подключения к ПК имеет право пользоваться системными ресурсами этого компьютера. Как правило, к системным ресурсам относятся:

- адреса портов ввода-вывода;
- адреса ячеек памяти;
- каналы запросов прерываний (IRQ);

Порты ввода-вывода позволяют установить связь между устройствами и программным обеспечением в компьютере. В современном компьютере 65 535 портов, пронумерованных от 0000h до FFFFh. Стандартное назначение адресов портов ввода-вывода Вы можете найти в рекомендованной литературе.

Каналы запросов прерывания (IRQ), или *аппаратные прерывания*, используются различными устройствами для сообщения системной плате (процессору) о необходимости обработки определенного запроса. В таблице 1.7 приведены назначения основных аппаратных прерываний в порядке убывания их приоритета.

Таблица 1.7

Аппаратные прерывания (в порядке убывания приоритета)

№ IRQ	Стандартная функция
0	Системный таймер
1	Контроллер клавиатуры
2	Резерв или второй контроллер прерываний
8	CMOS RTC – часы реального времени
9	Доступно (может использоваться вместо IRQ2)
10	Доступно
11	Доступно
12	PS/2 – мышь или доступно
13	Арифметический сопроцессор
14	Первичный IDE-канал
15	Вторичный IDE-канал или доступно
3	Последовательный порт (COM2, COM4)
4	Последовательный порт (COM1, COM3)
5	Звуковая плата или параллельный порт (LPT2)
6	Контроллер гибких дисков
7	Параллельный порт (LPT1)

Лабораторная работа N2

Исследование S.M.A.R.T.-атрибутов жестких дисков

1. Цель работы

Практическое овладение навыками получения и анализа S.M.A.R.T.- атрибутов жестких дисков.

2. Теоретический материал

2.1 Технология S.M.A.R.T.

S.M.A.R.T. (Self-Monitoring Analysis and Reporting Technology) – это технология внутренней оценки состояния диска, и механизм предсказания возможного выхода из строя жесткого диска. S.M.A.R.T. производит наблюдение за основными параметрами накопителя. Эти параметры называются **атрибутами**.

Атрибуты S.M.A.R.T. – особые характеристики, которые используются при анализе состояния и запаса производительности накопителя. Каждый производитель имеет свой характерный набор атрибутов и может свободно вносить изменения в этот набор в соответствии со своими требованиями.

Одной из основных функций системы мониторинга является возможность самоконтроля состояния жесткого диска. Выполнение данной тестовой процедуры может быть осуществлено как самим накопителем, не занятым клиентским заданием, так и пользователем, осуществляющим проверку атрибутов S.M.A.R.T. посредством специализированного программного обеспечения. В любом случае, чтобы начать принудительный процесс проверки, следует подать интерфейсную команду Smart Execute Offline Immediate. По прошествии некоторого времени, требуемого для получения финального результата, накопитель сохраняет полученные данные в специализированных атрибутах и журналах. Результаты тестирования используются накопителем для сравнения с полученными ранее данными. Таким образом, можно наблюдать тенденцию изменения атрибутов, что позволит делать выводы о примерном выходе из строя жесткого диска в целом.

Каждый атрибут имеет свой уникальный идентификатор – **ID**. Он характеризует некоторую реальную величину, например, количество изношенных секторов или общее время работы, на основании которой можно делать выводы о надежности конструкции в целом. Большинство жестких дисков, поддерживающих S.M.A.R.T., обычно имеют от 3 до 30 атрибутов.

Значения всех атрибутов надежности (value) обычно находятся в диапазоне от 1 до 253 включительно, но могут быть и в другом диапазоне. При производстве жесткого диска каждый атрибут получает максимальное значение. Постепенно, по мере износа накопителя, значения атрибутов надежности уменьшаются. Соответственно, высокое значение атрибутов говорит о низкой вероятности выхода жесткого диска из строя, и, наоборот, низкое значение атрибутов – о низкой его надежности и высокой вероятности скорого отказа.

Диапазон изменения атрибутов не стандартизирован. Каждый производитель вносит свою лепту в данную технологию. Так, например, для продуктов, произведенных компанией Hitachi Data Storage, максимальная величина каждого атрибута составляет 100 единиц. Для Samsung это число равно 253. Наибольшую путаницу внесли инженеры компании Western Digital, поскольку для своих продуктов они используют довольно странную методику измерений. Так, верхняя граница первого атрибута надежности составляет 200 единиц, а остальных – 100.

При работе накопителя ведутся журналы ошибок и значений атрибутов, в которых хранится информация о нескольких последних ошибках и **худших значениях** атрибутов с момента первого запуска жесткого диска.

Для каждого атрибута надежности разработчиками жестких дисков определяется **пороговое значение**, называемое Threshold, по достижении которого устройство можно считать небезопасным для хранения данных.

Помимо текущего значения, описывающего состояния атрибутов, имеются **необработанные (raw) значения**, которые несут определенный смысл для каждого атрибута. Например, необработанное значение атрибута *Power-On Hours* (Наработка в часах) является счетчиком единиц времени (часов, минут, секунд и т.п.), в течении которого жесткий диск находился в работающем состоянии.

Также каждый атрибут имеет **флаги**, указывающие на назначение атрибута; атрибут может иметь несколько флагов одновременно. Флаги атрибута:

- Жизненно-важный (LC) – атрибут, непосредственно описывающий надежность диска.
- Атрибут, отражающий производительность диска (PR).
- Атрибут, отражающий частоту появления ошибок (ER).
- Счетчик событий (EC) – означает, что атрибут используется как счетчик каких-либо событий.

- Самосохраняющийся атрибут (SP) – атрибут, значения которого автоматически сохраняются и восстанавливаются каждый раз, когда производятся тесты S.M.A.R.T.

- Коллекция реального времени (OC) – значения этого атрибута вычисляются во время проведения тестов реального времени.

Краткое описание основных атрибутов надежности S.M.A.R.T. приведено в таблице 2.1. Данная таблица включает не полный список всех атрибутов S.M.A.R.T., более полный список можно найти в рекомендуемой литературе.

Таблица 2.1

Основные S.M.A.R.T. атрибуты накопителей

№ атрибута	Имя атрибута	Описание
1	Raw Read Error Rate	уровень ошибок при чтении данных с диска, происхождение которых обусловлено аппаратной частью диска.
3	Spin Up Time	время раскрутки шпинделя диска из состояния покоя до рабочей скорости.
4	Start/Stop Count	полное число запусков/остановов шпинделя. У дисков некоторых производителей (Seagate, например) – счетчик включения режима энергосбережения. В поле raw value хранится общее количество запусков/остановов диска.
5	Reallocated Sectors Count	количество переназначенных секторов. Когда диск обнаруживает ошибку чтения/записи, он помечает сектор «переназначенным», и переносит данные в специально отведенную область. Чем меньше значение, тем хуже состояние поверхности дисков. Поле raw value содержит общее количество переназначенных секторов.
7	Seek Error Rate	частота ошибок при позиционировании блока головок. Чем их больше, тем хуже состояние механики и/или поверхности жесткого диска.
9	Power-On Hours	число часов, проведенных во включённом состоянии. В качестве порогового значения для него выбирается паспортное время наработки на отказ.
10	Spin-Up Retry Count	число повторных попыток раскрутки дисков до рабочей скорости, в случае если первая попытка была неудачной. Ненулевое значение свидетельствует о проблемах в механической части накопителя.
12	Device Power Cycle Count	количество полных циклов включения-выключения диска.
194 (231)	Temperature	показания встроенного термодатчика.

Продолжение таблицы 2.1

№ атрибута	Имя атрибута	Описание
197	Current Pending Sector Count	число подозрительных или нестабильных секторов, являющихся кандидатами на замену. Они не были ещё определены как плохие, но считывание их отличается от чтения стабильного сектора. В случае успешного последующего прочтения сектора он исключается из числа кандидатов. В случае повторных ошибочных чтений накопитель пытается восстановить его и выполняет операцию переназначения.
198	Uncorrectable Sector Count	число неисправимых ошибок при обращении к сектору. В случае увеличения числа ошибок велика вероятность критических дефектов поверхности и/или механики накопителя.
199	UltraDMA CRC Error Count	число ошибок, возникающих при передаче данных по внешнему интерфейсу.
200	Write Error Rate / Multi-Zone Error Rate	показывает общее количество ошибок, происходящих при записи сектора. Может служить показателем качества поверхности и механики накопителя.

2.2 Получение информации о S.M.A.R.T. атрибутах

Для получения значений S.M.A.R.T. атрибутов жёстких дисков на сегодняшний день существует множество платных и бесплатных программ, отличающихся дополнительными функциями при обработке данных. К наиболее известным программам данного типа относятся: *HDLife*, *Hard Drive Inspector*, *HD Tune*, *PCS*, *ActivSMART* и др. Однако с помощью несложных команд возможно написание собственных программ для получения значений атрибутов и предсказания выхода из строя жёстких дисков. Для получения значения атрибутов необходимо обратиться или непосредственно к жёсткому диску или к драйверу мини-порта контроллера SCSI.

Пример алгоритма программы для получения значений атрибута показан на рис. 2.1 и 2.2. Данная программа опрашивает первые десять накопителей и выводит значения атрибута по каждому накопителю в текущее консольное окно. В случае отсутствия накопителей или запуска программы без прав администратора выводится сообщение об ошибке. В приложении А приведён пример листинга 32-х разрядной программы на языке Ассемблера, которая получает значения атрибута температуры и выводит их в консольное окно. В связи с различной интерпретацией текущего значения температуры у разных производителей ориентироваться необходимо на необработанное значение атрибута.

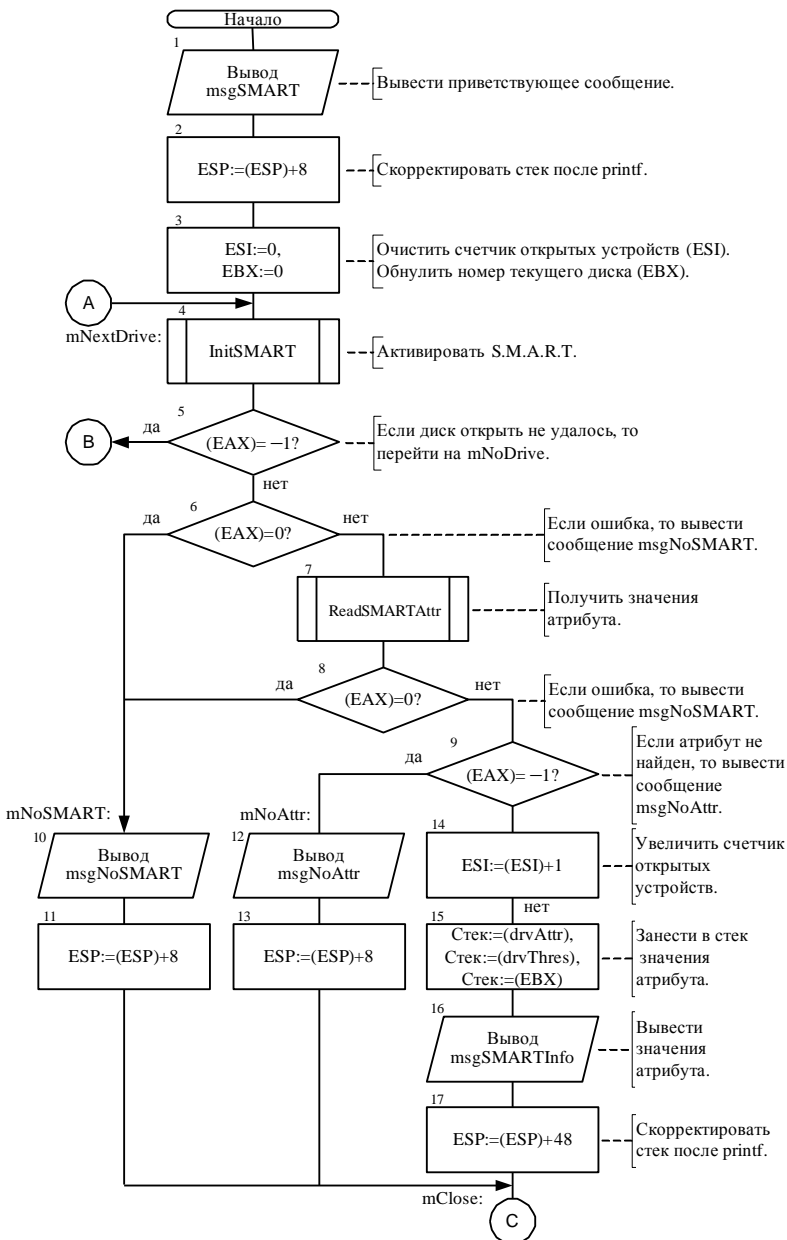


Рис. 2.1 – Алгоритм получения значений SMART атрибута накопителей

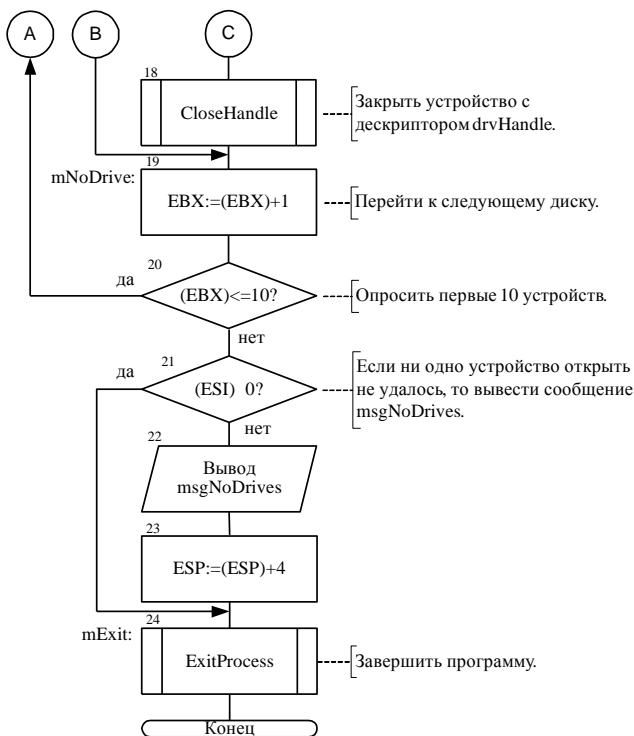


Рис. 2.2 – Продолжение алгоритма получения значений SMART атрибута накопителей

Чтение S.M.A.R.T. атрибутов можно выполнить с помощью функции **DeviceIoControl**, которая отправляет управляющий код непосредственно указанному драйверу устройства, заставляя соответствующее устройство выполнять указанную операцию.

Перед выполнением функции **DeviceIoControl** необходимо прочитать дескриптор (манипулятор или handle) диска «как файл» функцией **CreateFile**. Данная функция имеет следующие параметры:

1. *lpFileName* – указатель на символьную строку, устанавливающую имя открываемого объекта. Строка должна заканчиваться нулем;
2. *dwDesiredAccess* – режим доступа к объекту;
3. *dwShareMode* – режим совместного использования;
4. *lpSecurityAttributes* – параметры безопасности, который устанавливает возможность наследования возвращенного дескриптора дочерними процессами. Чаще всего данный параметр равен нулю;
5. *dwCreationDisposition* – выполняемое действие;

6. *dwFlagsAndAttribute* – атрибуты и флаги файла;
7. *hTemplateFile* – дескриптор шаблона файла.

В случае успешного выполнения функция **CreateFile** вернёт в регистр EAX дескриптор (хендл) объекта, имя которого было указано в первом параметре, в случае неудачи вернёт значение **INVALID_HANDLE_VALUE** (а именно -1 или 0FFFFFFFFh).

Более подробную информацию о параметрах функции можно найти в рекомендованной литературе, а пример получения дескриптора диска с помощью функции **CreateFile** приведен в процедуре **InitSMART** программы **SMART**.

*Данная функция работает, если программа запущена с правами администратора, в противном случае функция **CreateFile** завершится с ошибкой и возвращает значение – **INVALID_HANDLE_VALUE**.*

После успешного открытия устройства можно работать с ним, используя функцию **DeviceIoControl**. Она принимает восемь параметров:

1. *hDevice* – имя дескриптора устройства, над которым должна выполняться операция;
2. *dwIoControlCode* – управляющий код для операции;
3. *lpInBuffer* – указатель на буфер ввода данных, который содержит данные необходимые, чтобы выполнить операцию;
4. *nInBufferSize* – размер буфера ввода данных, в байтах;
5. *lpOutBuffer* – указатель на буфер вывода данных, который должен получить данные, возвращенные операцией;
6. *nOutBufferSize* – размер буфера вывода данных, в байтах;
7. *lpBytesReturned* – указатель на переменную, которая получает размер данных, сохраненных в буфере вывода данных, в байтах;
8. *lpOverlapped* – указатель на структуру **OVERLAPPED** необходимую для асинхронных операций.

Для передачи жесткому диску команды активации S.M.A.R.T. в качестве управляющего кода необходимо указать константу **DFP_SEND_DRIVE_COMMAND** (0007C084h), а для получения данных от диска константу **DFP_RECEIVE_DRIVE_DATA** (0007C088h). Сами команды S.M.A.R.T. указываются в буфере ввода данных, который удобно оформить в виде структуры данных (таблица 2.2). Для активации S.M.A.R.T. в поле **bFeaturesReg** необходимо занести константу **IDE_SMART_ENABLE** (0D8h), а для получения текущего и порогового значений атрибута константы **IDE_SMART_READ_ATTRIBUTES** (0D0h) и **IDE_SMART_READ_THRESHOLDS** (0D1h) соответственно. Номер диска/головки IDE для поля **bDriveHeadReg** вычисляется по формуле [0A0h or ((Drive and 1) shl 4)].

Структура буфера вывода данных представлена в таблице 2.3. Так как буферы вывода данных для текущих и пороговых значений имеют одинаковую структуру, то желательно создать два буфера с разными именами. Значения всех атрибутов возвращаются в массиве bBuffer со смещение в 2 байта. Его состав для текущих и пороговых значений показан в таблице 2.4.

