7. ОПЕРАЦИОННАЯ СИСТЕМА MICROSOFT WINDOWS

Не будет преувеличением сказать, что операционная система WINDOWS версии 3.0 открыла новый этап в жизни персонального компьютера IBM PC. Удобная, интуитивно ясная графическая оболочка, аналогичная используемой в компьютерах фирмы Apple, быстро завоевала популярность у пользователей IBM PC. В нашей стране WINDOWS также получила широкое распространение, особенно после того, как стали доступны компьютеры с памятью 2 мегабайта и более. Новая верисия WINDOWS 3.1 обладает улучшенным графическим интерфейсом и большей производительностью.

Однако WINDOWS - это не только графика и перекрывающиеся окна. С точки зрения программиста, WINDOWS является графически ориентированной мультизадачной операционной системой, работающей в защищённом режиме процессора.

Стандартные приложения WINDOWS используют как бы "вывернутую наизнанку" логику работы программ. Обычно, создавая программы для MS-DOS, программист полностью планирует сценарий работы программы. Он последовательно определяет, что и когда программа будет выводить на экран, что и когда должно быть введено с клавиатуры.

Логика работы приложений WINDOWS основана на использовании очередей событий. Событием является, например, нажатие клавиши на клавиатуре, прерывание от системного таймера, нажатие на кнопку мыши или перемещение мыши. Для каждого приложения создаётся своя собственная очередь, в которую записываются коды событий и дополнительная информация о событиях (например, код нажатой клавиши).

Приложение WINDOWS в цикле опрашивает состояние очереди событий. Если происходит какое либо событие, вызывается соответствующий модуль, который и реагирует на событие. Перед обработкой событие удаляется из очереди.

Программа не может знать последовательности событий, поэтому её поведение полностью определяется событиями, происходящими в системе. Такая логика работы программ называется логикой, управляемой событиями, а сами программы - программами, управляемыми событиями.

Кроме изменений в логике работы программ, WINDOWS реализует новый подход к использованию системных ресурсов. Операционная система MS-DOS не содержит сколько-нибудь значительной поддержки диалоговых программ, поэтому программисты вынуждены прибегать к использованию специальных библиотек, содержащих программы меню, управления окнами, модули для работы с мышью и графикой. Все необходимые модули прикомпоновывались к основной программе, сильно увеличивая размер загрузочного модуля.

Операционная система WINDOWS в своём ядре содержит все модули, необходимые для организации высококачественного графического диалога с пользователем. Причём нет необходимости в их компоновке с основным модулем - весь сервис WINDOWS реентерабельный, одни и те же модули используются всеми одновременно работающими приложениями.

Новый подход в создании программ, предложенный в WINDOWS, способствует созданию удобных программ с дружественным диалогом, использующих все доступные ресурсы компьютера. Однако WINDOWS едва ли получила бы такое широкое распространение, если бы она не позволяла одновременно с приложениями WINDOWS использовать разработанные ранее программы, ориентированные на MS-DOS (DOS-программы).

В зависимости от типа процессора и режима, в котором работает WINDOWS (стандартный или расширенный) выполнение DOS-программ организуется по-разному.

В стандартном режиме и при использовании процессора i80286 DOS-программы выполняются в реальном режиме процессора. В любой момент времени может выполняться только одна DOS-программа, остальные запущенные DOS-программы находятся в "замороженном" состоянии.

Расширенный режим требует наличия процессора i80386 или i80486. Он организует параллельную работу всех запущенных приложений WINDOWS и DOS-программ, выделяя в распоряжение каждому приложению и программе так называемую виртуальную машину. Виртуальную машину, на которой выполняется DOS-программа, мы будем называть виртуальной DOS-машиной.

Что такое виртуальная машина, хорошо знают те, кто работал на ЕС ЭВМ в операционной системе VM (известна также под названием "СВМ" - система виртуальных машин). Все ресурсы компьютера разделяются между запущенными программами. В распоряжение каждой программы предоставляется собственный виртуальный процессор, виртуальная память, виртуальная система прерываний. Дисплей, клавиатура и мышь используются совместно и распределяются той программе, которая в данный момент работает с оператором (про такую программу разработчики приложений WINDOWS говорят, что она имеет "фокус ввода" - input focus).