Таблица 2.2

Структура буфера ввода данных для команды DeviceIOControl

Имя структуры записи	Элемент структуры	Размер элемента, байт	Значение элемента	Описание
SendCmd-InParams (SCIP)	cBufferSize	4	512	размер буфера в байтах
	irDriveRegs		IDERegs	структура регистров диска
	bDriveNumber	1		физический номер диска
	bReserved	3		зарезервировано
	dwReserved	16		зарезервировано
	bBuffer	1		входной буфер
IDERegs	bFeaturesReg	1		регистр подкоманды S.M.A.R.T.
	bSectorCountReg	1	1	регистр количества секторов диска
	bSectorNumberReg	1	1	регистр номера сектора диска
	bCylLowReg	1	4Fh	младший разряд номера цилиндра диска
	bCylHighReg	1	C2h	старший разряд номера цилиндра диска
	bDriveHeadReg	1		регистр диска/головки
	bCommandReg	1	B0h	регистр фактической команды
	bReserved	1	0	зарезервировано

Таблица 2.3

Структура буфера вывода данных для команды DeviceIOControl

Имя структуры записи	Элемент структуры	Размер элемента, байт	Описание
SendCmd-OutParams (SCOP)	cBufferSize	4	размер буфера в байтах
	DriverStatus	DriverStatus	структура состояния диска
	bBuffer	массив	буфер произвольной длины для сохранения данных, прочитанных с диска
DriverStatus	bDriverError	1	код ошибки драйвера
	bIDEStatus	1	содержание регистра ошибки
	bReserved	1	зарезервировано
	dwReserved	2	зарезервировано

Таблица 2.4

Структура возвращаемых данных в bBuffer

Имя структуры записи	Элемент структуры	Размер элемента, байт	Описание
DriveAttribute	bAttrID	1	идентификатор атрибута
	wStatusFlags	2	флаги состояния
	bAttrValue	1	текущее нормализованное значение
	bWorstValue	1	худшее значение
	bRawValue	6	текущее ненормализованное значение
	bReserved	1	зарезервировано
AttrThreshold	bAttrID	1	идентификатор атрибута
	bWarrantyThreshold	1	пороговое значение
	bReserved	10	зарезервировано

При выводе информации в консольное окно удобно использовать API-функцию **printf**. Эта функция выводит отформатированный текст согласно указанному формату в стандартный выходной поток. Функция может иметь несколько параметров: первый параметр – *указатель на буфер* (строку), который будет форматироваться; следующие параметры – *аргументы, которые будут подставляться в исходную строку*. При выводе **printf** начинает печатать символы, которые находит по адресу, указанному в первом параметре, символ за символом. Когда в этом процессе попадается символ процента (%), то **printf** знает, что это спецификатор формата – специальный набор символов, который задает, как надо вывести число. Следующий по порядку параметр, указанный в **printf**, выводится так, как указано в спецификаторе формата. Затем процесс обработки символов (вывод их на печать) первого параметра продолжается. Если опять встретится знак процента, то будет выведен следующий параметр в указанном формате, после чего вывод строки продолжится до тех пор, пока не встретиться нулевой символ (символ с кодом 0).

Формат задается с помощью *спецификатора*, начинающегося со знака процента. Существует достаточно много опций для задания формата спецификатора из которых рассмотрим только три: модификатор размера, модификатор типа и опция заполнения лидирующими нулями. Размер зависит от типа модификатора, поэтому указывается совместно с ним. Размер определяет количество символов, которое будет выведено, однако если цифр в числе меньше, чем задано в спецификаторе, то число выведется с пробелами слева. Чтобы придать наглядный вид

числу при выводе, можно воспользоваться опцией заполнения лидирующими нулями, которая задается указанием нуля после знака процента.

Тип спецификатора задается в виде одного символа и является обязательным в спецификаторе. В нашем случае используются несколько типов:

%u – выводит на печать целое десятичное число без знака;

%x или **%X** – выводит на печать целое шестнадцатеричное число строчными или прописными символами, соответственно.

Пример использования функции **printf** на Ассемблере:

```
.data
msg      db 'Вы учитесь на %u курсе', 13, 10, 0
course    dw 4
.code
start:  movzx EBX, course
        call  printf, offset msg, EBX
        add   ESP, 4
```

Поскольку в Ассемблере параметры функции передаются через стек, то в указанном примере выполняется предварительное преобразование размера переменной *course* из байта в двойное слово, используя регистр EAX.

Если функция выполнена успешно, то в регистр EAX будет возвращена длина скопированной строки.

В отличие от других WinAPI-функций **printf** не знает сколько параметров может быть в неё отправлено, поэтому после её выполнения программист обязан освободит стек. Стек освобождается командой

ADD ESP, N,

где N – это количество освобождаемых байтов.

2.3 Использование структур данных в Ассемблере

Структура – конструкция, объединяющая набор переменных разных типов. Переменные, образующие структуру, называются элементами структуры или полями.

Для использования структур в программе необходимо выполнить три действия:

- 1) описать структуру;
- 2) определить экземпляр структуры;
- 3) организовать обращение к элементам структуры.

Описать структуру в программе означает указать ее шаблон. Этот шаблон можно рассматривать лишь как информацию для транслятора о расположении полей и их значении по умолчанию. Описание структуры можно располагать в любом месте программы, но до описания конкретных структурных переменных. Транслятор, встретившись с описанием структуры, не транслирует её текст, т.е. не выделяет место в памяти, а запоминает приведенное описание, чтобы воспользоваться им в дальнейшем, если в программе встретится объявления переменных типа этой структуры.

Для описания структур данных в Ассемблере имеется директива **STRUC**. Общий форма описания структур данных выглядит следующим образом:

```
<имя структуры> STRUC  
  <описание полей структуры>  
<имя структуры> ENDS
```

Здесь <описание полей> представляет собой последовательность директив описания данных **DB**, **DW**, **DD** и др. Их операнды определяют размер полей и, при необходимости, начальные значения. Этими значениями будут, возможно, инициализироваться соответствующие поля при определении структуры. Пример описания структуры:

```
point STRUC  
    x  DW  0  
    y  DW  0  
    z  DW  0  
    color DB  3 DUP (?)  
point ENDS
```

Для использования описанной с помощью шаблона структуры в программе необходимо *определить переменную* с типом данной структуры. Для этого используется следующая синтаксическая конструкция:

[имя переменной] *имя структуры* <[список значений]>,

где *имя переменной* – идентификатор переменной данного структурного типа;

список значений – заключенный в угловые скобки список начальных значений элементов структуры, разделенных запятыми. Если при определении новой переменной с типом данной структуры нет необходимости менять значения по умолчанию, которые записаны в шаблоне, то необходимо оставить пустым содержимое угловых скобок.

Используя выше указанный шаблон структуры можно инициализировать несколько переменных:

```
point1 point <>
point2 point <100, 200, 1, 255, 255, 255>
```

При этом значения полей переменной point1 будут взяты из шаблона.

Для того чтобы сослаться в команде на поле некоторой структуры, используется специальный оператор – символ « . » (точка). Он используется в следующей синтаксической конструкции:

имя переменной . имя поля структуры,

где *имя переменной* – идентификатор переменной некоторого структурного типа;

имя поля структуры – имя поля из шаблона структуры.

Пример:

```
mov AX, point2.y
```

3. Подготовка к работе

3.1. Изучить методические указания и рекомендованную литературу.

3.2. Подготовить ответы на контрольные вопросы.

4. Задание на выполнение работы

4.1. Используя текстовый редактор, создать и отредактировать исходный модуль программы **SMART.asm** таким образом, чтобы она выводила значения атрибута в соответствии с вариантом задания. Варианты задания указаны в таблице 2.5.

4.2. По полученным значениям сделать выводы о производительности и «здоровье» жёсткого диска.

Таблице 2.5

Варианты заданий

№ варианта	№ атрибута	№ варианта	№ атрибута
1	1	7	10
2	3	8	12
3	4	9	197
4	5	10	198
5	7	11	199
6	9	12	200

TITLE SMART

;32-х разрядное приложение получения значений S.M.A.R.T. атрибута
жесткого диска.
;Программа выводит в текущую консоль номер атрибута и его описание,
;текущее нормированное, худшее, пороговое и необработанное значения для
;каждого диска.

.386 ;Расширенная система команд.
;Плоская модель памяти и стандартная модель вызова подпрограмм.
.MODEL FLAT, STDCALL
;Директивы компоновщику для подключения библиотек.
INCLUDELIB import32.lib ;Работа с библиотекой ОС Kernel32.

;--- Внешние WinAPI-функции -----

EXTRN CreateFileA:	PROC	;Получить дескриптор устройства.
EXTRN DeviceIoControl:	PROC	;Получить информацию об устройстве.
EXTRN CloseHandle:	PROC	;Закрыть дескриптор устройства.
EXTRN printf:	PROC	;Вывести текст по шаблону.
EXTRN RtlZeroMemory:	PROC	;Очистить память.
EXTRN ExitProcess:	PROC	;Завершить процесс.

;--- Константы и структуры -----

;стандартные значения WinApi
GENERIC_READ = 80000000h ;допускается чтение
GENERIC_WRITE = 40000000h ;допускается запись
FILE_SHARE_READ = 1 ;допускается чтение другими процессами
FILE_SHARE_WRITE = 2 ;допускается запись другими процессами
OPEN_EXISTING = 3 ;предписывает открывать устройство,
;если оно существует

READ_ATTRIBUTE_BUFFER_SIZE = 512 ;размер буфера атрибутов
READ_THRESHOLD_BUFFER_SIZE = 512 ;размер буфера пороговых значений
DFP_SEND_DRIVE_COMMAND = 0007C084h ;управляющие коды для
DFP_RECEIVE_DRIVE_DATA = 0007C088h ;функции DeviceIoControl

IDE_SMART_ENABLE = 0D8h ;запрос на активацию S.M.A.R.T.
IDE_SMART_READ_ATTRIBUTES = 0D0h ;запрос на чтение атрибутов
IDE_SMART_READ_THRESHOLDS = 0D1h ;запрос на чтение порогового значения
IDE_COMMAND_SMART = 0B0h ;команда IDE для работы со S.M.A.R.T.
SMART_CYL_LOW = 4Fh ;младший байт цилиндра для S.M.A.R.T.
SMART_CYL_HI = 0C2h ;старший байт цилиндра для S.M.A.R.T.

ATTR_NAME equ <'Temperatura'> ;название атрибута

; Структура записи регистров IDE

```

IDERegs  STRUC
    bFeaturesReg      db ?      ;регистр подкоманды SMART
    bSectorCountReg   db ?      ;регистр количества секторов IDE
    bSectorNumberReg   db ?      ;регистр номера сектора IDE
    bCylLowReg         db ?      ;младший разряд номера цилиндра IDE
    bCylHighReg        db ?      ;старший разряд номера цилиндра IDE
    bDriveHeadReg      db ?      ;регистр номера диска/головки IDE
    bCommandReg        db ?      ;фактическая команда IDE
                                db ?      ;зарезервировано. Должен быть 0
IDERegs  ENDS

; Структура записи входных параметров для функции DeviceIOControl
SendCmdInParams  STRUC
    cBufferSize      dd ?      ;размер буфера в байтах
    irDriveRegs       IDERegs <> ;структура со значениями регистров диска
    bDriveNumber      db ?      ;физ. номер диска для отправки команд SMART
                                db 3 dup (?) ;зарезервировано
                                dd 4 dup (?) ;зарезервировано
    bInBuffer          label byte ;входной буфер
SendCmdInParams  ENDS

; Структура состояния диска
DriverStatus  STRUC
    bDriverError      db ?      ;код ошибки драйвера
    bIDEStatus         db ?      ;содержит значение регистра ошибки IDE
                                ;контроллера, только когда bDriverError = 1
                                db 2 dup (?) ;зарезервировано
                                dd 2 dup (?) ;зарезервировано
DriverStatus  ENDS

; Структура записи параметров, возвращаемых функцией DeviceIOControl
SendCmdOutParams  STRUC
    cBufferSize      dd ?      ;размер буфера в байтах
    DrvStatus         DriverStatus <> ;структура состояния диска
    bOutBuffer         label byte ;буфер произвольной длины для хранения
                                ;данных, полученных от диска
SendCmdOutParams  ENDS

; Структура записи значений атрибутов
DriveAttribute  STRUC
    bAttrID           db ?      ;идентификатор атрибута
    wStatusFlags       dw ?      ;флаг состояния
    bAttrValue         db ?      ;текущее нормализованное значение
    bWorstValue        db ?      ;худшее значение
    bRawValue          db 6 dup (?) ;текущее ненормализованное значение
                                db ?      ;зарезервировано

```


DriveAttribute ENDS

; Структура записи порогового значения атрибута

AttrThreshold STRUC

 bAttrID db ? ; идентификатор атрибута

 bWarrantyThreshold db ? ; пороговое значение

 db 10 dup (?) ; зарезервировано

AttrThreshold ENDS

;--- Данные -----

.DATA

sDrive DB '\\.\PhysicalDrive' ; имя устройства,

cDrive DB '0', 0 ; включая его номер

msgSMART DB 'Чтение значений S.M.A.R.T. атрибута '

 DB 'всех подключенных дисков...', 13, 10, 0

msgNoSMART DB 'HDD%u не поддерживает '

 DB 'технологии S.M.A.R.T.!', 13, 10, 0

msgNoAttr DB 13, 10, 'Атрибут ',ATTR_NAME,' для HDD%u '

 DB 'не найден!',13,10,0

msgNoDrives DB 13, 10, 'Не удалось открыть ни одного устройства '

 DB '(видимо, у Вас нет прав администратора)', 13, 10, 0

msgSMARTInfo DB 13, 10, 'Значения атрибута для HDD%u', 13, 10

 DB 'Attribute number = %u (' , ATTR_NAME, ')', 13, 10

 DB 'Attribute value = %u', 13, 10

 DB ' Worst value = %u', 13, 10

 DB ' Threshold value = %u', 13, 10

 DB ' RAW value = %02X %02X %02X %02X %02X %02X (hex)', 13, 10, 0

drvHandle DD ? ; дескриптор устройства

SCIP SendCmdInParams <> ; указатель на структуру входных параметров

SCOP SendCmdOutParams <> ; указатель на структуру выходных параметров

 DB 512 dup (?) ; зарезервированное место для буфера

bReturned DD ? ; число байт, возвращённых функцией

drvAttr DriveAttribute <> ; указатель на структуру значений атрибута

drvThres AttrThreshold <> ; указатель на структуру с пороговым значением

;--- Основной код -----

.CODE

Start:

call printf, offset msgSMART ; Выводим приветствие.

add esp, 4*2 ; Корректируем стек.

xor esi, esi ; esi - счетчик успешно открытых устройств

mov ebx, 0 ; ebx - номер текущего диска

mNextDrive:

```

call    InitSMART, ebx          ;Активация S.M.A.R.T.
test    eax, eax                ;Проверяем результат.
js      mNoDrive                ;Переходим, если диск открыть не удалось (eax = -1).
jz      mNoSMART                ;Переходим, если ошибка (eax = 0).
; Чтение S.M.A.R.T. атрибутов
call    ReadSMARTAttr, ebx, offset drvAttr, offset drvThres
test    eax, eax                ;Проверяем результат
jz      mNoSMART                ;Переходим, если ошибка (eax = 0)
js      mNoAttr                 ;Переходим, если атрибут не найден (eax = -1)
inc     esi                     ;Увеличиваем значение счетчика открытых устройств.
; Заносим в стек значения в обратном порядке для передачи параметров printf
movzx   eax, drvAttr.bRawValue[0]
push    eax
movzx   eax, drvAttr.bRawValue[1]
push    eax
movzx   eax, drvAttr.bRawValue[2]
push    eax
movzx   eax, drvAttr.bRawValue[3]
push    eax
movzx   eax, drvAttr.bRawValue[4]
push    eax
movzx   eax, drvAttr.bRawValue[5]
push    eax
movzx   eax, drvThres.bWarrantyThreshold
push    eax
movzx   eax, drvAttr.bWorstValue
push    eax
movzx   eax, drvAttr.bAttrValue
push    eax
movzx   eax, drvAttr.bAttrID
push    eax
push    ebx
call     printf, offset msgSMARTInfo    ;Выводим значения атрибута.
add     esp, 4*12

mClose:
call    CloseHandle, drvHandle          ;Закрываем устройство.

mNoDrive:
inc     ebx                            ;Увеличиваем номер диска.
cmp     ebx, 10                        ;Сравниваем его с максимальным.
jbe     mNextDrive                     ;Повторяем цикл, если он не превышает 10
test    esi, esi                       ;Проверяем кол-во успешно открытых устройств.
jnz     mExit                          ;Переходим, если их больше 0,
call     printf, offset msgNoDrives     ;иначе выводим сообщение об ошибке.
add     esp, 4*1

```

```

mExit:
    call    ExitProcess, 0                ;Выходим из программы.

mNoAttr:                                ;В случае ошибки чтения атрибута
    call    printf, offset msgNoAttr, ebx ;выводим сообщение об ошибке.
    add     esp, 4*2
    jmp     mClose

mNoSMART:                               ;В случае ошибки активации S.M.A.R.T.
    call    printf, offset msgNoSMART, ebx ;выводим сообщение об ошибке.
    add     esp, 4*2
    jmp     mClose

;== Активация S.M.A.R.T. для диска номер Drive =====
; Возвращает EAX = 1 - успешное завершение, 0 - ошибка (нет прав),
; -1 - диск не найден
InitSMART    PROC
    ARG     Drive: DWORD

    mov     al, byte ptr Drive
    add     al, '0'                    ;Превращаем номер диска в символ '0', '1' и т.д.
    mov     cDrive, al                ;Сохраняем его в cDrive (корректируем строку sDrive)
; Получаем дескриптор диска
    call    CreateFileA, offset sDrive, GENERIC_READ or GENERIC_WRITE,
FILE_SHARE_READ or FILE_SHARE_WRITE, 0, OPEN_EXISTING, 0, 0
    cmp     eax, -1                    ;В случае ошибки
    je      mIExit                    ;выходим, возвращая -1
    mov     drvHandle, eax             ;Сохраняем дескриптор в drvHandle
; Подготавливаем буфер SCIP для активации S.M.A.R.T
    call    InitSMARTInBuf, Drive, IDE_SMART_ENABLE, offset SCIP, 0
; Активируем S.M.A.R.T. для текущего диска
    call    DeviceIoControl, drvHandle, DFP_SEND_DRIVE_COMMAND, offset
SCIP, size SCIP, offset SCOP, size SCOP, offset bReturned, 0
    test    eax, eax                  ;Проверяем успешное выполнение команды
    jz      mIExit                    ;В случае ошибки возвращаем 0,
    mov     eax, 1                    ;иначе 1.
mIExit:    ret
InitSMART  ENDP

;== Чтение значений S.M.A.R.T. атрибута =====
; DriveAttr и AttrThres – буферы для значений атрибута,
; Drive – номер диска.
; Возвращает EAX <> 1 - успешное завершение, 0 - ошибка (нет прав),
; -1 - атрибут не найден.
ReadSMARTAttr PROC

```

```

ARG    Drive: DWORD, DriveAttr: DWORD, AttrThres: DWORD
uses   esi, edi                ;Сохраняем значения регистров в стеке

; Определяем размер буферов под значения атрибутов
ATTR_SIZE=size SendCmdOutParams+READ_ATTRIBUTE_BUFFER_SIZE
THRES_SIZE=size SendCmdOutParams+READ_THRESHOLD_BUFFER_SIZE
; Очищаем буферы для значений атрибута.
call   RtlZeroMemory, DriveAttr, size DriveAttribute
call   RtlZeroMemory, AttrThres, size AttrThreshold

; Подготавливаем буфер SCIP для чтения атрибутов S.M.A.R.T.
call   InitSMARTInBuf, Drive, IDE_SMART_READ_ATTRIBUTES, offset
SCIP, READ_ATTRIBUTE_BUFFER_SIZE
; Читаем значения атрибутов
call   DeviceIoControl, drvHandle, DFP_RECEIVE_DRIVE_DATA, offset
SCIP, size SCIP, offset SCOP, ATTR_SIZE, offset bReturned, 0
test   eax, eax                ;Проверяем успешное выполнение команды
jz     mRSExit                 ;В случае ошибки возвращаем 0.

; Буфер SCOP.bOutBuffer содержит идущие друг за другом значения в
; формате структуры DriveAttribute, начиная со 2-го байта
mov     esi, offset SCOP.bOutBuffer[2]

mov     ecx, 30
mNextAttr:
lodsb                                     ;Загружаем элемент строки [DS:SI] в AL
cmp     al, 194
je      mFoundAttr
cmp     al, 231
je      mFoundAttr
add     esi, size DriveAttribute-1
loop    mNextAttr
mov     eax, -1                 ;Если атрибут не найден, возвращаем -1.
jmp     mRSExit

mFoundAttr:
dec     esi                       ;Корректируем адрес найденной
                                   ;структуры, после команды lodsb
mov     edi, DriveAttr           ;edi = буфер для копирования данных
mov     ecx, size DriveAttribute ;ecx = размер данных
rep     movsb                   ;Копируем данные в буфер.

; Подготавливаем буфер SCIP для чтения пороговых значений S.M.A.R.T.
call   InitSMARTInBuf, Drive, IDE_SMART_READ_THRESHOLDS, offset
SCIP, READ_THRESHOLD_BUFFER_SIZE

```

```

; Читаем пороговые значения
call DeviceIoControl, drvHandle, DFP_RECEIVE_DRIVE_DATA, offset
SCIP, size SCIP, offset SCOP, THRES_SIZE, offset bReturned, 0
test eax, eax ;Проверяем успешное выполнение команды
jz mRSExit ;В случае ошибки возвращаем 0.

```

```

mov esi, offset SCOP.bOutBuffer[2]
mov ecx, 30

```

mNextThres:

```

lodsb
cmp al, 194
je mFoundThres
cmp al, 231
je mFoundThres
add esi, size AttrThreshold-1
loop mNextThres
mov eax, -1
jmp mRSExit

```

mFoundThres:

```

dec esi
mov edi, AttrThres
mov ecx, size AttrThreshold
rep movsb
mov eax, 1 ;Возвращаем 1 (успех).

```

mRSExit: ret

ReadSMARTAttr ENDP

=== Инициализация структуры SendCmdInParams =====

; Drive – номер диска, Feature – номер функции,

; Buffer – указатель на структуры данных, AddSize – размер буфера.

InitSMARTInBuf PROC

ARG Drive:DWORD, Feature:DWORD, Buffer:DWORD, AddSize:DWORD

```

mov eax, AddSize
push eax

```

; Вычисляем размер буфера для чтения параметров

```

add eax, size SendCmdInParams
call RtlZeroMemory, Buffer, eax ;Очищаем буфер.
pop eax ;eax = AddSize
mov SCIP.cBufferSize, eax ;Записываем размер буфера.
mov eax, Feature ;Записываем номер функции.
mov SCIP.irDriveRegs.bFeaturesReg, al
mov SCIP.irDriveRegs.bSectorCountReg, 1
mov SCIP.irDriveRegs.bSectorNumberReg, 1
mov SCIP.irDriveRegs.bCylLowReg, SMART_CYL_LOW

```

```

mov     SCIP.irDriveRegs.bCylHighReg, SMART_CYL_HI
mov     al, byte ptr Drive
mov     SCIP.bDriveNumber, al
;Вычисляем номер диска/головки
and     al, 1
shl     al, 4
or      al, 0A0h
mov     SCIP.irDriveRegs.bDriveHeadReg, al
mov     SCIP.irDriveRegs.bCommandReg, IDE_COMMAND_SMART
ret
InitSMARTInBuf ENDP

END     Start

```

5. Требования к отчёту

Отчет должен содержать:

- титульный лист с указанием названия ВУЗа, кафедры, номера и темы лабораторной работы, а также фамилии и инициалов студента, подготовившего отчёт;
- цель работы;
- вариант задания;
- графический алгоритм программы с кратким описанием;
- полный листинг программы;
- снимок экрана монитора с результатом работы программы;
- выводы.

6. Контрольные вопросы

- 6.1. Что такое S.M.A.R.T. и его атрибуты?
- 6.2. От чего зависит работоспособность жёсткого диска?
- 6.3. По каким критериям можно определить скорый выход из строя жёсткого диска?
- 6.4. Как можно определить сколько раз включался компьютер и сколько часов он проработал?
- 6.5. Как можно получить значения атрибутов?
- 6.6. Назначение функции DeviceIoControl и её формат?
- 6.7. Какова структура буфера ввода данных функции DeviceIoControl?
- 6.8. Какова структура буфера вывода данных функции DeviceIoControl?
- 6.9. Нарисуйте графический алгоритм процедуры InitSMART программы SMART.
- 6.10. Как работать со структурами данных в Ассемблере?

7. Рекомендуемая литература

- 7.1. Маврицин, М. Все, что вы хотели знать о S.M.A.R.T. [Электронный документ] / Михаил Маврицин. – Режим доступа: <http://pcjs.chat.ru/smartdoc.html>. – 9.02.09.
- 7.2. Мюллер, С. Модернизация и ремонт ПК, 19-е издание. [Текст]: Пер. с англ. / С. Мюллер. – М.: ООО «И.Д. Вильямс», 2011. – с.540...542.
- 7.3. CreateFile function (Windows) [Электронный документ]. – Режим доступа: [https://msdn.microsoft.com/en-us/library/windows/desktop/aa363858\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa363858(v=vs.85).aspx), свободный – 26.04.2017.
- 7.4. DeviceIoControl function (Windows) [Электронный документ]. – Режим доступа: [http://msdn.microsoft.com/en-us/library/aa363216\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa363216(VS.85).aspx), свободный – 26.04.2017.
- 7.5. Пирогов, В.Ю. Ассемблер для Windows. [Текст] Изд. 4-е перераб. и доп. / В.Ю. Пирогов. – СПб.: БХВ-Петербург, 2011. – с. 200...207, 344...349.
- 7.6. Аблязов, Р.З. Программирование на ассемблере на платформе x86-64. [Текст] / Р.З. Аблязов. – М.: ДМ К Пресс, 2011. – с. 149...153.
- 7.7. Финогенов, К.Г. Использование языка Ассемблера [Текст]: учеб. пособие для вузов / К.Г. Финогенов. – М: Горячая линия: Телеком, 2004. – с. 67...68.

ЛАБОРАТОРНАЯ РАБОТА N3

Работа с интерфейсом ввода-вывода ПЭВМ

1. Цель работы

Практическое овладение навыками работы с последовательным интерфейсом (COM-портом) ПЭВМ, используя порты ввода-вывода.

2. Рекомендуемая литература

Авдеев В.А. Периферийные устройства: интерфейсы, схемотехника, программирование. – М.: ДМК Пресс, 2009, с.315-323.

Гук М. Аппаратные интерфейсы ПК. Энциклопедия.– СПб.: Питер, 2002, с.49-76, 491-493.

Ан П. Сопряжение ПК с внешними устройствами: Пер. с англ. – М.: ДМК Пресс, 2001, –с.35-41.

Кузьминов А.Ю. Интерфейс RS232. Связь между компьютером и микроконтроллером. – М.: Радио и связь, 2004, с.4-16.

Несвижский В. Программирование аппаратных средств в Windows. – СПб.: БХВ-Петербург, 2004, с.553-572.

Яшкардин В. RS-232. Рекомендованный стандарт 232. [Электронный документ] / Владимир Яшкардин. – Режим доступа: <http://www.softelectro.ru/rs232.html> . – 13.03.12.

3. Подготовка к работе

3.1. Изучить методические указания и рекомендованную литературу.

3.2. Подготовить ответы на контрольные вопросы.

4. Контрольные вопросы

4.1. В чём заключается инициализация COM-порта?

4.2. Приведите несколько способов, с помощью которых можно провести инициализацию COM-порта.

4.3. Как задаётся скорость передачи данных при работе через порты ввода-вывода?

4.4. Как определить базовый адрес порта COM3?

5. Задание на выполнение работы

5.1. Разработать программу выполняющую тестирование последовательного порта в выбранном по варианту режиме. Программа должна включать в себя следующие этапы:

- инициализации универсального асинхронного приёмопередатчика (УАПП, UART);
- ввод данных с клавиатуры и передачу их в УАПП;
- приём этих же данных из УАПП и отображение их экране.

Номер варианта выдаёт преподаватель. Варианты заданий приведены в таблице 1.

Табл. 5.1 – Варианты заданий

№ варианта	Скорость обмена, бод	Число битов данных	Контроль четности/ нечётности	Число стоповых битов
1	110	5	чет.	1
2	300			1,5
3	600			2
4	1 200	6	неч.	1
5	2 400			1,5
6	3 600			2
7	4 800	7	неч.	1
8	9 600			1,5
9	19 200			2
10	38 400	8	нет	1
11	57 600			1,5
12	115 200			2

6. Отчёт должен содержать:

- 6.1 Титульный лист (с названием ВУЗа, кафедры, номера лабораторной работы и её названия, фамилии И.О. студента, подготовившего отчёт)
- 6.2 Цель работы.
- 6.3 Номер варианта и параметры инициализации СОМ-порта.
- 6.4 Графический алгоритм программы с кратким описанием.
- 6.5 Полный листинг программы.
- 6.6 Снимок экрана монитора с результатом работы программы.

7. Общие сведения

В некоторых прикладных задачах программисту приходится работать с низкоскоростными устройствами (различными микроконтроллерами, устройствами ввода-вывода аналоговой и цифровой информации). Как правило такие устройства оснащены универсальными асинхронными приёмо-передатчиками (УАПП) и общение с ними возможно через последовательный порт (Com-порт) компьютера. Далее будем рассматривать работу с УАПП 16550A и совместимые с ним.

Работать с последовательным портом можно тремя способами:

- через программное прерывание INT 14h;
- через порты ввода-вывода;
- используя функции Win32 API.

BIOS представляет программе пользователя набор стандартных функций для работы с последовательным портом (табл. 7.1). Для прерывания **INT 14h** номер функции задаётся через регистр AH, отправляемые в порт данные через AL, а номер порта указывается в DX. Если функция возвращает какие-либо данные, то она это делает через регистры AH и AL. Поскольку для задания скорости обмена во время инициализации отводится только три бита в регистре AH, то устанавливаемая скорость не превышает 9600 бод.

Табл. 7.1 – Основные функции INT 14h

Код в AH	Функция	Входные регистры	Выходные регистры
00h	Инициализация COM-порта	AL – параметры инициализации, DX – номер порта	AH – состояние порта, AL – состояние модема
01h	Передача байта (символа) в порт	AL – символ, DX – номер порта	AH – состояние порта
02h	Приём байта из порта	DX – номер порта	AH – состояние порта, AL – принятый символ
03h	Получение состояния порта	DX – номер порта	AH – состояние порта, AL – состояние модема

Второй способ работы с последовательным портом заключается в использовании адресов портов ввода-вывода, назначенных COM-порту. В процессе начального тестирования POST BIOS проверяет наличие последовательных портов (регистров UART 8250 или совместимых) по стандартным адресам и помещает базовые адреса обнаруженных портов в ячейки **BIOS Data Area** с 0000:0400h по 0000:0407h (табл. 7.2). Нулевое значение адреса является признаком отсутствия порта с данным номером, однако в большинстве случаев эти ячейки заполнены, даже если физически COM-портов нет. Перед работой с последовательным

портом рекомендуется определить его адрес путем чтения соответствующей области BIOS, а не брать стандартное значение, так как возможно назначение альтернативных адресов.

Табл. 7.2 – Адреса переменных BIOS для COM-портов

Имя порта	Адрес в BIOS	Базовый адрес по умолчанию
COM1	0000:0400h – 0000:0401h	3F8h
COM2	0000:0402h – 0000:0403h	2F8h
COM3	0000:0404h – 0000:0405h	3E8h
COM4	0000:0406h – 0000:0407h	2E8h

Каждый УАПП содержит 8 специальных управляющих и контрольных регистров (портов ввода-вывода) выполняющих одну или несколько функций в зависимости от режима работы COM-порта. Эти регистры располагаются один за другим начиная с базового адреса. Назначение регистров COM1 в зависимости от состояния седьмого бита D7 порта 3FBh (бита DLAB – Divisor Latch Access Bit, бит загрузки делителя) приведено в табл. 7.3. Все регистры имеют размер 8 бит.

Табл. 7.3 – Назначение регистров порта COM1

Адрес регистра	Состояние 7-го бита (DLAB) рег. 3FBh	Операция	Название регистра
3F8h	0	Запись	Регистр передатчика
3F8h	0	Чтение	Регистр приёмника
3F8h	1	Запись/чтение	Регистр делителя скорости (младший байт)
3F9h	1	Запись/чтение	Регистр делителя скорости (старший байт)
3F9h	0	Запись/чтение	Регистр разрешения прерывания
3FAh	x	Чтение	Регистр идентификации прерывания
3FAh	x	Запись	Регистр управления буфером FIFO
3FBh	x	Запись/чтение	Регистр управления линией
3FCh	x	Запись/чтение	Регистр управления модемом
3FDh	x	Чтение	Регистр состояния линии
3FEh	x	Чтение	Регистр состояния модема
3FFh	x	Запись/чтение	Рабочий регистр

Если бит загрузки делителя DLAB сброшен, т.е установлен в 0, то регистр 3F8h может быть регистром приёмником или передатчиком в зависимости от выполняемой операции. Регистр передатчика предназначен для временного хранения байта данных, автоматически выводимого на линию TxD, а регистр приёмника – для временного хранения вводимого байта данных с линии RxD. Если бит DLAB равен 1, то регистр 3F8h обрабатывает младший байт делителя частоты для скорости передачи данных, а регистр 3F9h – старшую часть делителя.

Необходимая скорость (V) передачи данных (в бод) задаётся значением делителя K , который определяется по формуле

$$K = \frac{115\,200}{V}.$$

Делитель имеет размер в два байта, поэтому занимает два регистра. Начиная со скорости 600 бод старшая часть делителя равна 0. В табл. 7.4 приведены некоторые значения байтов делителя, определяющие соответствующие скорости передачи данных.

Табл. 7.4 – Коды значений для установки скорости передачи данных

Код значения		Скорость, бод
Порт 3F9h	Порт 3F8h	
04h	17h	110
01h	80h	300
00h	C0h	600
00h	60h	1 200
00h	40h	1 800
00h	30h	2 400
00h	20h	3 600
00h	18h	4 800
00h	10h	7 200
00h	0Ch	9 600
00h	06h	19 200
00h	03h	38 400
00h	02h	57 600
00h	01h	115 200

Процесс работы с COM-портом состоит из нескольких этапов:

- определение наличия нужного порта;
- инициализация последовательного порта;
- передача и приём данных;
- завершение работы с портом.

В приложении А приведён листинг программы, которая передаёт код символа, введённого с клавиатуры, в первый COM-порт и получает эхо сигнал, который выводится на экран.

COM-порт инициализирован на скорость 600 бит/с, 8 бит данных, без контроля чётности, с 1 битом данных.

Данные вводятся с использованием программного прерывания **INT 16h** и выводятся на экран с помощью программного прерывания **INT 10h**. Выход из программы по нажатию клавиши ESC.

Листинг 1 – Программа тестирования COM-порта

Data Segment

```
Text db '"Testirovanie posledovatel'nogo porta"', 13, 10
      db 'Rabotu vipolnil: Nikitin', 13, 10, 13, 10
      db 'Dla vihoda nagmite ESC', 13, 10, '$'
Error db 'Port ne obnaruzhen', 13, 10, '$'
```

Data Ends

Assume cs:code, ds:Data

Code Segment

;-----Очистка экрана и установка курсора в верхний левый угол-----

ClearScreen Proc

```
mov ah, 06h      ;функция прокрутки активной страницы вверх
mov al, 00h      ;количество строк
mov bh, 07h      ;атрибут пробела = Ч/Б, норм. яркости
mov cx, 00h      ;верхняя левая позиция
mov dx, 184Fh    ;нижняя правая позиция (Y=24, X=79)
int 10h
mov ah, 02h
mov bh, 00h
mov dx, 00h
int 10h
ret
```

ClearScreen Endp

;-----Вывод текста на экран-----

Print Proc

```
mov ah, 9
int 21h
ret
```

Print Endp

;-----Инициализация COM1 -----

Initcom1 Proc

```
mov ax, 40h      ;ES указывает на область данных BIOS
mov es, ax
mov dx, es:[0]   ;получаем базовый адрес COM1 (03F8h)
test dx, 0FFFFh  ;адрес порта есть?
jnz m0
lea dx, error
call print
jmp exit
```

```
m0: add dx, 03h      ;регистр контроля (управления) линии (03FBh)
     mov al, 10000000b ;устанавливаем 7 бит
     out dx, al
```

```

dec dx          ;старший байт делителя
dec dx          ;скорости обмена (03F9h)
mov al, 00h     ;старший байт для 600 бод
out dx, al
dec dx          ;(03F8h)
mov al, 0C0h    ;младший байт для 600 бод
out dx, al
add dx, 03h     ;указываем на регистр контроля линии (03FBh)
mov al, 00h     ;очищаем al
or al, 00000011b ;длина символа (11b) — 8 бит данных
or al, 00000000b ;длина стоп-бита (0b) — 1 стоп-бит
or al, 00000000b ;наличие бита чётности (00b) — не генерировать бит
четности
out dx, al
inc dx          ;регистр управления модема (03FCh)
mov al, 00010000b ;диагностический режим (0001XXXXb)
out dx, al
sub dx, 3       ;регистр разрешения прерываний (03F9h)
mov al, 0       ;прерывания запрещены
out dx, al
ret
Initcom1 Endp

```

;-----Преобразование кода символа-----

```

Preobr Proc
mov bx, ax
mov al, 3Dh     ;вывод символа "="
mov ah, 0Eh
int 10h
mov cx, 16
m6:  rol bx,1
     jc m7
     mov al, 30h     ;вывод символа "0"
     jmp m8
m7:  mov al, 31h     ;вывод символа "1"
m8:  int 10h
     loop m6
     ret
Preobr Endp

```

;-----Работа с портом-----

```

Work Proc
m1:  mov ah, 00h     ;ввод символа
     int 16h
     cmp al, 1Bh     ;проверка нажатия ESC
     je m5
     push ax
     push ax
     mov ah, 0Eh     ;вывод этого символа на экран

```

```

int 10h
pop ax
mov ah, 00h
call Preobr
mov al, 1Ah          ;вывод символа "стрелочка"
int 10h
mov dx, es:[0]       ;извлечение адреса COM1 (03F8h)
add dx, 05h          ;регистр статуса линии (03FDh)
mov cx, 0Ah          ;счётчик цикла равен 10
m2: in al, dx         ;забираем из порта 03FDh данные в AL
test al, 20h         ;готов к приему данных? (00100000b)
jz m3
loop m2
m3: sub dx, 05h       ;регистр хранения данных (03F8h)
pop ax              ;занести передаваемый байт
out dx, al          ;передаём в COM-порт код символа из AL
add dx, 05h         ;регистр статуса линии (03FDh)
m4: in al, dx         ;данные приняты ? (читаем порт и передаём в AL)
test al, 01h
jz m4
sub dx, 05h         ;регистр хранения данных (03F8h)
in al, dx           ;прочитать принятые данные
mov ah, 0Eh         ;вывести символ на экран
int 10h
mov ah, 0
call Preobr
mov al, 13          ;перевод в начало строки
int 10h
mov al, 10          ;переход на другую строчку
int 10h
jmp m1
m5: ret
Work Endp

```

```

Start: mov ax, Data;
mov ds, ax;
call ClearScreen    ;очистка экрана
lea dx, Text
call Print          ;вывод строки Text
call Initcom1       ;инициализация COM-порта
call Work           ;работа с портом
exit: mov ax, 4c00h
int 21h

```

Code Ends
End Start

ЛАБОРАТОРНАЯ РАБОТА N4

Анализ трафика между периферийными устройствами и ЭВМ

1. Цель работы

Изучение параметров и технических характеристик периферийных устройств, подключенных к ЭВМ по интерфейсу USB, и анализ трафика между периферийными устройствами и ЭВМ.