Когда WINDOWS работает в расширенном режиме, процессор может находиться либо в защищённом, либо в виртуальном режиме. DOS-программа получает управление, когда процессор находится в виртуальном режиме. При этом, используя механизм трансляции страниц, WINDOWS отображает участок расширенной памяти в область младших адресов задачи, в рамках которой работает DOS-программа. Таким образом для DOS-программы организуется виртуальное адресное пространство, которая программа не может отличить от реальной памяти, расположенной в пределах первого мегабайта.

Дополнительно в распоряжение DOS-программы может быть предоставлена расширенная или дополнительная память (интерфейсы EMS и XMS). Размер этой памяти можно задать в pif-файле (см. руководство по WINDOWS).

Если DOS-программа выдаёт команду программного прерывания, процессор переходит в защищённый режим и управление передаётся ядру WINDOWS. WINDOWS эмулирует для DOS-программы все прерывания DOS и BIOS, аналогично тому, как это делают DOS-экстендеры. Поэтому виртуальную DOS-машину, на которой выполняется DOS-программа, можно считать DOS-экстендером.

Для приложений WINDOWS и виртуальной DOS-машины доступен интерфейс DPMI. Это, в частности, означает, что виртуальная DOS-машина не просто допускает выполнение DOS-программ, но и предоставляет им дополнительные возможности, которые они не имели бы, если бы работали под управлением "настоящей" MS-DOS.

Описанию этих дополнительных возможностей и посвящена данная глава книги.

7.1. Использование функций DPMI

Приведённая ниже программа демонстрирует использование функций интерфейса DPMI, описанного в предыдущей главе. Эта программа может работать только под управлением WINDOWS версий 3.0, 3.1 и только в расширенном режиме на процессорах i80386 или i80486. Такое ограничение связано с тем, что только в расширенном режиме существует понятие виртуальной DOS-машины, и только в этом режиме DOS-программа может воспользоваться сервисом DPMI.

Вы можете также попробовать запустить эту программу под управлением DOS-экстендера, входящего в состав интегрированной системы разработки программ Borland C++ 3.1. Запустите программу DPMIRES.EXE, входящую в состав Borland C++ 3.1, и затем - программу, приведённую ниже. (DOS-экстендеры, входящие в состав Borland C++ 2.0 или 3.0, не вполне совместимы с DPMI, поэтому наш пример с этими системами работать не будет).

Программа начинает свою работу с проверки доступности сервера DPMI, при этом не делается никаких предположений относительно средств, обеспечивающих присутствие DPMI. Это означает, что вы можете проверить работоспособность этой программы в различных средах, предоставляющих интерфейс DPMI, например на виртуальной DOS-машине операционной системы OS/2 версии 2.0.

Программа демонстрирует возможность вызова в защищённом режиме прерываний реального режима. В первой части программы вывод на экран и ввод с клавиатуры выполняется в защищённом режиме, но с использованием привычных вам прерываний реального режима.