2. Рекомендуемая литература

Агуров П.В. Практика программирования USB. – СПб.: БХВ-Петербург, 2006. с. 52...74, 96...113.

Агуров П.В. Интерфейс USB. Практика использования и программирования. – СПб.: БХВ-Петербург, 2004.

Гук М.Ю. Аппаратные средства IBM PC. Энциклопедия. 3-е изд. – СПб.: Питер, 2006. с. 872...899.

3. Подготовка к работе

- 3.1. Изучить методические указания и рекомендованную литературу.
- 3.2. Подготовить ответы на контрольные вопросы.
- 3.3. Подготовить исследуемое периферийное устройство, подключаемое к ЭВМ по интерфейсу USB. Можно использовать манипулятор «мышь», флэш-память, мобильный телефон и т.п. с соответствующим интерфейсом.

4. Контрольные вопросы

- 4.1. Логическая и физическая архитектура USB?
- 4.2. Из каких элементов состоит шина USB?
- 4.3. Какие скорости передачи данных определяет стандарт USB?
- 4.4. Какую информацию содержит стандартный дескриптор USB-устройства?
- 4.5. С помощью чего однозначно определяется модель USB-устройства?
- 4.6. Что называют конечной точкой USB-устройства?
- 4.7. Сколько конечных точек может иметь USB-устройство?
- 4.8. Что является инициатором транзакции USB?
- 4.9. Какие существуют типы дескрипторов устройства и для чего они предназначены?
- 4.10. Какой формат имеет PnP-идентификатор USB-устройств?

5. Задание на выполнение работы

5.1. Подключить к компьютеру устройство с USB-интерфейсом (например, манипулятор «мышь», сотовый телефон, флэш-память и т.п.).

5.2. Запустить программу USB View. В меню «Options» задать просмотр дескрипторов конфигурации.

5.3. Для подключенного устройства просмотреть и проанализировать его дескрипторы.

5.4. Записать в отчет идентификатор изготовителя, идентификатор продукта данного устройства, поддерживаемую версию интерфейса USB, число конфигураций и конечных точек.

5.5. Запустить программу SnoopyPro. Выбрать пункт меню «View» → «USB Devices» (или нажать клавишу F2). Откроется окно USB Devices (см. рис. 1).

5.6. Выбрать пункты «Unpack Drivers» в меню «File» и «Install Service» в том же меню.

5.7. Найти подключенное устройство в списке. Щёлкнуть правой кнопкой мыши на этом устройстве и в появившемся контекстном меню выбрать пункт «Install and Restart». При этом в главном окне программы появится окно журнала.

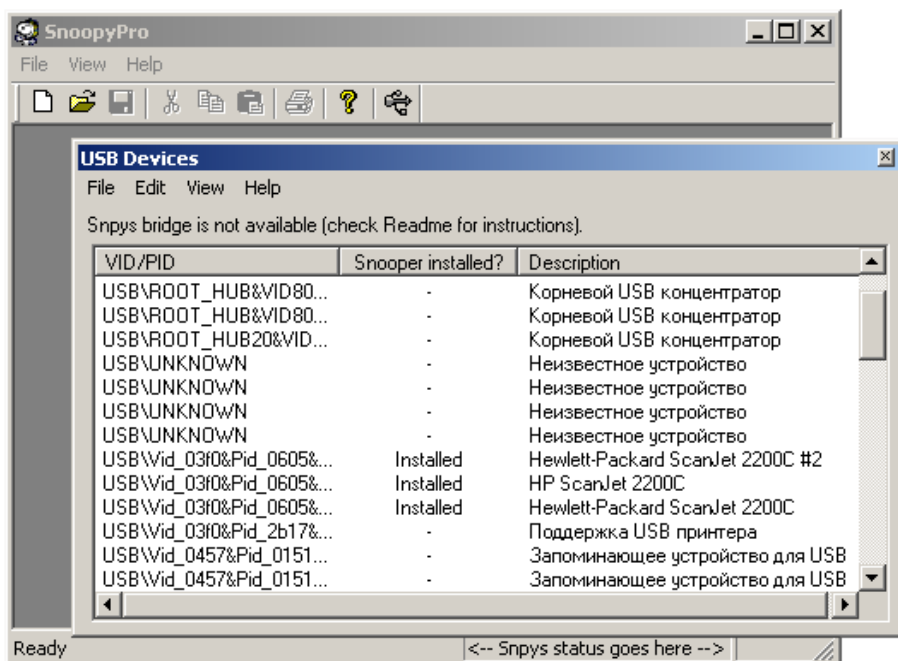


Рис. 1

5.8. Произвести какие-либо действия с устройством, которые привели бы к обмену информацией между устройством и компьютером (например, если это мышь, то следует ей подвигать; если флэш-память – что-нибудь записать на неё). При этом программа производит протоколирование URB-блоков и показывает количество прошедших между компьютером и устройством пакетов в левом верхнем углу окна журнала.

5.9. Остановить протоколирование нажатием на кнопку «Pause» или «Stop». В окне журнала появится список прошедших URB-блоков (см. рис. 2).

5.10. Просмотреть содержимое нескольких URB-блоков. Разобраться в структуре возвращаемых устройством данных. Записать в отчет направление, функцию и первые 10 байт данных из буфера обмена для первых пяти блоков URB.

5.11. Отчитаться о проделанной работе.

6. Отчёт должен содержать:

6.1. Титульный лист (с названием ВУЗа, кафедры, лабораторной работы, а также фамилии И.О. студента, подготовившего отчёт).

6.2. Цель работы.

6.3. Идентификатор изготовителя, идентификатор продукта вашего устройства, поддерживаемую версию интерфейса USB, число конфигураций и конечных точек.

6.4. Направление обмена, функцию и первые 10 байт данных из буфера обмена для пяти первых блоков URB.

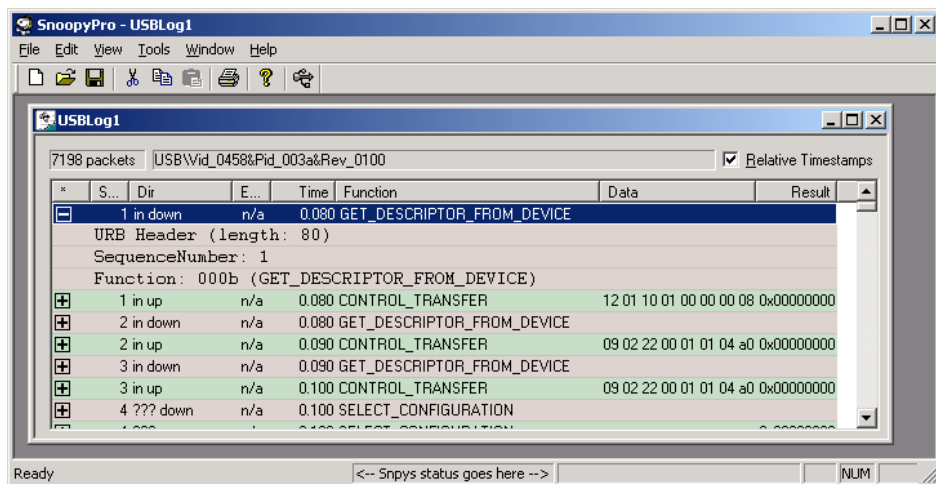


Рис. 2

7. Общие сведения

7.1 Дескрипторы USB устройств

Каждое USB-устройство имеет структуру данных, или форматированный блок информации, называемую **дескриптором**, содержащую информацию о USB-устройстве в целом, или о его части. Существуют следующие виды дескрипторов USB-устройств.

- **Стандартный дескриптор устройства** (*device descriptor*) – содержит основную информацию об USB-устройстве в целом и обо всех существующих конфигурациях. USB-устройство может иметь только один такой дескриптор. HS-устройство, имеющее различную информацию для HS- и FS-режимов, должно иметь также уточняющий дескриптор устройства. Названия полей и их описания для стандартного дескриптора устройства показаны в табл. 7.1.

Табл. 7.1 – Поля стандартного дескриптора устройства

Поле	Описание
bLength	Размер дескриптора в байтах
bDescriptorType	Тип дескриптора (USB_DEVICE_DESCRIPTOR TYPE)
bcdUSB	Номер версии спецификации USB в формате BCD
bDeviceClass	Код класса USB
bDeviceSubClass	Код подкласса USB-устройства
bDeviceProtocol	Код протокола USB
bMaxPacketSize0	Максимальный размер пакета для нулевой конечной точки
idVendor	Идентификатор изготовителя
idProduct	Идентификатор продукта
bcdDevice	Номер версии устройства в формате BCD
iManufacturer	Индекс дескриптора строки, описывающей изготовителя
iProduct	Индекс дескриптора строки, описывающей продукт
iSerialNumber	Индекс дескриптора строки, содержащей серийный номер USB-устройства
bNumConfigurations	Количество возможных конфигураций USB-устройства

- **Уточняющий дескриптор устройства** (*device qualifier descriptor*) – содержит дополнительную информацию о HS-устройстве при его работе на другой скорости. Например, если устройство работает в FS-режиме, то уточняющий дескриптор вернет информацию об HS-режиме работы и наоборот. Названия полей и их описания для уточняющего дескриптора показаны в табл. 7.2.

Табл. 7.2 – Поля уточняющего дескриптора устройства

Поле	Описание
bLength	Размер дескриптора в байтах
bDescriptorType	Тип дескриптора (DEVICE_QUALIFIER)
bcdUSB	Номер версии спецификации USB в формате BCD (равно 0200h или 0101h)
bDeviceClass	Код класса USB
bDeviceSubClass	Код подкласса USB-устройства
bDeviceProtocol	Код протокола USB
bMaxPacketSize0	Максимальный размер пакета для нулевой конечной точки
bNumConfigurations	Количество дополнительных конфигураций устройства
bReserved	Зарезервировано, должно быть равно нулю

• **Дескриптор конфигурации** (*configuration descriptor*) – содержит информацию об одной из возможных конфигураций USB-устройства. Названия полей и их описания для дескриптора конфигурации показаны в табл. 7.3.

Табл. 7.3 – Поля дескриптора конфигурации

Поле	Описание
bLength	Размер дескриптора в байтах
bDescriptorType	Тип дескриптора (USB_CONFIGURATION_DESCRIPTOR_TYPE)
wTotalLength	Общий объем данных (в байтах), возвращаемый для данной конфигурации
bNumInterfaces	Количество интерфейсов, поддерживаемых данной конфигурацией
bConfigurationValue	Идентификатор конфигурации, используемый при вызове SET_CONFIGURATION для установки данной конфигурации
iConfiguration	Индекс дескриптора строки, описывающей данную конфигурацию
bmAttributes	Характеристики конфигурации
MaxPower	Код мощности, потребляемой USB-устройством от шины

• **Дескриптор интерфейса** (*interface descriptor*) – содержит информацию об одном из интерфейсов, доступных при определенной конфигурации USB-устройства. Названия полей и их описания для дескриптора интерфейса показаны в табл. 7.4.

Табл. 7.4 – Поля дескриптора интерфейса

Поле	Описание
bLength	Размер дескриптора в байтах
bDescriptorType	Тип дескриптора (USB_INTERFACE_DESCRIPTOR_TYPE)
bInterfaceNumber	Номер данного интерфейса (нумеруются с 0) в наборе интерфейсов, поддерживаемых в данной конфигурации
bAlternateSetting	Альтернативный номер интерфейса
bNumEndpoints	Число конечных точек для этого интерфейса без учета нулевой конечной точки
bInterfaceClass	Код класса интерфейса
bInterfaceSubclass	Код подкласса интерфейса
bInterfaceProtocol	Код протокола
iInterface	Индекс дескриптора строки, описывающей интерфейс

• **Дескриптор конечной точки** (*endpoint descriptor*) – содержит информацию об одной из конечных точек, доступных при использовании определенного интерфейса. Названия полей и их описания для дескриптора конечной точки показаны в табл. 7.5.

Табл. 7.5 – Поля дескриптора конечной точки

Поле	Описание
bLength	Размер дескриптора в байтах
bDescriptorType	Тип дескриптора (USB_ENDPOINT_DESCRIPTOR_TYPE)
bEndpointAddress	Код адреса конечной точки
bmAttributes	Атрибуты конечной точки
wMaxPacketSize	Максимальный размер пакета для конечной точки
bInterval	Интервал опроса конечной точки при передаче данных (задается в миллисекундах)

• **Дескриптор строки** (*string descriptor*) – содержит текст в формате Unicode. Строка не ограничивается нулем, а длина строки вычисляется вычитанием 2 из размера дескриптора. Названия полей и их описания для дескриптора строки показаны в табл. 7.6.

Табл. 7.6 – Поля дескриптора строки

Поле	Описание
bLength	Размер дескриптора в байтах (N+2)
bDescriptorType	Тип дескриптора (USB_STRING_DESCRIPTOR_TYPE)
bString	Строка символов Unicode

Дескриптор строки является не обязательным. Если USB-устройство не поддерживает дескрипторы строк, все ссылки на такие дескрипторы из

дескрипторов устройства, конфигурации или интерфейса должны иметь нулевое значение.

7.2 Программное обеспечение для мониторинга USB-устройств

Существует множество программ для мониторинга USB-устройств: Advanced USB Port Monitor, Device Monitoring Studio, SnoopyPro, USBDeview, USBlyzer. Для проведения лабораторной работы была выбрана программа SnoopyPro (рис. 3), так как она является удобной и легкой в использовании, позволяет просматривать и протоколировать содержимое блоков URB, передаваемых между компьютером и USB-устройством. Программа разработана в рамках проектов с открытым исходным кодом и является бесплатной.

Так как распознавание USB-устройств в SnoopyPro производится по идентификаторам производителя и продукта, то сначала необходимо их определить. Для этого можно использовать программу USB View (см. рис. 4).

Программа USB View отображает древовидную структуру подключенных к компьютеру USB-устройств и содержимое их дескрипторов. Поля idVendor и idProduct дескриптора устройства содержат шестнадцатеричные идентификаторы производителя и продукта, по которым можно однозначно определить USB-устройство.

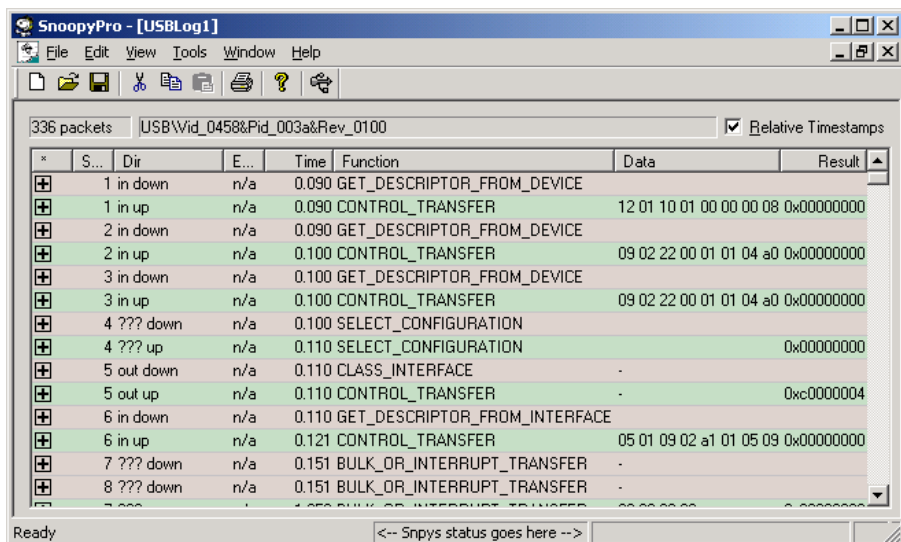


Рис. 3

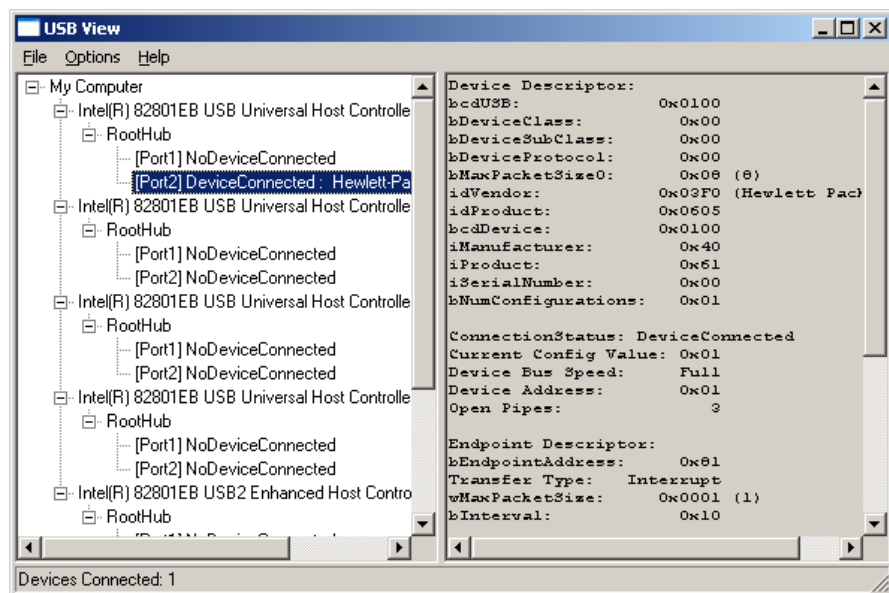


Рис. 4

ЛАБОРАТОРНАЯ РАБОТА N7

Программирование внешних устройств. Клавиатура

1. Цель работы

Изучить устройство клавиатуры, интерфейс связи с ПЭВМ и схемы связи контроллера клавиатуры и манипулятора «мышь».

2. Рекомендуемая литература

Агуров П.В. Практика программирования USB. – СПб.: БХВ-Петербург, 2006.

Комиссарова В. Программирование драйверов для Windows. – СПб.: БХВ-Петербург, 2007.

Они У. Использование Microsoft Windows Driver Model. 2-е изд. (+CD). Для профессионалов. – СПб.: Питер, 2007.

Солдатов В.П. Программирование драйверов Windows. Изд. 3-е, перераб. и доп. – М.: ООО «Бином-Пресс», 2006.

3. Подготовка к работе

- 3.1. Изучить методические указания и рекомендованную литературу.
- 3.2. Подготовить ответы на контрольные вопросы.

4. Контрольные вопросы

- 4.1. Что называется ASCII – кодом и на какие группы делится таблица символов ASCII – кодов?
- 4.2. Что называется скан-кодом? Перечислите основные виды таблиц скан-кодов и их предназначение.
- 4.3. Как образуются прерывания от манипулятора «мышь»?
- 4.4. Опишите принцип обмена данных между клавиатурой и конечным пользователем.
- 4.5. Какое назначение имеет регистр состояния контроллера клавиатуры, расположенного на системной плате?
- 4.6. Какое назначение имеет регистр команд контроллера клавиатуры, расположенного на системной плате?
- 4.7. Какое назначение имеет регистр данных контроллера клавиатуры, расположенного на системной плате?
- 4.8. Как происходит тестирование клавиатуры при включении компьютера?

5. Задание на выполнение работы

5.1. Разобрать клавиатуру. Изучить устройство и принцип работы. Составить список функциональных частей клавиатуры.

5.2. Подключить клавиатуру к системному блоку. Описать принцип ее подключения и определить месторасположения контроллера i8042.

5.3. Получить задание от преподавателя и определить последовательность символов, введенных преподавателем (по скан-коду).

5.4. Записать последовательность ASCII – кодов преобразованных символов из задания №3.

5.5. Составить алгоритм обработки сигнала контроллерами клавиатуры.

6. Отчёт должен содержать:

6.1 Титульный лист (с названием ВУЗа, кафедры, номера лабораторной работы и её названия, фамилии И.О. студента, подготовившего отчёт)

6.2 Цель работы.

6.3 Задание и результат его выполнения.

7. Общие сведения

7.1 Работа с клавиатурой

Клавиатура является основным средством ввода текстовой информации в компьютер, поэтому при программировании на низком уровне (на языке ассемблера) программист в первую очередь вынужден осваивать работу с клавиатурой: он должен изучить способы кодирования текстовых символов и особенности использования функций операционной системы. Если у программиста возникает потребность в работе на уровне аппаратного обеспечения, он должен также разобраться с особенностями программирования контроллера клавиатуры и контроллера прерываний.

7.2 Представление символов и управляющих кодов в памяти компьютера

Система представления символов в персональных компьютерах базируется на Американском стандартном коде для обмена информацией (American Standard Code for Information Interchange), который был введен в 1963 году и ставил в соответствие каждому символу семиразрядный двоичный код, обеспечивающий представление 128 символов. ASCII-код включал две группы символов:

- управляющие символы, используемые в коммуникационных протоколах для передачи команд периферийным устройствам;
- символы пишущей машинки — цифры, буквы и специальные знаки.

Управляющие символы имеют коды с номерами от 0 до 1Ah. К управляющим относится также символ с кодом 7Fh. Каждый управляющий символ выполняет строго определенную функцию. Функции и кодовые обозначения управляющих символов описаны в табл. 1.1. Все остальные символы относятся к алфавитно-цифровой группе (группе символов пишущей машинки).

Таблица 1.1. Управляющие символы ASCII-кода

Код символа	Мнемоническое обозначение	Назначение символа
00h	NUL	Пустой символ
01h	SOH	Начало заголовка (начало блока данных)
02h	STX	Начало текста
03h	ETX	Конец текста
04h	EOT	Конец передачи
05h	ENQ	Запрос подтверждения
06h	ACK	Подтверждение
07h	BEL	Звонок (звуковой сигнал)
08h	BS	Забой (возврат на одну позицию влево)
09h	HT	Горизонтальная табуляция
0Ah	LF	Перевод строки
0Bh	VT	Вертикальная табуляция
0Ch	FF	Перевод формата (переход к новой странице)
0Dh	CR	Возврат каретки
0Eh	SO	Переход на нижний регистр
0Fh	SI	Переход на верхний регистр
10h	DLE	Завершение сеанса связи
11h	DC1	Управление устройством № 1
12h	DC2	Управление устройством № 2
13h	DC3	Управление устройством № 3
14h	DC4	Управление устройством № 4
15h	NAK	Ошибка передачи
16h	SYN	Холостой ход передатчика
17h	ETB	Конец передачи блока
18h	CAN	Отмена
19h	EM	Конец носителя данных
1Ah	SUB	Подстановка (замена символа)

продолжение таблицы 1.1

Код символа	Мнемоническое обозначение	Назначение символа
1Bh	ESC	Переход (посылка сложной команды)
1Ch	FS	Разделитель файлов
1Dh	GS	Разделитель групп
1Eh	RS	Разделитель записей
1Fh	US	Разделитель элементов
7Fh	DEL	Удаление символа

Чтобы отобразить символы европейских алфавитов и символы псевдографики, ASCII-код был расширен до 256 символов. Это так называемая американская кодировка (кодировка IBM), которая в операционных системах корпорации Microsoft носит также название «Кодовая страница 437».

Однако по мере распространения персональных компьютеров постоянно возникала потребность в добавлении изображений новых символов, поэтому каждая страна мира сейчас имеет свою собственную кодовую страницу, а в многоязычных странах обычно применяется несколько таких страниц. Графическое представление символов расширенного ASCII-кода показано на рис. 1.1.



Младшая цифра	Старшая цифра															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		►		0	@	P	`	p	A	P	a				р	Ё
1	☺	◄	!	1	A	Q	a	q	Б	С	б				с	ё
2	☹	↕	«	2	B	R	b	r	В	Т	в				т	€
3	♥	!!	#	3	C	S	c	s	Г	У	г				у	€
4	♦	¶	\$	4	D	T	d	t	Д	Ф	д				ф	Ї
5	♣	§	%	5	E	U	e	u	Е	Х	е				х	ї
6	♠	—	&	6	F	V	f	v	Ж	Ц	ж				ц	Ў
7	•	↕	'	7	G	W	g	w	З	Ч	з				ч	Ў
8	■	↑	(8	H	X	h	x	И	Ш	и				ш	°
9	○	↓)	9	I	Y	i	y	Й	Щ	й				щ	°
A	■	—	*	:	J	Z	j	z	К	Ъ	к				ъ	°
B	♂	←	+	;	K	[k	{	Л	Ы	л				ы	√
C	♀	└	,	<	L	\	l		М	Ь	м				ь	№
D	♪	←	-	=	M]	m	}	Н	Э	н				э	□
E	♪	▲	.	>	N	^	n	~	О	Ю	о				ю	■
F	☼	▼	/	?	O		o	△	П	Я	п				я	

Рис. 1.1. Представление символов ASCII-кода в русской кодовой странице 866

Представление символов ASCII-кода в русской кодовой таблице MS-DOS (кодовая страница 866) показано на рис. 1.1. Как видно из рисунка, символов в таблице гораздо больше, чем клавиш в алфавитно-цифровой части типовой клавиатуры (101-клавишный AT-совместимый вариант исполнения), поэтому каждой клавише поставлено в соответствие несколько различных символов. ASCII-код, генерируемый при нажатии клавиши, определяется не только этой клавишей, но и состоянием управляющих клавиш Caps Lock и Shift, а также режимом работы драйвера клавиатуры, то есть текущим языком — русским или английским.

Кроме ASCII-кодов, для идентификации клавиш используются также скан-коды. Скан-коды в старых клавиатурах (появившихся до использования микроконтроллеров) являлись порядковыми номерами клавиш: нумерация велась сверху вниз, справа налево. С целью сохранения совместимости со старым программным обеспечением микропроцессоры современных клавиатур преобразуют действительные порядковые номера клавиш в номера, соответствующие латинской раскладке на клавиатуре IBM XT с учетом дополнительных клавиш клавиатуры IBM AT. В результате распределение номеров перестало быть строго упорядоченным. Кроме того, функции BIOS также выполняют перекодировку скан-кодов с целью упрощения анализа этих кодов в прикладных программах. Поэтому существует несколько различных видов таблиц скан-кодов:

- собственная внутренняя таблица встроенного микроконтроллера клавиатуры;
- таблица для обмена кодами между контроллером клавиатуры и специализированным клавиатурным микропроцессором системной платы;
- таблица кодов, которые клавиатурный микропроцессор передает подпрограммам BIOS;

При работе с функциями BIOS интерес представляет последняя из этих таблиц. Скан-коды BIOS для клавиш алфавитно-цифровой группы и скан-коды клавиш функциональной, дополнительной и цифровой групп приведены в таблице 1.2. Следует учитывать, что клавиши цифровой группы, расположенной с правой стороны клавиатуры, могут использоваться не только как свободные (цифровые), но и как управляющие — в зависимости от состояния клавиши Num Lock.

7.3 Ввод информации с клавиатуры при помощи функций BIOS

Для ввода информации с клавиатуры можно использовать либо функции операционной системы, либо прямой опрос контроллера клавиатуры. Функции MS-DOS, используемые для ввода данных с клавиатуры непригодны для сколько-нибудь серьезной работы. Функции DOS имеют два очень серьезных недостатка. Первый недостаток заключается в том, что они не позволяют полностью реализовать

возможности функциональных клавиш. Второй недостаток — клавиатурные функции DOS предназначены для работы в режиме терминала. В процессе считывания символа они выполняют ряд дополнительных операций, что делает весьма неудобным их использование в любом другом, не терминальном режиме.

Таблица 1.2. Скан-коды BIOS для клавиш алфавитно-цифровой группы

Скан-код Русский	Режим		Скан-код Латинский	Режим	
	Латинский	Русский		Латинский	Русский
01h	Esc		1Eh	A и a	Ф и ф
02h	1 и !	1 и !	1Fh	S и s	Ы и ы
03h	2 и @	2 и «	20h	D и d	В и в
04h	3 и #	3 и №	21h	F и f	А и а
05h	4 и \$	4 и ;	22h	G и g	П и п
06h	5 и %	5 и %	23h	H и h	Р и р
07h	6 и ^	6 и :	24h	J и j	О и о
08h	7 и &	7 и ?	25h	K и k	Л и л
09h	8 и *	8 и *	26h	L и l	Д и д
0Ah	9 и (9 и (27h	; и :	Ж и ж
0Bh	0 и)	0 и)	28h	“ и ’	Э и э
0Ch	- и _	- и _	29h	` и ~	Ё и ё
0Dh	= и +	= и +	2Ah	Левая клавиша Shift	
0Eh	Back Space		2Bh	\ и	\ и /
0Fh	Tab		2Ch	Z и z	Я и я
10h	Q и q	Й и й	2Dh	X и x	Ч и ч
11h	W и w	Ц и ц	2Eh	C и c	С и с
12h	E и e	У и у	2Fh	V и v	М и м
13h	R и r	К и к	30h	B и b	И и и
14h	T и t	Е и е	31h	N и n	Т и т
15h	Y и y	Н и н	32h	M и m	Ь и ь
16h	U и u	Г и г	33h	, и <	Б и б
17h	I и i	Ш и ш	34h	. и >	Ю и ю
18h	O и o	Щ и щ	35h	/ и ?	. и ,
19h	P и p	З и з	36h	Правая клавиша Shift	
1Ah	[и {	Х и х	37h	*	
1Bh] и }	Ъ и ъ	38h	Alt	
1Ch	Enter		39h	Пробел	
1Dh	Ctrl		3Ah	Caps Lock	

Функции BIOS обладают гораздо более широкими возможностями, чем функции DOS. Этих возможностей вполне достаточно для

выполнения любых операций реальном режиме работы процессора. Вызов клавиатурных функций BIOS выполняется по прерыванию Int 16h.

7.4 Контроллер прерываний

Устройства ввода информации сообщают центральному процессору о поступлении новых данных с помощью сигналов прерываний. Прерывания от клавиатуры, мыши PS/2-типа и других периферийных устройств, прежде чем поступить в процессор, проходят через контроллер прерываний, где подвергаются предварительной обработке. Контроллер позволяет управлять приоритетами, прохождением сигналов и адресами векторов прерываний.

Контроллер прерываний IBM AT состоит из двух микросхем Intel 8259, включенных в режиме каскадирования (рис. 1.2). Первая микросхема была ведущей, а вторая — ведомой (ведомый контроллер подключен к входу IRQ2 ведущего). На входы IRQ0, IRQ1, IRQ3-IRQ7 ведущей микросхемы и на входы IRQ8-IRQ15 ведомой поступают запросы прерываний, из которых выбирается немаскированный запрос с наивысшим приоритетом, после чего контроллер вырабатывает сигнал **INT** и передает в процессор вектор прерывания.

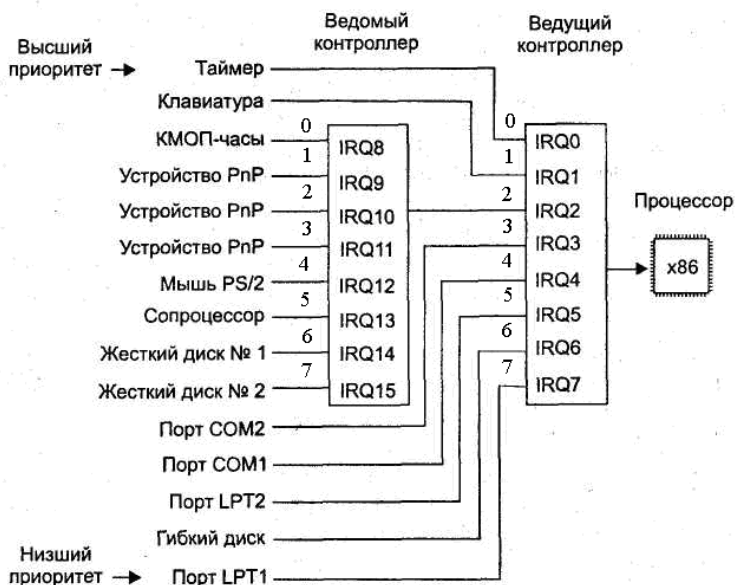


Рис. 1.2. Традиционный порядок подключения внешних устройств к контроллеру прерываний

Вектор формируется путем сложения базового значения (записанного в соответствующий регистр микросхемы) и номера линии, на которую поступил запрос (ведущей микросхеме IRQ0 соответствует линия 0, IRQ7 — линия номер 7; ведомой микросхеме IRQ8 соответствует линия 0, IRQ15 — линия 7). Базовый вектор ведущей микросхемы в реальном режиме DOS имеет значение 08h; вектор ведомой — 70h. Соответственно, ведущая схема вырабатывает вектора с номерами 08h-0Fh, ведомая — с номерами 70h-77h (Табл. 1.3). Приоритеты запросов прерывания (по убыванию) располагаются в следующем порядке: IRQ0, IRQ1, IRQ8-IRQ15, IRQ3-IRQ7.

Все последующие модели АТ-совместимых компьютеров вынуждены имитировать работу микросхем i8259 с целью сохранения совместимости со старым программным обеспечением. Большая часть возможностей указанных микросхем, к счастью для программистов, не используется в АТ-совместимых персональных компьютерах: приоритеты прерываний, поступающих от периферийных устройств, и адреса соответствующих векторов жестко зафиксированы (устанавливаются BIOS в процессе начальной загрузки).

Таблица 1.3. Аппаратные прерывания АТ – совместимых компьютеров

Прерывание	Номер вектора	Адрес вектора	Источник сигнала прерывания
IRQ0	08h	0000:0020h	Системный таймер
IRQ1	09h	0000:0024h	Клавиатура
IRQ2	0Ah	0000:0028h	Ведомая микросхема контроллера
IRQ3	0Bh	0000:002Ch	Последовательный порт COM2
IRQ4	0Ch	0000:0030h	Последовательный порт COM1
IRQ5	0Dh	0000:0034h	Параллельный порт LPT2
IRQ6	0Eh	0000:0038h	Контроллер дисководов гибких дисков
IRQ7	0Fh	0000:003Ch	Параллельный порт LPT1
IRQ8	70h	0000:01C0h	Часы реального времени
IRQ9	71h	0000:01C4h	Любое устройство PnP
IRQ10	72h	0000:01C8h	Любое устройство PnP
IRQ11	73h	0000:01CCh	Любое устройство PnP
IRQ12	74h	0000:01D0h	Мышь PS/2 – типа
IRQ13	75h	0000:01D4h	Математический сопроцессор
IRQ14	76h	0000:01D8h	Контроллер жесткого диска №1
IRQ15	77h	0000:01DCh	Контроллер жесткого диска №2

7.5 Непосредственная работа с контроллером клавиатуры

Реальная необходимость в непосредственной работе с клавиатурой возникает в том случае, когда создается программа, которая переводит процессор из реального режима в защищенный, а затем выполняет в защищенном режиме всю дальнейшую работу. Переход в защищенный режим приводит к тому, что функции BIOS, рассчитанные на реальный режим, становятся непригодными для использования.

Для управления работой клавиатуры в машинах типа IBM AT и PS/2 использовался микроконтроллер Intel 8042. Кроме клавиатуры, этот контроллер управлял также координатным устройством, в качестве которого обычно использовалась мышь типа PS/2. С целью обеспечения совместимости нового аппаратного обеспечения со старыми программами все современные наборы микросхем, предназначенные для изготовления системных плат (так называемые чипсеты), вынужденно повторяют в своей структуре особенности контроллера.

Писать программы для клавиатурного контроллера i8042 и встроенного микропроцессора клавиатуры нет необходимости и возможности, поскольку соответствующие программы уже записаны в ПЗУ контроллеров. Поэтому клавиатурный контроллер на системной плате и микропроцессор клавиатуры могут выполнять только те операции, которые заложены в них разработчиками аппаратуры и внесены в соответствующие наборы команд. Самопроверка контроллера и программирование основных параметров его работы, а также самотестирование и программирование параметров клавиатуры (и мыши PS/2) производятся при включении компьютера и могут быть изменены с помощью прерываний BIOS, пока процессор еще не переведен из реального режима в защищенный. В защищенном режиме необходимо выполнять только два типа операций: считывание кодов нажимаемых клавиш и зажигание/гашение светодиодов на клавиатуре.

Управление работой контроллера и обмен данными с мышью и клавиатурой осуществляются при помощи трех регистров: регистра состояния, регистра команд и регистра данных. Кроме того, при поступлении информации от клавиатуры контроллер i8042 вырабатывает прерывание IRQ1, а при приеме данных от мыши — IRQ12. Интерфейсы клавиатуры и мыши аналогичны, наборы команд управления также имеют некоторое сходство. Упрощенная схема подключения к компьютеру клавиатуры и мыши типа PS/2 показана на рис. 1.3.