Во второй части программы демонстрируется непосредственная запись в видеопамять. При этом для адресации видеопамяти программа заказывает селектор в таблице LDT с помощью специально предназначенных для этого функций DPMI.

```
Листинг 21. Использование интерфейса DPMI
Файл dpmi.c
-------------------------------------------------------------


#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <stdarg.h>

typedef struct {
         unsigned long edi, esi, ebp, reserved, ebx, edx, ecx, eax;
         unsigned flags, es, ds, fs, gs, ip, cs, sp, ss;
} RM_INT_CALL;

#define MONO_MODE           0x07
```

```
#define BW_80_MODE          0x02
#define COLOR_80_MODE       0x03

// Макро для вычисления линейного адреса исходя из
// логического адреса реального режима

#define ABSADDR(seg, ofs) \
         ((((unsigned long) seg) << 4) + ((ofs) & 0xFFFF))

typedef struct {                                                    // байт
доступа
         unsigned accessed   : 1;
         unsigned read_write : 1;
         unsigned conf_exp   : 1;
         unsigned code       : 1;
         unsigned xsystem    : 1;
         unsigned dpl        : 2;
         unsigned present    : 1;
} ACCESS;

typedef struct {                // дескриптор
         unsigned        limit;
         unsigned        addr_lo;
         unsigned char   addr_hi;
         ACCESS          access;
         unsigned        reserved;
} DESCRIPTOR;

// Структура для записи информации о памяти

typedef struct {
        unsigned long avail_block;
        unsigned long max_page;
        unsigned long max_locked;
        unsigned long linadr_space;
        unsigned long total_unlocked;
        unsigned long free_pages;
        unsigned long tot_phys_pages;
        unsigned long free_linspace;
        unsigned long size_fp;
        char reserved[12];
} PMI;


void dos_exit(unsigned);
void dpmi_init(void);
void set_pmode(void);
void cdecl pm_printf(const char *, ...);
void pm_puts(char *);
void pm_putch(int);
int  rm_int(unsigned, unsigned , RM_INT_CALL far *);
int  mi_show(void);
unsigned get_sel(void);
int set_descriptor(unsigned pm_sel, DESCRIPTOR far *desc);
void vi_print(unsigned int x, unsigned int y, char *s, char attr);
void vi_hello_msg(void);



void main() {

        clrscr();
        printf("DOS Protected Mode Interface Demo, © Frolov A.V., 1992\n\r"
```

```
                "------------------------------------------------------
\n\r\n\r");

// Определяем текущий видеорежим и
// сегментный адрес видеобуфера

        video_init();

// Инициализируем защищённый режим

        dpmi_init();

        printf("\n\r\n\r\n\rДля входа в защищённый режим нажмите любую
клавишу...");
        getch();

// Входим в защищённый режим

        set_pmode();

// Стираем экран и выводим сообщение, находясь в
// защищённом режиме. Пользуемся выводом через
// эмулируемое прерывание реального режима DOS

        textcolor(BLACK);       textbackground(LIGHTGRAY);      clrscr();
        pm_printf(" Установлен защищённый режим работы процессора!\n\r"
                        " ----------------------------------------------
\n\r\n\r");

// Выводим текущую информацию о распределении памяти

        mi_show();

        pm_printf("\n\r\n\r\n\r Для продолжения нажмите любую клавишу...");
        getch();

        clrscr();

// Получаем селектор для непосредственного доступа к видеопамяти

        alloc_videosel();

        pm_printf("\n\r\n\r\n\r Для продолжения нажмите любую клавишу...");
        getch();
        clrscr();

// Выводим сообщения, пользуясь непосредственным доступом
// к видеопамяти

        vi_hello_msg();
        vi_print(0, 3,
                " Для возврата в реальный режим нажмите любую клавишу",
0x7f);
        getch();

// Освобождаем полученный селектор

        free_videosel();

        textcolor(LIGHTGRAY);   textbackground(BLACK);  clrscr();

// Завершаем работу программы выходом в DOS

        dos_exit(0);
```

```
}

// -------------------------------------------------
// Процедура для завершения работы программы
// -------------------------------------------------

void dos_exit(unsigned err) {
         asm mov ax, err
         asm mov ah, 04ch
         asm int 21h
}

// -------------------------------------------------
// Инициализация для работы с DPMI
// -------------------------------------------------

        union REGS inregs, outregs;
        struct SREGS segregs;
        void (far *pm_entry)();
        unsigned hostdata_seg, hostdata_size, dpmi_flags;

void dpmi_init(void) {

// Проверяем доступность и параметры сервера DPMI

        inregs.x.ax = 0x1687;
        int86x(0x2F, &inregs, &outregs, &segregs);
        if(outregs.x.ax != 0) {
                printf("Сервер DPMI не активен."); exit(-1);
        }

// Определяем версию сервера DPMI

        printf("Версия сервера DPMI: \t\t\t%d.%d\n",
                outregs.h.dh, outregs.h.dl);

// Определяем тип процессора

        printf("Тип процессора:\t\t\t\t");
        if(outregs.h.cl == 2) printf("80286");
        else if(outregs.h.cl == 3) printf("80386");
        else if(outregs.h.cl == 4) printf("80486");

// Определяем возможность работы с 32-разрядными
// программами

        dpmi_flags = outregs.x.bx;
        printf("\nПоддержка 32-разрядных программ:\t");
        if(dpmi_flags && 1) printf("ПРИСУТСТВУЕТ");
        else printf("ОТСУТСТВУЕТ");

// Определяем размер области памяти для сервера DPMI

        hostdata_size = outregs.x.si;
        printf("\nРазмер памяти для сервера DPMI:\t\t%d байт",
                hostdata_size * 16);

// Определяем адрес точки входа в защищённый режим

        FP_SEG(pm_entry) = segregs.es;
        FP_OFF(pm_entry) = outregs.x.di;
        printf("\nАдрес точки входа в защищённый режим: \t%Fp\n",
                pm_entry);
```

```
// Заказываем память для сервера DPMI

        if(hostdata_size) {
                if(_dos_allocmem(hostdata_size, &hostdata_seg) != 0) {
                        printf("Мало стандартной памяти"); exit(-1);
                }
        }

}

// ------------------------------------------------
// Процедура для установки защищённого режима
// ------------------------------------------------

void set_pmode() {

// Входим в защищённый режим

         asm {
                  mov ax, hostdata_seg
                  mov es, ax
                  mov ax, dpmi_flags
         }
         (*pm_entry)();

}

// ----------------------------------------
// Процедура вывода символа на экран в
// защищённом режиме
// ----------------------------------------

void pm_putch(int chr) {

// Структура для вызова прерывания должна
// быть определена как статическая

         static RM_INT_CALL regs;
         static RM_INT_CALL far *pregs = (void far *) 0;

// В первый раз инициализируем структуру
// и указатель на неё

         if (!pregs) {
                  pregs = &regs;
                  memset(pregs, 0, sizeof(RM_INT_CALL));
                  regs.eax = 0x0200;
         }
         regs.edx = chr;

// Вызываем прерывание DOS для вывода символа

         rm_int(0x21, 0, pregs);
}

// ----------------------------------------
// Процедура вывода строки на экран в
// защищённом режиме
// ----------------------------------------

void pm_puts(char *str_ptr) {
         while (*str_ptr) { pm_putch(*str_ptr); str_ptr++; }
}
```

```
// -----------------------------------------
// Процедура вывода строки на экран в
// защищённом режиме, аналог функции printf()
// -----------------------------------------

void cdecl pm_printf(const char *fmt, ...)
{
        char buffer[512], *sptr=buffer;
        va_list marker;
        va_start(marker, fmt);
        vsprintf(buffer, fmt, marker);
        va_end(marker);
        while (*sptr) pm_putch(*sptr++);
}

// -----------------------------------------
// Процедура вызова прерывания реального режима
// -----------------------------------------

int rm_int(unsigned int_number, // номер прерывания
                        unsigned params,              // количество слов
параметров,
                                                              //
передаваемых через стек
                        RM_INT_CALL far *rm_call) // адрес структуры
                                                              // для вызова
прерывания
{
        asm {
                push di
                push bx
                push cx
                mov ax, 0300h      // функция вызова прерывания
                mov bx, int_number
                mov cx, params;
                les di, rm_call          // запись в ES:DI адреса структуры
                int 31h            // вызов сервера DPMI
                jc error
                mov ax, 0               // нормальное завершение
                jmp short rm_int_end
        }
error:  asm mov ax, 0          // завершение с ошибкой
rm_int_end:   asm pop cx
                asm pop bx
                asm pop di
}

// -----------------------------------------------------
// Процедура отображает текущее состояние памяти
// -----------------------------------------------------

int mi_show(void) {

        PMI minfo, far *minfoptr = &minfo;
        unsigned long psize, far *psizeptr=&psize;
        unsigned sel;
        void far *fp;


        get_mi(minfoptr);

        pm_printf(" Информация об использовании памяти\n\r"
                                " ----------------------------------\n\r"
        "\r\n Размер максимального доступного блока:\t\t%ld байт"
```

```
          "\r\n Доступно незафиксированных страниц:\t\t%ld",
          minfo.avail_block,
          minfo.max_page);

          pm_printf("\r\n Доступно зафиксированных страниц:\t\t%ld"
          "\r\n Размер линейного адресного пространства:\t%ld страниц"
          "\r\n Всего имеется незафиксированных страниц:\t%ld",
          minfo.max_locked,
          minfo.linadr_space,
          minfo.total_unlocked);

          pm_printf("\r\n Количество свободных страниц:\t\t\t%ld"
          "\r\n Общее количество физических страниц:\t\t%ld",
          minfo.free_pages,
          minfo.tot_phys_pages);

          pm_printf("\r\n Свободное линейное адресное пространство:\t%ld
страниц"
          "\r\n Размер файла/раздела для страничного обмена:\t%ld страниц",
          minfo.free_linspace,
          minfo.size_fp);

          get_page_size(psizeptr);
          pm_printf("\r\n Размер страницы:\t\t\t\t%ld байт\r\n", psize);

// Выводим текущие значения регистров CS и DS

          asm mov sel,cs
          pm_printf("\n\r CS = %04.4X,   ",sel);
          asm mov sel,ds
          pm_printf("DS = %04.4X",sel);

// Выводим значение текущего приоритетного кольца

          fp = (void far *) main;
          sel = FP_SEG(fp) & 3;
          pm_printf("\n\r Номер приоритетного кольца = %d\n\r",sel);

}

// -----------------------------------------------
// Процедура для получения информации об
// использовании памяти
// -----------------------------------------------

int get_mi(PMI far *minfo) {
        asm {
                  mov ax, 0500h
                  les di, minfo       // ES:DI = адрес структуры DMI
                  int 31h
                  jc error
                  mov ax, 0
                  jmp short get_mi_end
        }
        error: asm mov ax, 1
        get_mi_end:
}

// -----------------------------------------------