Регистр состояния доступен только для считывания через порт 64h. Формат регистра показан на рис. 1.4. Значение разрядов регистра следующее:

- бит 0 — признак наличия данных в выходном буфере (0 — буфер пуст, 1 — буфер заполнен);

- бит 1 — признак наличия данных во входном буфере (0 — буфер пуст, 1 — буфер заполнен);
- бит 2 — признак типа последнего общесистемного сброса (0 — сброс по включении питания, 1 — программный сброс);
- бит 3 — признак записи команды (0 — последняя операция записи являлась операцией записи данных, 1 — записи команды);
- бит 4 — состояние «замка» клавиатуры (0 — клавиатура заблокирована, 1 — не заблокирована);
- бит 5 — признак ошибки тайм-аута передачи в АТ-режиме (0 — нормальное завершение передачи, 1 — произошла ошибка); признак наличия данных в выходном буфере мыши в PS/2-режиме (0 — буфер пуст, 1 — буфер заполнен);
- бит 6 — признак ошибки тайм-аута приема в АТ-режиме (0 — нормальное завершение приема, 1 — произошла ошибка); общий признак ошибки тайм-аута при приеме или передаче данных в PS/2-режиме (0 — нормальное завершение операции, 1 — произошла ошибка);
- бит 7 — признак возникновения ошибки паритета при приеме или передаче данных (0 — нет ошибки, 1 — обнаружена ошибка по четности).

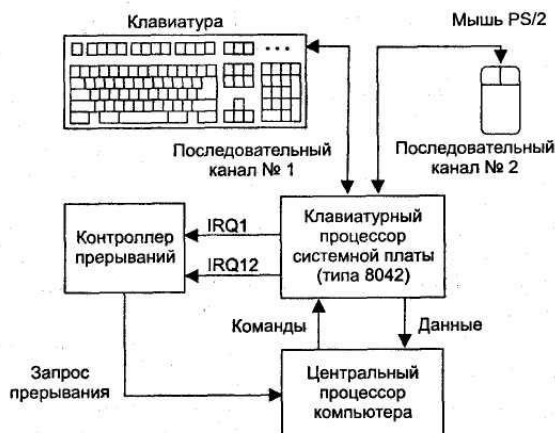


Рис. 1.3. Упрощенная схема подключения клавиатуры и мыши типа PS/2 к компьютеру

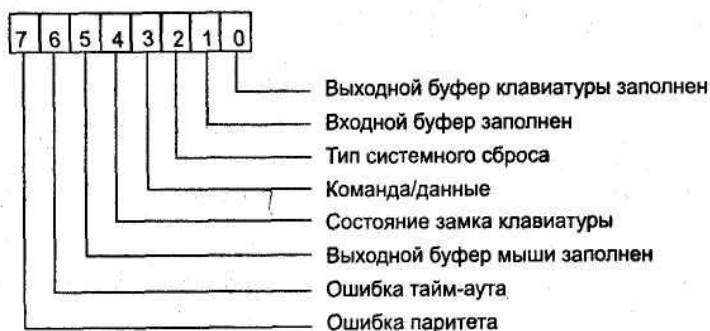


Рис. 1.4. Формат регистра состояния контроллера клавиатуры

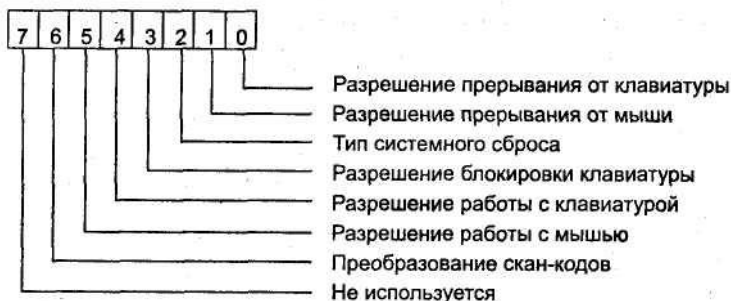


Рис. 1.5. Формат регистра команд контроллера клавиатуры

Регистр команд доступен для записи через порт 64h. Формат регистра показан на рис. 1.5. Разряды регистра имеют следующее назначение:

- бит 0 — управление выдачей сигнала прерывания по готовности данных в выходном буфере клавиатуры (0 — генерация прерывания запрещена, 1 — разрешена);
- бит 1 — управление выдачей сигнала прерывания по готовности данных в выходном буфере мыши (0 — генерация прерывания запрещена, 1 — разрешена);
- бит 2 — установка признака системного сброса (значение, записанное в этот разряд, переносится в разряд 2 регистра состояния);
- бит 3 — управление функцией блокировки клавиатуры (0 — функция блокировки разрешена, 1 — функция запрещена и блокировка игнорируется);
- бит 4 — управление интерфейсом клавиатуры (0 — обмен данными с клавиатурой разрешен, 1 — запрещен);

- бит 5 — признак типа протокола передачи данных в AT-режиме (0 — протокол IBM); управление интерфейсом мыши в PS/2-режиме (0 — обмен данными с мышью разрешен, 1 — запрещен);
- бит 6 — преобразование скан-кодов в PC-совместимые (0 — выключено, 1 — включено); по умолчанию имеет значение 1, то есть скан-коды преобразуются в PC-совместимые;
- бит 7 — не используется (зарезервирован).

Регистр данных доступен для записи и считывания через порт 60h. В режиме считывания он служит для приема информации от клавиатуры и мыши. В режиме записи регистр данных служит для передачи команд клавиатуре, координатному устройству (мышь PS/2-типа) и клавиатурному контроллеру системной платы i8042. Какого-либо особого формата данный регистр не имеет.

Клавиатура может работать в трех различных режимах, которые отличаются друг от друга наборами скан-кодов, выдаваемых клавиатурой. Набор № 1 предназначен для компьютеров типа IBM «PS/2 30», набор №2 - для «PC/AT», «PS/2 50» и «PS/2 60», набор № 3 — для «PS/2 80». Для переключения режимов работы используется команда F0h. Однако программисты никогда не сталкиваются непосредственно с наборами кодов, выдаваемыми клавиатурой, а имеют дело с преобразованным набором, выдаваемым клавиатурным процессором системной платы.

При включении питания клавиатура устанавливается в режим № 2, причем для всех клавиш разрешена посылка кодов нажатия и отпускания, а также разрешен автоповтор. Самотестирование после включения выполняется в следующем порядке:

- включаются все индикаторы;
- тестируется встроенный микроконтроллер клавиатуры;
- тестируется оперативная память клавиатуры;
- все индикаторы выключаются;
- клавиатура выдает компьютеру результат самотестирования.

Когда разрешен опрос клавиш, клавиатура передает компьютеру скан-коды, сообщающие об изменении их состояния (под изменением состояния подразумевается нажатие, длительное удержание или отпускание). Кроме того, возможна посылка следующих специализированных кодов в ответ на команду или при возникновении неисправностей.

Режим работы клавиатуры устанавливается при запуске компьютера процедурами BIOS, и после этого изменять его обычно нет необходимости. Единственная команда, которую драйвер периодически посылает клавиатуре, — команда переключения светодиодов EDh. Данная команда является ответной реакцией драйвера на нажатие клавиш Num Lock, Caps Lock и Scroll Lock.

Как уже отмечалось выше, набор скан-кодов, которые выдает клавиатурный процессор системной платы, отличается и от набора скан-

кодов, которые используют процедуры BIOS. На рис. 1.6 показаны коды, присвоенные клавишам основной и функциональной групп, на рис. 1.7 — коды дополнительной клавиатуры, а на рис. 1.8 — коды цифровой клавиатуры.

01	3B	3C	3D	3E	3F	40	41	42	43	44	45	46			
29	02	03	04	05	06	07	08	09	0A	0B	0C	0D	2B	0E	
0F	10	11	12	13	14	15	16	17	18	19	1A	1B			
3A	1E	1F	20	21	22	23	24	25	26	27	28		1C		
2A	2C	2D	2E	2F	30	31	32	33	34	35		36			
1D		38	39								E0,38			E0,1D	

Рис. 1.6. Скан-коды клавиш основной и функциональной клавиатур

E0,2A, E0,37	46	E1,1D,45, E1,9D,C5
E0,2A, E0,52	E0,2A, E0,47	E0,2A, E0,49
E0,2A, E0,53	E0,2A, E0,4F	E0,2A, E0,51
	E0,2A, E0,48	
E0,2A, E0,4B	E0,2A, E0,50	E0,2A, E0,4D

Рис. 1.7. Скан-коды клавиш дополнительной клавиатуры

45	E0,35	37	4A
47	48	49	4E
4B	4C	4D	
4F	50	51	E0,1C
52	53		

Рис. 1.8. Скан-коды клавиш цифровой клавиатуры

Нажатие клавиши приводит к передаче от клавиатуры к компьютеру одного символа или последовательности символов (от двух до шести). При нажатии обычных клавиш (алфавитно-цифровых или

функциональных) передается только один байт, содержащий скан-код. Последовательности генерируются для клавиш, которые отсутствовали в 84-кнопочной клавиатуре XT-типа, и состоят из кодовых пар, причем каждая пара начинается с кода E0h, а во втором байте передается скан-код. Последовательность из четырех байт (двух пар) передается в том случае, если нажата дополнительная клавиша, заменяющая собой нажатие определенной последовательности обычных клавиш. Специфическая последовательность из шести байт генерируется только в одном случае — при нажатии клавиши Pause.

При отпускании клавиши клавиатура также посылает в компьютер скан-код, но старший (знаковый) разряд кода при этом устанавливается в единицу. Отпускание клавиши, выдающей пару кодов, можно отличить от нажатия по второму символу пары (первым кодом в паре по прежнему является E0h, а у скан-кода при отпускании будет установлен старший разряд). При отпускании дополнительных клавиш генерируются две пары кодов, но порядок этих пар является обратным тому, который генерируется при их нажатии, и установлены старшие разряды скан-кодов. При отпускании клавиши Pause клавиатура никакой информации в компьютер не передает.

Например, при нажатии клавиши Пробел вырабатывается код 39h, а при отпускании — код B9h. При нажатии клавиши 1 на основной клавиатуре вырабатывается код 02h, а при отпускании — код 82h; Нажатие клавиши 1 на цифровой клавиатуре порождает код 4Fh, отпускание — код CFh. При нажатии правой клавиши Ctrl вырабатывается последовательность E0h, 1Dh, а при отпускании — последовательность E0h, 9Dh. Нажатие Insert порождает последовательность кодов E0h, 2Ah, E0h, 52h, а отпускание — последовательность E0h, D2h, E0h, AAh. Нажатие клавиши Pause приводит к выдаче последовательности E1h, 1Dh, 45h, E1h, 9Dh, C5h, а при отпускании данной клавиши никаких кодов не вырабатывается вообще.

Для поддержки расширенного интерфейса управления конфигурацией и питанием (Advanced Configuration and Power Interface, сокращенно ACPI) на клавиатуру были до-бавлены три клавиши:

- Power (Выключить питание) — вырабатывает последовательность E0h, 5Eh;
- Sleep (Переключить систему в спящий режим) — вырабатывает последовательность E0h, 5Fh;
- Wake (Разбудить систему) — вырабатывает последовательность E0h, 63h.

ЛАБОРАТОРНАЯ РАБОТА N4

Определение основных характеристик видеоадаптера

1. Цель работы

Изучить порядок применения функций видео-BIOS для определения основных характеристик и состояния SVGA-видеоадаптера.

2. Рекомендуемая литература

Агуров П.В. Практика программирования USB. – СПб.: БХВ-Петербург, 2006.

Комиссарова В. Программирование драйверов для Windows. – СПб.: БХВ-Петербург, 2007.

Они У. Использование Microsoft Windows Driver Model. 2-е изд. (+CD). Для профессионалов. – СПб.: Питер, 2007.

Солдатов В.П. Программирование драйверов Windows. Изд. 3-е, перераб. и доп. – М.: ООО «Бином-Пресс», 2006.

3. Подготовка к работе

- 3.1. Изучить методические указания и рекомендованную литературу.
- 3.2. Подготовить ответы на контрольные вопросы.

4. Контрольные вопросы

- 4.1. Чем отличаются текстовый и графический режимы экрана?
- 4.2. Как получить список видеорежимов?
- 4.3. Какие характеристики видеорежимов знаете?
- 4.4. Как определить производителя ВА?
- 4.5. Из каких узлов состоит видеоадаптер?

5. Задание на выполнение работы

5.1. Разработать программу, которая должна выполнять следующие обязательные действия:

а) определение названия фирмы-разработчика и номера версии видео-BIOS;

б) определение следующих основных характеристик установленного видеоадаптера:

- номер версии VBE;

- список видеорежимов, поддерживаемых графическим контроллером и цифроаналоговыми преобразователями видеоадаптера в шестнадцатеричном формате;

- объем памяти видеоадаптера в килобайтах;
- дополнительный номер версии VBE;
- названия фирмы-разработчика;
- название видеоадаптера;
- дополнительный номер версии видеоадаптера;
- информация, определяемая фирмой-изготовителем;
- список видеорежимов поддерживаемых видеоадаптером в шестнадцатеричном формате;

в) определение и отображение номера текущего видеорежима в шестнадцатеричном формате;

г) получение информация о произвольном видеорежиме, введенном в шестнадцатеричном формате.

д) отображение на экране и запись в файл полученных сведений.

6. Отчёт должен содержать:

6.1 Титульный лист (с названием ВУЗа, кафедры, номера лабораторной работы и её названия, фамилии И.О. студента, подготовившего отчёт)

6.2 Цель работы.

6.3 Задание и результат его выполнения.

6.4 Состав и назначение переменных и процедур программы.

6.5 Исходный текст анализирующей части программы.

6.6 Выводы по результатам, полученным в ходе выполнения работы.

7. Общие сведения

7.1 Краткое описание видеоадаптера

Видеоадаптер (ВА) – периферийное устройство, преобразующее коды графических и текстовых изображений в электрические сигналы изображения, подаваемые на устройство отображения (видеомонитор, ВМ). ВА в архитектуре ПЭВМ, основанных на центральном процессоре (ЦП) Intel 80x86, содержит следующие функциональные узлы:

- графический контроллер (ГК);
- ПЗУ видеоBIOS (ROM videoBIOS);
- микросхемы динамического ОЗУ (videoRAM), которые образуют т.н. видеопамять (ВП) в адресном пространстве системной памяти.

К числу основных графических характеристик ВА принадлежат следующие параметры:

- число элементов изображения по горизонтали и вертикали (разрешение);

- число цветов (оттенков) элементов изображения.

Совокупность данных параметров получила название видеорежим и определяет качество изображения.

Разрешение часто обозначается как $N \times M$, где N – число элементов по горизонтали, а M – по вертикали. Видеорежимы делятся на графические и текстовые. Возможность ВА работать в тех или иных видеорежимах зависит от объема ВП, производительности ГК, количества бит, отводимых на кодирование цвета точек (пикселей), разрядности и быстродействия цифроаналоговых преобразователей (ЦАП).

Выполняя текущую программу, ЦП по мере необходимости формирует массив кодов видеоданных, размещаемых в ВП. В графическом режиме видеоданные представляют собой упорядоченно размещенную в ВП совокупность кодов цветов каждой точки текущего изображения.

В текстовом режиме каждая пара байтов видеоданных кодирует один символ, его цвет, цвет фона. ГК преобразует коды символов в коды пикселей без записи последних в ВП с помощью знакогенератора – области данных, содержащих точечное описание символов. Все символы занимают на экране область фиксированного размера – символьную матрицу пикселей. Часть пикселей матрицы образует изображение символов, остальные – цвет фона.

ГК с постоянной частотой циклически считывает видеоданные из ВП и с помощью трех ЦАП (и знакогенератора в текстовом режиме) преобразует их в аналоговые напряжения основных цветов изображения (красного – R, зеленого – G, синего – B, из них складываются результирующие цвета точек на экране BM), которые выдаются на BM, последовательно формируя горизонтальные строки изображения.

Кроме того, ГК формирует для BM управляющие импульсы с TTL-уровнями для обозначения начала каждой строки изображения (строчные синхроимпульсы, HSYNC) и начала каждого кадра (кадровые синхроимпульсы, VSYNC). Изменения видеоданных согласно выполняемой программе производит ЦП или графический ускоритель (акселератор), которым оснащаются современные модели ВА. Акселераторы выполняют часть операций по изменению содержимого ВП без участия ЦП, получая от него соответствующие команды и параметры через управляющие регистры.

ПЗУ видеоBIOS хранит данные, относящиеся к работе ВА, а также исполняемые коды некоторых функций по управлению ВА. Функции видеоBIOS исполняет ЦП через вызов программного прерывания INT 10h. ВидеоBIOS ВА, работающих в видеорежимах выше VGA (называемых SuperVGA, SVGA), содержит дополнительные функции, отвечающие стандарту VESA (Video Electronics Standards Association). Этот стандарт определяет состав, назначение и порядок вызова дополнительных функций, расширяющих функции BIOS INT 10h (VESA BIOS Extension – VBE). Так VBE версии 1.2 определяет группу из 8 дополнительных

функций (код группы 4Fh). В версии 2.0 добавлены еще две новые функции.

Изготовители ВА нередко используют готовые микросхемы ГК, ВП и заказное программное обеспечение видеоBIOS других изготовителей.

Более подробные сведения о работе ВА содержатся в лекционном материале и рекомендованной литературе.

7.2 Определение названия фирмы-разработчика и номера версии видеоBIOS.

Искомые сведения записаны в коде ASCII в области данных видеоBIOS в диапазоне адресов от C006 : 0000 до C006 : 00FF. Внутри данного диапазона адресов могут содержаться не только коды латинских символов, поэтому отображать следует только байты, имеющие значения от 0 до 7Fh, т.е. именно коды латинских символов.

7.3 Определение параметров видеоадаптеров SVGA.

Получение данных о ВА любых изготовителей, отвечающих стандарту VESA, производится выполнением информационных функций VBE:

- общая информация о реализации VBE и ВА (код группы 4F, код функции в группе – 00h);
- характеристики видеорежима (код группы 4F, код функции в группе – 01h);
- определение текущего видеорежима ВА (код группы 4F, код функции в группе – 03h).

Эти функции выполняются стандартным для всех функций BIOS образом: в программно доступный регистр ЦП AX загружается код группы (в старшую часть AH) и код функции в группе (в младшую часть AL), в остальные регистры ЦП загружаются другие входные параметры, характерные для каждой запрашиваемой функции. Затем вызывается программное прерывание INT 10h.

После выполнения запрошенной функции в определенные регистры ЦП и в заданную область памяти выводятся выходные параметры (результаты выполнения функции).

Получение общей информации о реализации VBE и видеоадаптере

На входе: AH	4Fh
AL	00h
ES:DI	Указатель на буфер размером 256 байт (для VBE версии 1.2) или 512 байт (для VBE версии 2.0), в него записывается информация о ВА и реализации VBE
На выходе: AL	4Fh
AH	0 - в случае успешного завершения, 1 - в случае ошибки

Значения остальных регистров сохраняются.

Если видеоBIOS поддерживает данную функцию, то в регистре AL возвращается значение 4Fh. Если функция не реализована, тогда значение будет иным.