// Процедура для получения размера страницы памяти
// -----------------------------------------------

int get_page_size(long far *page_size) {
        asm {
```

```
                mov ax, 0604h
                int 31h
                jc error

                les di, page_size // ES:DI = адрес page_size
                mov es:[di], cx
                mov es:[di+2], bx

                mov ax, 0
                jmp short gps_end
        }
        error: asm mov ax, 1
        gps_end:
}

// ------------------------------------------------
// Определение сегментного адреса видеопамяти
// ------------------------------------------------

unsigned crt_mode, crt_seg;

int video_init(void) {

        union REGS r;

// Определяем текущий видеорежим

        r.h.ah=15;
        int86(0x10,&r,&r);
        crt_mode = r.h.al;

        if(crt_mode == MONO_MODE) crt_seg = 0xb000;
        else if(crt_mode == BW_80_MODE || crt_mode == COLOR_80_MODE)
                crt_seg = 0xb800;
        else {
                printf("\nИзвините, этот видеорежим недопустим.");
                exit(-1);
        }
}

// ------------------------------------------------
// Получение селектора для адресации видеопамяти
// ------------------------------------------------

char far *vid_ptr;
DESCRIPTOR d;
unsigned ldtsel;

int alloc_videosel(void) {

        void far *fp;
        unsigned long addr;

        FP_SEG(vid_ptr) = crt_seg;
        FP_OFF(vid_ptr) = 0;
        pm_printf(" Адрес видеопамяти реального режима:\t %Fp\r\n", vid_ptr);

// Получаем свободный LDT-селектор

        if (! (ldtsel = get_sel())) {
          pm_printf(" Ошибка при получении селектора");
          dos_exit(-1);
        }
        pm_printf(" Получен селектор:\t\t\t%04.4X\n\r", ldtsel);
```

```
// Подготавливаем дескриптор для полученного селектора

        d.limit = 0x2000;
        addr = ABSADDR(crt_seg, 0);
        d.addr_lo = addr & 0xFFFF;
        d.addr_hi = addr >> 16;
        d.access.accessed = 0;      // не использовался
        d.access.read_write = 1;    // разрешены чтение/запись
        d.access.conf_exp = 0;      // не стек
        d.access.code = 0;          // это сегмент данных
        d.access.xsystem = 1;       // не системный дескриптор
        d.access.dpl = 3;                       // приоритетное кольцо 3
        d.access.present = 1;       // сегмент присутствует в памяти
        d.reserved = 0;

// Устанавливаем дескриптор

        if (!set_descriptor(ldtsel, &d)) {
                        pm_printf(" Ошибка при установке дескриптора");
getch();
                        dos_exit(-1);
         }

// Выводим на экран адрес видеопамяти

         FP_SEG(vid_ptr) = ldtsel;
         FP_OFF(vid_ptr) = 0;
         pm_printf(" Адрес видеопамяти защищённого режима:\t%Fp\r\n",
vid_ptr);
}

// -------------------------------------------------
// Освобождение селектора видеопамяти
// -------------------------------------------------

int free_videosel(void) {
         if (!sel_free(ldtsel)) {
                        pm_printf(" Ошибка при освобождении селектора");
                        dos_exit(-1);
         }
}

// ------------------------------------------------
// Получить один селектор в LDT
// ------------------------------------------------

unsigned get_sel(void) {
         asm {
                  mov ax, 0         // получить селектор
                  mov cx, 1         // нужен один селектор
                  int 31h
                  jc error
                  jmp short gs_end // AX содержит новый LDT-селектор
         }
error: asm mov ax, 0     // произошла ошибка
gs_end:
}

// -------------------------------------------------
// Установить дескриптор для LDT-селектора
// -------------------------------------------------

int set_descriptor(unsigned pm_sel, DESCRIPTOR far *desc) {
```

```
         asm {
                  push di
                  push bx
                  mov ax, 000Ch
                  mov bx, pm_sel
                  les di, desc
                  int 31h
                  jc error
                  mov ax, 1
                  jmp short sd_end
          }
error:  asm mov ax, 0
sd_end: asm pop bx
                  asm pop di
}

// ------------------------------------------------
// Освободить LDT-селектор
// ------------------------------------------------

int sel_free(unsigned pmodesel) {
         asm {
                  mov ax, 0001h
                  mov bx, pmodesel
                  int 31h
                  jc error
                  mov ax, 1
                  jmp short done
          }
error: asm mov ax, 0
done:
}

// ------------------------------------------------------
// Вывод символа непосредственной записью в видеопамять
// ------------------------------------------------------

void vi_putch(unsigned int x, unsigned int y ,char c, char attr) {

        register unsigned int offset;
        char far *vid_ptr;

        offset=(y*160) + (x*2);
        vid_ptr=MK_FP(ldtsel, offset);
        *vid_ptr++=c; *vid_ptr=attr;
}
// ------------------------------------------------------
// Вывод строки непосредственной записью в видеопамять
// ------------------------------------------------------

void vi_print(unsigned int x, unsigned int y, char *s, char attr) {
        while(*s) vi_putch(x++, y, *s++, attr);
}
// ------------------------------------------------------
// Вывод сообщения непосредственной записью в видеопамять
// ------------------------------------------------------

void vi_hello_msg(void) {

        vi_print(0, 0,
                " Демонстрация работы с интерфейсом "
                "DPMI                ¦ © Frolov A.V., 1992 ", 0x30);
```

```
}
```

7.2. Драйверы, резидентные программы и WINDOWS

В этом разделе, как и в следующем, мы не будем ничего говорить о защищённом режиме работы процессора. Мы рассмотрим здесь некоторые особенности, которые необходимо учитывать при разработке резидентных программ и драйверов, работающих совместно с WINDOWS.

Очень часто резидентные программы или драйверы перехватывают аппаратное прерывание клавиатуры и отслеживают коды нажимаемых клавиш, выполняя те или иные действия при нажатии заданных комбинаций. Например, драйвер "секретного" диска Norton DISKREET может при нажатии заданной комбинации клавиш блокировать доступ к "секретному" диску, экрану и клавиатуре.

Так как WINDOWS в расширенном режиме использует собственную систему клавиатурного ввода/вывода, основанную на очередях сообщений, а также реализует концепцию виртуальных машин, нажатие активизирующих резидентную программу комбинаций клавиш в неподходящий момент может не привести к желаемому результату и даже стать причиной "зависания" системы.

Есть два возможных решения этой проблемы. Во-первых, можно запретить запуск WINDOWS, если активна резидентная программа или драйвер, не способные работать совместно с WINDOWS. Во-вторых, на время работы WINDOWS можно запретить выполнение резидентной программой или драйвером специфических функций, несовместимых с WINDOWS (например, запретить активизацию резидентной программы при нажатии комбинации клавиш).

Перед запуском WINDOWS и перед её завершением вызываются функции прерывания INT 2Fh 1605h и 1606h соответственно. Ваша резидентная программа или драйвер могут подготовить собственные обработчики для этих прерываний и отслеживать моменты запуска WINDOWS и завершения её работы.