Перед вызовом этой функции необходимо выделить буфер размером 256 байт (для VBE версии 1.2) или 512 байт (для VBE версии 2.0) и поместить ссылку на него в регистр ES:DI. В таблице 1 представлено содержимое буфера.

Таблица 1 Общая информация о реализации VBE и BA

Смещение	Размер, байт	Описание
00h	4	В случае успешного завершения байты имеют значения ASCII-кодов символов, образующих слово "VESA"
04h	2	Номер версии VBE. Старший байт содержит старшую часть номера версии, младший байт – младшую часть номера версии
06h	4	Полный адрес (в области данных BIOS) начала строки текста, содержащей наименование изготовителя BA в ASCII-кодах, строка заканчивается нулевым байтом. Первое слово адреса содержит смещение, второе – код сегмента.
0Ah	4	Возможности BA, В реализации VBE версии 1.2 использован только бит D0. <ul style="list-style-type: none"> • Бит D0 содержит единицу, если ЦАП BA может работать с данными переменной длины. В противном случае ЦАП может использовать для представления каждого компонента цвета (RGB-красный, зеленый, синий) только 6 бит. • Бит D1 (VBE 2.0) содержит единицу, если BA не полностью совместим с VGA. • Бит D2 (VBE 2.0) содержит единицу, если BIOS не поддерживает другие функции VBE
0Eh	4	Адрес начала списка видеорежимов, поддерживаемых ГК и ЦАП. Первое слово адреса содержит смещение, второе – сегмент. Список состоит из 16-битовых величин, являющихся номерами режимов, и заканчивается кодом 0FFFFh.
12h	2	Объем ВП, выраженный в блоках размером по 64 Кбайт
Следующие поля таблицы поддерживаются только VBE версии 2.0		

14h	2	Дополнительный номер версии VBE (номер пересмотренной версии)
16h	4	Адрес начала строки, закрытой нулем. В строке содержится имя фирмы-изготовителя
1Ah	то же	Адрес начала строки, закрытой нулем. В строке записано название ВА
1Eh	то же	Адрес начала строки, закрытой нулем. В строке содержится дополнительный номер версии ВА
22h	222	Не используется
100h	256	Информация фирмы-изготовителя

Определение текущего видеорежима ВА

На входе: AH 4Fh

AL 03h

На выходе: AL 4Fh

AH 0 - в случае успешного завершения,
1 - в случае ошибки

BX Номер режима.

Текущий режим – видеорежим ВА, действующий в момент выполнения данной функции.

Характеристики режимов VBE приведены в Приложении 1.

Если текущий режим ВА не соответствует стандарту VBE, то возвращаемый в регистре BX код может означать номер действительного стандартного текущего режима IBM (см. Приложение 2).

Получение информации о характеристиках видеорежима ВА

На входе: AH 4Fh

AL 01h

CX Номер режима

ES:DI Указатель на буфер размером 256 байт (см. ниже)
для таблицы характеристик.

На выходе: AL 4Fh

AH 0 - в случае успешного завершения,
1 - в случае ошибки

Функция позволяет определить различные характеристики любого режима ВА, отвечающего стандарту VESA. Программа должна подготовить буфер и передать указатель на него функции. В случае успешного завершения в буфер будут записаны характеристики режима. В приводимой ниже табл. 2. описана часть информации, помещаемой в буфер.

Таблица 2 Некоторые характеристики видеорежима

Смещение	Размер	Описание
00h	слово	Биты атрибутов режима: D0 – ВА поддерживает режим; D1 – доступна дополнительная информация; D2 – поддерживаются функции BIOS; D3 - 1 – цветной режим, 0 – монохромный режим; D4 - 1 – графический режим, 0 – текстовый режим; D5 - 1 – назначение регистров или адресация портов регистров несовместимы с ВА VGA; D6 - 1 – нельзя использовать окно для доступа к ВП через область памяти с адресами A000:0000h-A000:FFFFh; D7 - 1 - можно использовать адресацию защищенного режима для отображения ВП на адресное пространство ЦП.
Следующие поля таблицы поддерживаются только VBE версии 1.2 и выше		
12h	слово	Разрешение по горизонтали в пикселях
14h	слово	Разрешение по вертикали в пикселях
16h	байт	Адрес начала списка видеорежимов, поддерживаемых ГК и ЦАП. Первое слово адреса содержит смещение, второе – сегмент. Список состоит из 16-битовых величин, являющихся номерами режимов, и заканчивается кодом 0FFFFh.
17h	байт	Объем ВП, выраженный в блоках размером по 64 Кбайт
19h	байт	Количество бит на пиксель
1Fh	байт	Количество бит, представляющих красный компонент пикселя
21h	байт	Количество бит, представляющих зелёный компонент пикселя
23h	байт	Количество бит, представляющих синий компонент пикселя
Следующие поля таблицы поддерживаются только VBE версии 2.0 и выше		
28h	4 байта	Адрес начала видеопамати
2Ch	4 байта	Адрес свободного пространства
30h	4 байта	Размер свободного пространства

При выполнении всех информационных функций VBE обязательна проверка содержимого регистров AH и AL!

7.4 Получение списка видеорежимов, поддерживаемых ВА.

Список режимов, получаемый при выполнении функции 4F00h (таблица 1), указывает лишь на способность ГК и ЦАП данного ВА поддерживать перечисленные видеорежимы.

Полная поддержка того или иного режима требует кроме соответствующих возможностей ГК и ЦАП еще и достаточного объема ВП, установленной в ВА. Видеоадаптеры не всегда содержат весь объем ВП, с которым может работать данная микросхема ГК.

Объем требуемой видеопамати для графических режимов определяется следующим образом:

$$V = N \times M \times K,$$

где N – число пикселей по горизонтали, M – по вертикали,
K – количество бит на пиксель.

Для окончательной проверки следует выполнить функцию 4F01h для всего списка режимов, определенных функцией 4F00h и проверить значения бита D0 атрибутов режима (таблица 2). Данная функция проводит необходимые вычисления и сравнения и сообщает о результате проверки в виде значения бита D0.

Стандартные видеорежимы VBE

Номер	Режим	Разрешение	Символьная матрица	Число цветов
100h	Графический	640 x 400	–	256
101h	Графический	640 x 480	–	256
102h	Графический	800 x 600	–	16
103h	Графический	800 x 600	–	256
104h	Графический	1024 x 768	–	16
105h	Графический	1024 x 768	–	256
106h	Графический	1280 x 1024	–	16
107h	Графический	1280 x 1024	–	256
108h	Текстовый	80 x 60	8 x 8	16
109h	Текстовый	132 x 25	8 x 16	16
10Ah	Текстовый	132 x 43	8 x 8	16
10Bh	Текстовый	132 x 50	8 x 16	16
10Ch	Текстовый	132 x 60	8 x 16	16
10Dh	Графический	320 x 200	–	32K
10Eh	Графический	320 x 200	–	64K
10Fh	Графический	320 x 200	–	16M
110h	Графический	640 x 480	–	32K
111h	Графический	640 x 480	–	64K
112h	Графический	640 x 480	–	16M
113h	Графический	800 x 600	–	32K
114h	Графический	800 x 600	–	64K
115h	Графический	800 x 600	–	16M
116h	Графический	1024 x 768	–	32K
117h	Графический	1024 x 768	–	64K
118h	Графический	1024 x 768	–	16M
119h	Графический	1280 x 1024	–	32K
11Ah	Графический	1280 x 1024	–	64K
11Bh	Графический	1280 x 1024	–	16M

Стандартные видеорежимы IBM

Номер	Режим	Разрешение	Символьная матрица	Число цветов
00h	Текстовый	40x25	9(8)x16(14)	16
01h	Текстовый	40x25	9(8)x16(14)	16
02h	Текстовый	80x25	9(8)x16(14)	16
03h	Текстовый	80x25	9(8)x16(14)	16
04h	Графический	320x200	–	4
05h	Графический	320x200	–	4
06h	Графический	640x200	–	2
07h	Текстовый	80x25	9(8)x16(14)	моно
0Dh	Графический	320x200	–	16
0Eh	Графический	640x200	–	16
0Fh	Графический	640x350	–	моно
10h	Графический	640x350x	–	16
11h	Графический	640x480	–	2
12h	Графический	640x480	–	16
13h	Графический	320x200	–	256

В скобках даны размеры символьной матрицы ВА стандарта EGA, без скобок – VGA.

ЛАБОРАТОРНАЯ РАБОТА N5

Программное извлечение флеш-диска

1. Цель работы

Практическое овладение навыками составления программ, работающих с USB-накопителями.

2. Рекомендуемая литература

Агуров П.В. Практика программирования USB. – СПб.: БХВ-Петербург, 2006. с. 69...73, 332...341, 368...374, 566...567.

SetupAPI Reference [Электронный документ]. – Режим доступа: <http://msdn.microsoft.com/en-us/library/dd445255.aspx> . – 5.11.2009.

GMax. Безопасное извлечение USB-устройств [Электронный документ] / GMax. – Режим доступа: http://wasm.ru/article.php?article=usb_eject . – 5.11.2009.

Программное извлечение USB-диска [Электронный документ]. – Режим доступа: <http://superadm.net/index.php?name=pages&op=view&id=126> . – 5.11.2009.

Аблязов Р. Работа с устройствами в Windows [Электронный документ] / Аблязов Р. – Режим доступа: <http://pblog.ru/?p=105> . – 5.11.2009.

3. Подготовка к работе

- 3.1. Изучить методические указания и рекомендованную литературу.
- 3.2. Подготовить ответы на контрольные вопросы.
- 3.3. Подготовить флеш-диск с интерфейсом USB.

4. Контрольные вопросы

- 4.1. Логическая и физическая архитектура USB?
- 4.2. Что является инициатором транзакции USB?
- 4.3. Почему не желательно вынимать USB-накопитель из разъёма без использования безопасного отключения?
- 4.4. Какой формат имеет PnP-идентификатор USB-устройств?
- 4.5. Какой формат имеет идентификатор экземпляра устройства USB-накопителя?
- 4.6. Что делает и какой формат имеет функция SetupDiGetDeviceRegistryProperty?
- 4.7. Что делает и какой формат имеет функция CM_Get_Device_ID?

4.8. Какие функции из SetupAPI.dll используются для нахождения и отключения устройства?

4.9. Как определить строковый идентификатор производителя и продукта?

5. Задание на выполнение работы

5.1. Используя среду программирования Free Pascal разработать программу, останавливающую только Вашу флешку для безопасного извлечения её из разъёма.

Программа может быть выполнена в консольном виде или с графическим пользовательским интерфейсом.

5.2. Подготовить отчёт и отчитаться о проделанной работе преподавателю.

6. Отчёт должен содержать:

6.1 Титульный лист (с названием ВУЗа, кафедры, лабораторной работы, а также фамилии И.О. студента, подготовившего отчёт).

6.2 Цель работы.

6.3 Графический алгоритм программы с краткими пояснениями.

6.4 Полный листинг программы с комментариями.

7. Общие сведения

При работе с USB-накопителями информации необходимо правильно извлекать эти устройства из системы. В операционных системах семейства Windows для этого имеется функция «*Безопасное извлечение устройства*», которую можно вызвать командой

rundll32.exe shell32.dll, Control_RunDLL hotplug.dll

Однако данную операцию можно выполнить и программным способом. Существует два метода программного извлечения устройства. Первый метод использует функции библиотеки **SetupAPI.dll**. Если устройство не готово для извлечения в данный момент, то выдаётся соответствующая ошибка или сообщение.

Второй метод использует функции прямого обращения к драйверу. В отличие от первого метода, он может извлечь устройство, даже если оно не готово, но при этом операционная система не уведомляется об отключении диска, что может вызвать ошибку при обращении к диску.

В приложении А приведён листинг программы, которая позволяет безопасно извлекать из системы первый попавшийся USB-накопитель. В данной программе проверяется PnP-идентификатор экземпляра устройства. Однако кроме проверки PnP-идентификатора есть и другие

способы нахождения конкретного устройства, например по строке описания этого устройства или идентификатору оборудования.

7.1 Библиотека SetupAPI.dll

SetupAPI это системный компонент, содержащий функции для установки драйвера устройства, и связывающий пользовательское приложение с устройством.

В дальнейшем рассмотрим некоторые функции из двух групп SetupAPI: *Device Installation Functions* и *PnP Configuration Manager Functions*. Первая группа включает функции установки устройств в Microsoft Windows, вторая – дополняет её функциями конфигурирования устройств (CM_xxx).

Функция **SetupDiGetClassDevs** возвращает дескриптор (манипулятор или handle) класса устройства, заданного в качестве параметра. Имеет следующий формат:

hDevInfoSet := **SetupDiGetClassDevs** (*ClassGuid*, *Enumerator*, *hwndParent*, *Flags*),

где *hDevInfoSet* – имя дескриптора класса;

ClassGuid – идентификатор класса;

Enumerator – системный компонент, определяющий PnP-идентификатор устройства;

hwndParent – дескриптор родительского окна;

Flags – флаг управления функцией. Может принимать пять значений, мы будем использовать флаг по умолчанию DIGCF_DEFAULT = 2.

Жесткие диски и USB-накопители имеют глобальный уникальный идентификатор класса *ClassGuid* = {4d36e967-e325-11ce-bfc1-08002be10318}.

Функция **SetupDiEnumDeviceInfo** возвращает структуру с информацией об очередном устройстве указанного класса. Если функция вернула значение TRUE, то информация извлечена успешно, а если FALSE, то в большинстве случаев это означает что мы пришли к концу списка.

SetupDiEnumDeviceInfo (*hDeviceInfoSet*, *MemberIndex*, *DeviceInfoData*),

где *hDeviceInfoSet* – дескриптор класса устройств;

MemberIndex – порядковый номер в списке устройств указанного класса;

DeviceInfoData – возвращаемая структура с информацией об устройстве.

Функция **SetupDiGetDeviceRegistryProperty** позволяет получить PnP свойства устройства.

SetupDiGetDeviceRegistryProperty (*hDeviceInfoSet*, *DeviceInfoData*, *Property*, *PropertyRegDataType*, *PropertyBuffer*, *PropertyBufferSize*, *RequiredSize*),

где *hDeviceInfoSet* – дескриптор класса устройств;

DeviceInfoData – указатель на структуру с информацией об устройстве;
Property – параметр, указывающий какое именно свойство требуется получить. Для получения строки с описанием устройства необходимо указать константу или *SPDRP_DEVICEDESC* (0x00000000) или *SPDRP_FRIENDLYNAME* (0x0000000C). Для получения идентификатора оборудования (*HardwareID*) необходимо указать константу *SPDRP_HARDWAREID* (0x00000001);

PropertyRegDataType – указатель на переменную, в которую помещается тип возвращаемых функцией данных;

PropertyBuffer – указатель на буфер, в который возвращается значение указанного свойства. Если этот параметр указать как *null* и *PropertyBufferSize* указан как 0, то функция возвращает в *RequiredSize* необходимый размер буфера;

PropertyBufferSize – размер буфера для получения значения свойства;

RequiredSize – дополнительный параметр для получения размера буфера, если не используется, то *null*.

Функция **SetupDiDestroyDeviceInfoList** удаляет всю информацию об устройствах указанного класса, и очищает память. В качестве параметра этой функции указывается дескриптор класса устройств (*hDeviceInfoSet*), который предварительно был получен функцией *SetupDiGetClassDevs*.

Функция **CM_Get_Parent** получает дескриптор родительской ветки в дереве устройств локальной машины.

CM_Get_Parent (*pdnDevInst*, *dnDevInst*, *ulFlags*),

где *pdnDevInst* – возвращаемый указатель на идентификатор родительского устройства;

dnDevInst – идентификатор устройства;

ulFlags – не используется, должен быть нулём.

Функция **CM_Get_Device_ID_Size** возвращает размер строки идентификатора устройства.

CM_Get_Device_ID_Size (*pullLen*, *dnDevInst*, *ulFlags*),

где *pullLen* – указатель на переменную для записи длины строки;

dnDevInst – идентификатор устройства;

ulFlags – не используется, должен быть нулём.

Функция **CM_Get_Device_ID** возвращает текстовый идентификатор экземпляра устройства ID.

CM_Get_Device_ID (*dnDevInst*, *Buffer*, *BufferLen*, *ulFlags*),

где *dnDevInst* – дескриптор устройства;

Buffer – указатель на буфер для записи строки идентификатора устройства;

BufferLen – длина строки идентификатора устройства;

ulFlags – не используется, должен быть нулём.

Функция **CM_Request_Device_Eject** выполняет безопасное извлечение устройства, а если это не возможно, то возвращает информацию об ошибке.

CM_Request_Device_Eject (*dnDevInst*, *pVetoType*, *pszVetoName*, *ulNameLength*, *ulFlags*),

где *dnDevInst* – дескриптор устройства;

pVetoType – дополнительный параметр для возвращения кода ошибки, если отказано в извлечении устройства;

pszVetoName – дополнительный параметр для возвращения текстового описания ошибки, в случае отказа в извлечении устройства;

ulNameLength – максимальная длина текстового описания ошибки;

ulFlags – не используется, должен быть нулем.

Функция **CM_Locate_DevNode** позволяет получить дескриптор устройства по строке идентификатора.

CM_Locate_DevNode (*pdnDevInst*, *pDeviceID*, *ulFlags*),

pdnDevInst – указатель на возвращаемый дескриптор устройства;

pDeviceID – указатель на строку идентификатора устройства;

ulFlags – флаг управления функцией. Может принимать четыре значения, мы будем использовать CM_LOCATE_DEVNODE_NORMAL = 0.

Функция **CM_Get_DevNode_Status** позволяет получить статус устройства, по которому можно определить, можно ли извлечь данное устройство. Если в статусе (*pulStatus*) возвращается флаг DN_REMOVABLE (0x4000), то устройство можно извлечь.

CM_Get_DevNode_Status (*pulStatus*, *pulProblemNumber*, *dnDevInst*, *ulFlags*),

где *pulStatus* – указатель на переменную со статусом устройства;

pulProblemNumber – указатель на переменную с номером ошибки;

dnDevInst – идентификатор устройства, у которого необходимо проверить статус;

ulFlags – не используется, должен быть нулем.

7.2 PnP-идентификаторы USB-накопителей

Каждое USB-устройство, спроектированное по спецификации PnP, должно иметь идентификатор, который однозначно определяет модель данного устройства.

Идентификатор устройства должен иметь строго определённый для данного класса устройств формат. Для USB-устройства идентификатор имеет следующий формат:

USB\Vid_vvvv&Pid_dddd&Rev_rrrr,

где **vvvv** – код идентификатора производителя, зарегистрированного в Комитете USB-производителей;

dddd – идентификатор, присвоенный производителем данной модели USB-устройства;

rrrr – номер версии разработки.

Все эти поля вводятся как шестнадцатеричные числа.

При идентификации USB-накопителя драйвером порта, последний создаёт новый *идентификатор устройства* (device ID) и набор *идентификаторов оборудования* (hardware ID). Первая строка идентификатора оборудования должна совпадать с идентификатором устройства. Идентификатор оборудования имеет следующий формат

```
USBSTOR\t*v(8)p(16)r(4)
USBSTOR\t*v(8)p(16)
USBSTOR\t*v(8)
USBSTOR\v(8)p(16)r(1)
v(8)p(16)r(1)
USBSTOR\GenDisk
GenDisk,
```

где **t*** – тип устройства (для USB-накопителей DISK);

v(8) – идентификатор производителя из 8 символов;

p(16) – идентификатор продукта;

r(4) – идентификатор версии разработки.

Идентификатор оборудования необходим для точного подбора драйвера для устройства.

Кроме *идентификатора устройства* (device ID) каждый накопитель имеет *идентификатор экземпляра* (instance ID), который отличает устройство от других устройств того же типа. Идентификатор экземпляра может определять устройство относительно шины (например, учитывать USB-порт, к которому подключено устройство) или представлять глобально уникальный дескриптор (например, серийный номер устройства). Идентификаторы устройства и экземпляра дают *идентификатор экземпляра устройства* (device instance ID, DIID или код экземпляра устройства), который однозначно идентифицирует экземпляр устройства в системе.

Просмотреть код экземпляра устройства можно в диспетчере устройств, выбрав интересующее дисковое устройство и перейдя на вкладку «Сведений» в свойствах этого устройства.

Например: код экземпляра устройства
USBSTOR\DISK&VEN_JETFLASH&PROD_TS256MJF2A/120&REV_8.07\6
&38D7AE47&0&7ZNDZ4S6&0 содержит идентификатор устройства
USBSTOR\DISK&VEN_JETFLASH&PROD_TS256MJF2A/120&REV_8.07 и
 идентификатор экземпляра **6&38D7AE47&0&7ZNDZ4S6&0**.
 Идентификатор оборудования для того же устройства содержит:

```
USBSTOR\DiskJetFlashTS256MJF2A/120__8.07
USBSTOR\DiskJetFlashTS256MJF2A/120__
```

USBSTOR\DiskJetFlash

USBSTOR\JetFlashTS256MJF2A/120__8

JetFlashTS256MJF2A/120__8

USBSTOR\GenDisk

GenDisk

Листинг 1 – Программа EjectFlesh

```
program EjectFlesh;
{$MODE OBJFPC}
```

uses

```
Windows, strings;
```

const

```
setupapi = 'SetupApi.dll';
GUID_DEVCLASS_DISKDRIVE: TGUID = (D1: $4D36E967; D2: $E325; D3: $11CE;
D4: ($BF, $C1, $08, $00, $2B, $E1, $03, $18)); // GUID класса накопителей
GUID_DEVCLASS_USB: TGUID = (D1: $36FC9E60; D2: $C465; D3: $11CF; D4:
($44, $45, $53, $54, $00, $00, $00, $00)); // GUID класса хост-контроллера и USB
хабов;
```

type

```
HDEVINFO = THandle;
```

```
PSP_DEVINFO_DATA = ^SP_DEVINFO_DATA;
```

```
SP_DEVINFO_DATA = packed record
```

```
    cbSize : DWORD;
```

```
    ClassGuid : TGUID;
```

```
    DevInst : DWORD;
```

```
    Reserved : DWORD;
```

```
end;
```

var

```
    q: char;
```

```
    hDevInfoSet: HDEVINFO;
```

```
    DevInfo: SP_DEVINFO_DATA;
```

```
    i: Integer;
```

```
    Parent: DWORD;
```

```
    VetoName: PChar;
```

```
    VetoNameString: String;
```

```
// функции из SetupApi.dll
```

```
function SetupDiGetClassDevsA(ClassGuid: PGUID; Enumerator: PChar; hwndParent:
HWND; Flags: DWORD): HDEVINFO; stdcall; external setupapi;
```

```
function SetupDiEnumDeviceInfo(DeviceInfoSet: HDEVINFO; MemberIndex: DWORD;
DeviceInfoData: PSP_DEVINFO_DATA): boolean; stdcall; external setupapi;
```

```
function SetupDiDestroyDeviceInfoList(DeviceInfoSet: HDEVINFO): boolean; stdcall;
external setupapi;
```

```
function CM_Get_Parent(pdnDevInst: PDWORD; dnDevInst: DWORD; ulFlags:
DWORD): DWORD; stdcall; external setupapi;
```

```
function CM_Get_Device_ID_Size(pulLen: PDWORD; dnDevInst: DWORD; ulFlags:
DWORD): DWORD; stdcall; external setupapi;
```

```

function CM_Get_Device_IDA(dnDevInst: DWORD; Buffer: PChar; BufferLen:
DWORD; ulFlags: DWORD): DWORD; stdcall; external setupapi;
function CM_Locate_DevNodeA(pdnDevInst: PDWORD; pDeviceID: PChar; ulFlags:
DWORD): DWORD; stdcall; external setupapi;
function CM_Request_Device_EjectA(dnDevInst: DWORD; pVetoType: Pointer;
pszVetoName: PChar; ulNameLength: DWORD; ulFlags: DWORD): DWORD; stdcall;
external setupapi;

```

```

function CompareMem(p1, p2: Pointer; len: DWORD): boolean;

```

```

var

```

```

    i: DWORD;

```

```

begin

```

```

    result := false;

```

```

    if len = 0 then exit;

```

```

    for i := 0 to len-1 do

```

```

        if PByte(DWORD(p1) + i)^ <> PByte(DWORD(p2) + i)^ then exit;

```

```

    result := true;

```

```

end;

```

```

function IsUSBDevice(DevInst: DWORD): boolean;

```

```

var

```

```

    IDLen: DWORD;

```

```

    ID: PChar;

```

```

    IDString: String;

```

```

begin

```

```

    result := false;

```

```

    IDString := "";

```

```

    if (CM_Get_Device_ID_Size(@IDLen, DevInst, 0) <> 0) or (IDLen = 0) then exit;

```

```

    inc(IDLen);

```

```

    ID := GetMemory(IDLen);

```

```

    if ID = nil then exit;

```

```

    if ((CM_Get_Device_IDA(DevInst, ID, IDLen, 0) <> 0) or (not (CompareMem(ID,
PChar('USBSTOR'), 7)))) then

```

```

        begin

```

```

            IDString := StrPas(ID);

```

```

            FreeMemory(ID);

```

```

            exit;

```

```

        end;

```

```

    IDString := StrPas(ID);

```

```

    Write('Eject flash-disk?(y - yes; Any other key - no)');

```

```

    ReadLn(q);

```

```

    if q = 'y' then result := true;

```

```

    FreeMemory(ID);

```

```

end;

```

```

BEGIN

```

```

    DevInfo.cbSize := sizeof(SP_DEVINFO_DATA);

```

```

    hDevInfoSet := SetupDiGetClassDevsA(@GUID_DEVCLASS_DISKDRIVE, nil, 0, 2);

```

```

    if hDevInfoSet = INVALID_HANDLE_VALUE then exit;

```

```

    i := 0;

```


10

```
    while (SetupDiEnumDeviceInfo(hDevInfoSet, i, @DevInfo)) do
begin
    if IsUSBDevice(DevInfo.DevInst) then
    begin
        if CM_Get_Parent(@Parent, DevInfo.DevInst, 0) = 0 then
        begin
            VetoName := GetMemory(260);
            if (CM_Request_Device_EjectA(Parent, nil, VetoName, 260, 0) <> 0) then
            begin
                if (CM_Locate_DevNodeA(@Parent, VetoName, 0) <> 0) then
                begin
                    FreeMemory(VetoName);
                    continue;
                end;
                FreeMemory(VetoName);
                if (CM_Request_Device_EjectA(Parent, nil, nil, 0, 0) <> 0) then continue;
            end;
            VetoNameString := StrPas(VetoName);
            FreeMemory(VetoName);
            break;
        end;
    end;
    inc(i);
end;
SetupDiDestroyDeviceInfoList(hDevInfoSet);
END.
```

ЛАБОРАТОРНАЯ РАБОТА N6

Разработка драйвера для манипулятора типа «мышь»

1. Цель работы

Получение навыков создания фильтрующего драйвера WDM для оптического манипулятора типа «мышь» с интерфейсом USB.

2. Рекомендуемая литература

Агуров П.В. Практика программирования USB. – СПб.: БХВ-Петербург, 2006.

Комиссарова В. Программирование драйверов для Windows. – СПб.: БХВ-Петербург, 2007.

Они У. Использование Microsoft Windows Driver Model. 2-е изд. (+CD). Для профессионалов. – СПб.: Питер, 2007.

Солдатов В.П. Программирование драйверов Windows. Изд. 3-е, перераб. и доп. – М.: ООО «Бином-Пресс», 2006.

3. Подготовка к работе

- 3.1. Изучить методические указания и рекомендованную литературу.
- 3.2. Подготовить ответы на контрольные вопросы.

4. Контрольные вопросы

- 4.1. С помощью чего осуществляется ввод и вывод в драйверах?
- 4.2. Какую архитектуру имеет подсистема ввода-вывода в Windows XP?
- 4.3. Какую роль выполняют драйверы в операционной системе?
- 4.4. Какие типы драйверов существуют в операционной системе Windows XP?
- 4.5. Что называют объектом физического устройства и объектом функционального устройства?
- 4.6. Что такое расширение объекта устройства?
- 4.7. В чём заключаются основные особенности драйверной модели WDM?
- 4.8. Для чего предназначены драйверы шины, функциональные драйверы и фильтрующие драйверы? Что такое стек драйверов?
- 4.9. Назначение функции DriverEntry. Какую особенность имеет функция DriverEntry в фильтрующих драйверах?
- 4.10. Для чего используются функции завершения для пакетов?

5. Задание на выполнение работы

5.1. Получить у преподавателя вариант задания.

5.2. Разработать драйвер манипулятора «мышь» с интерфейсом USB, изменяющий поведение устройства в соответствии со своим вариантом.

5.3. Установить драйвер в системе и проверить его работоспособность. Восстановить «драйвер по умолчанию».

5.4. Работу оформить в виде отчета и сдать преподавателю.

Табл. 5.1 – Варианты заданий

№ варианта	Задание
1	Разработать драйвер, отключающий нажатие правой кнопки мыши.
2	Разработать драйвер, отключающий нажатие средней кнопки мыши.
3	Разработать драйвер, переопределяющий между собой нажатия левой и правой кнопок.
4	Разработать драйвер, переопределяющий между собой нажатия средней и правой кнопок.
5	Разработать драйвер, в результате работы которого указатель мыши двигался бы дискретно со смещением, кратным 10 пикселям по горизонтали и 5 пикселям по вертикали.
6	Разработать драйвер, переопределяющий между собой нажатия средней и левой кнопок.
7	Разработать драйвер, в результате работы которого указатель мыши двигался бы дискретно со смещением, кратным 5 пикселям по горизонтали и 10 пикселям по вертикали.
8	Разработать драйвер, отключающий нажатие левой кнопки мыши.
9	Разработать драйвер, в результате работы которого нажатие средней кнопки мыши было бы эквивалентно нажатию правой кнопки.
10	Разработать драйвер, в результате работы которого нажатие средней кнопки мыши было бы эквивалентно нажатию левой кнопки.

6. Отчёт должен содержать:

- 6.1 Титульный лист (с названием ВУЗа, кафедры, номера лабораторной работы и её названия, фамилии И.О. студента, подготовившего отчёт)
- 6.2 Цель работы.
- 6.3 Номер варианта и формулировку индивидуального задания;
- 6.4 Программный код драйвера с пояснениями и комментариями

7. Общие сведения

7.1 Понятие драйвера

Драйвер удобно рассматривать как контейнер для функций, вызываемых операционной системой для выполнения различных операций, относящихся к работе оборудования. Некоторые функции (такие как DriverEntry и AddDevice, а также диспетчерские функции для некоторых типов запросов IRP) присутствуют во всех контейнерах без исключения. В большинстве драйверов присутствуют диспетчерские функции для нескольких типов IRP. Драйвер, как и приложение, представляет собой исполняемый файл. Он обладает расширением .SYS, но с точки зрения структуры ничем не отличается от любого другого 32-разрядного графического или консольного приложения. Как и приложение, драйвер использует ряд вспомогательных функций, многие из которых komponуются динамически из ядра операционной системы, из драйвера класса или другой дополнительной библиотеки.

Однако, в отличие от приложения, в драйвере отсутствует главный модуль. Вместо него он содержит набор функций, вызываемых системой в тот момент, когда она сочтет нужным. Конечно, эти функции могут пользоваться другими вспомогательными функциями в драйвере, в статических библиотеках и операционной системе, но сам драйвер не отвечает ни за что, кроме своего оборудования; система отвечает за все остальное, в том числе и за принятие решений по поводу того, когда должен выполняться код драйвера.

Далее приводится один из возможных сценариев вызова функций драйвера операционной системой:

1. Пользователь подключает устройство. Система загружает исполняемый файл драйвера в виртуальную память и вызывает функцию DriverEntry. Функция DriverEntry выполняет некоторые операции и возвращает управление.
2. Администратор Plug and Play (PnP Manager) вызывает функцию AddDevice. Функция также выполняет некоторые операции и возвращает управление.

3. PnP Manager отправляет драйверу несколько пакетов IRP. Диспетчерская функция поочередно обрабатывает все пакеты и возвращает управление.
4. Приложение открывает манипулятор (handle) устройства, на что система посылает драйверу очередной пакет IRP. Диспетчерская функция выполняет некоторые операции и возвращает управление.
5. Приложение пытается прочитать данные, что также сопровождается отправкой IRP. Диспетчерская функция помещает IRP в очередь и возвращает управление.
6. Предыдущая операция ввода/вывода завершается выдачей аппаратного прерывания, к которому подключен драйвер. Обработчик прерывания выполняет некоторые операции, планирует вызов DPC и возвращает управление.
7. Выполняется функция отложенного вызова процедур (DPC). Среди прочего, она удаляет пакет IRP, поставленный в очередь на шаге 5, и программирует оборудование на чтение данных. Затем функция DPC возвращает управление системе.
8. Проходит время. Система многократно вызывает функции драйвера.
9. В конечном итоге пользователь отключает устройство. PnP Manager отправляет драйверу несколько пакетов IRP; драйвер обрабатывает их и возвращает управление. Операционная система вызывает функцию DriverUnload, которая выполняет минимальный объем работы и возвращает управление. Система выгружает код драйвера из виртуальной памяти.

На каждом шаге процесса система решает, какие действия должен выполнить драйвер, будь то инициализация, обработка IRP, обработка прерывания или что-нибудь еще. Таким образом, система выбирает нужную функцию драйвера, а функция делает то, что ей положено, и возвращает управление системе.

7.2 Состав драйвера

Программный код разрабатываемого фильтрующего драйвера состоит из следующих частей:

- **Раздел директив предкомпилятора** – кроме подключения заголовочных файлов содержит инструкции компилятора о распределении кода и данных между перемещаемой и неперемещаемой памятью, указывающие, какие части драйвера должны постоянно оставаться резидентными, а какие могут выгружаться.

- **Функция DriverEntry** – устанавливает диспетчерские функции для всех типов IRP. Фильтрующий драйвер должен поддерживать все типы IRP, которые поддерживаются драйвером, находящимся непосредственно под ним. Если бы фильтр оставил элемент таблицы MajorFunction в состоянии по умолчанию, то IRP этого типа отклонялись бы с кодом STATUS_INVALID_DEVICE_REQUEST.

- **Функция AddDevice** – вызывается для каждого подходящего устройства. В этой функции создается объект устройства и подключение его к стеку драйверов.

- **Функция ReadComplete** – данная функция установлена функцией DispatchRead в качестве функции завершения для пакетов IRP_MJ_READ в тексте примера драйвера. В этой функции производится замена данных пакета с целью изменения функциональности устройства.

- **Функция DispatchAny** – передает большинство IRP, кроме тех, для которых в функции DriverEntry установлены особые диспетчерские функции, вниз по стеку драйверов.

- **Функция DispatchPnp** – диспетчерская функция для обработки пакетов IRP_MJ_PNP обрабатывает несколько особых случаев для дополнительных кодов функций пакетов IRP_MJ_PNP.

- **Функция DispatchPower** – данная функция обрабатывает запросы управления питанием. При получении IRP с дополнительным кодом IRP_MN_SET_POWER состояние энергопотребления системы пересылается следующему драйверу.

- **Функция DriverUnload** – при отключении устройства диспетчер ввода-вывода вызывает функцию DriverUnload, которая должна удалить код драйвера из виртуальной памяти.

- **Функция DispatchRead** – в программном коде примера эта функция установлена в качестве диспетчерской для запросов IRP_MJ_READ. В ней устанавливается процедура обратного вызова для обработки пакетов этого типа. Когда пакет будет возвращаться обратно, установленная функция обратного вызова перехватит его с целью заданного изменения функциональности устройства.

7.3 Программные средства для разработки и компиляции драйвера

Для компиляции драйвера на компьютере должен быть установлен пакет Microsoft [Windows Driver Development Kit](#) (DDK) для Windows Server 2003. Компиляция и сборка драйвера производится с помощью утилиты Build пакета DDK. Для этого в меню запуска программ Пуск – Программы – ... следует выбрать запуск соответствующей среды, в результате чего появится консольное окно, для которого уже будут должным образом установлены переменные окружения. В том случае, если у разработчика имеются файлы makefile, sources (описывающие процесс сборки данного конкретного драйвера), а пакет DDK установлен корректно, то необходимо лишь перейти в рабочую директорию проекта командой cd и ввести команду build.

Поскольку в пакете DDK отсутствует интегрированная среда разработки, для написания драйвера необходимо средство редактирования исходного кода. Наиболее удобными средствами являются [Microsoft Visual C++](#) или Microsoft Visual Studio. Эти средства

предоставляют такие удобства написания кода как подсветка и проверка синтаксиса. Компилятор и редактор связей Visual Studio C++ создают нормальный бинарный код, вполне работоспособный при указании соответствующих опций (настроек) компиляции, однако эталоном следует считать бинарный код, получающийся при компиляции кода драйвера с использованием утилиты Build из состава пакета DDK. Разумеется, встроенный интерактивный отладчик Visual Studio и прилагаемая документация становятся для разработки драйвера совершенно бесполезными, поскольку не предназначены для работы с программным обеспечением для режима ядра.

7.4 Порядок установки драйвера в системе

1. Открыть Диспетчер устройств.
2. Открыть окно «Свойства» для установленной в системе мыши.
3. Открыть вкладку «Драйвер», и затем нажать кнопку «Обновить драйвер».
4. Следовать указаниям Мастера обновления драйверов, при появлении запроса выбрать опцию «Вывести список всех известных драйверов». Нажать кнопку «Далее».
5. Нажать кнопку «Установить с диска» и затем указать месторасположение .inf-файла.
6. Завершить оставшуюся часть установки. Потребуется перезагрузить компьютер после установки фильтрующего драйвера.

Для восстановления драйвера по умолчанию необходимо на вкладке «Драйвер» в «Свойствах» манипулятора нажать кнопку «Откатить» и согласится с появившимся запросом.