Функция 1605h вызывается при запуске WINDOWS:

Регистры при вызове прерывания:

```
AX      1605h
ES:BX   0000h:0000h
DS:SI   0000h:0000h
CX      0000h
DX      Флаги:
    Бит 0 = 0, если выполняется инициализация WINDOWS в расширенном режиме;
    Бит 0 = 1, если выполняется инициализация DOS-экстендера "Microsoft 286
DOS extender" (используется в стандартном режиме работы WINDOWS);
    Биты 1-15 зарезервированы, их содержимое неопределено.
```

Регистры перед возвратом из прерывания:

```
CX      0000h, если WINDOWS может продолжать инициализацию;
CX <> 0, если запуск WINDOWS недопустим.
```

Функция 1606h вызывается при завершении WINDOWS расширенном или стандартном режиме:

Регистры при вызове прерывания:

```
AX      1606h
DX      Флаги:
    Бит 0 = 0, если выполняется завершение WINDOWS, работавшей в расширенном
режиме;
    Бит 0 = 1, если выполняется завершение DOS-экстендера "Microsoft 286 DOS
extender";
    Биты 1-15 зарезервированы, их содержимое не определено.
```

Обработчик функции 1605h может выполнить необходимые действия, связанные с модификацией алгоритма работы резидентной программы или драйвера, а также при помощи соответствующей установки регистра CX может разрешить или запретить запуск WINDOWS.

Обработчик функции 1606h получает управление при завершении работы WINDOWS и может восстановить прежний алгоритм работы критичной к WINDOWS резидентной программы или драйвера.

Приведённая ниже резидентная программа перехватывает прерывание INT 2Fh и отслеживает фунции 1605h и 1606h, вызавая сообщение и ожидая нажатия на любую клавишу при запуске и завершении работы WINDOWS:

Листинг 22. Контроль запуска WINDOWS
Файл wintsr.asm

```
---------------------------------------------------------------


.MODEL tiny
        .CODE
        .STARTUP

        jmp begin


old_int2Fh_off  dw 0  ; Адрес старого обработчика
old_int2Fh_seg  dw 0  ; прерывания 2Fh

; Сообщение, которое будет выдано на экран
; при запуске WINDOWS

msg_win     db 'WINDOWS Started. Press any key...$'
msg_win_off dw offset msg_win
```

```
; Сообщение, которое будет выдано на экран
; при завершении WINDOWS

msg_win1     db 'WINDOWS Ended. Press any key...$'
msg_winend_off dw offset msg_win1

; Новый обработчик прерывания 2Fh нужен
; для проверки наличия программы в памяти
; при ее запуске для предохранения
; от повторного запуска

new_int2Fh  proc  far
                cmp   ax,0FF00h
                jz    installed

                cmp   ax,1605h
                jz    winstart

                cmp   ax,1606h
                jz    winend
                jmp   dword ptr cs:old_int2Fh_off

winstart:       ; запуск WINDOWS

                push ax
                push bx
                push cx
                push dx
                push ds

                mov   dx,cs:msg_win_off
                mov   ah,9
                push  cs
                pop   ds
                int   21h

                mov   ax,0
                int   16h

                pop ds
                pop dx
                pop cx
                pop bx
                pop ax

                jmp   dword ptr cs:old_int2Fh_off

winend:         ; завершение WINDOWS

                push ax
                push bx
                push cx
                push dx
                push ds

                mov   dx,cs:msg_winend_off
                mov   ah,9
                push  cs
                pop   ds
                int   21h

                mov   ax,0
                int   16h
```

```
                pop ds
                pop dx
                pop cx
                pop bx
                pop ax

                jmp   dword ptr cs:old_int2Fh_off


; Если код функции 0FF00h, то возвращаем
; в регистре AX значение 00FFh. Это признак
; того, что программа уже загружена в память

installed:
                mov   ax,00FFh
                iret

new_int2Fh  endp

;==============================

; Точка входа в программу
; В этом месте начинается выполнение программы

begin proc

; Проверяем, не загружена ли уже программа
; в память

                mov   ax,0FF00h
                int   2Fh

                cmp   ax,00FFh
                jne   first_start

                mov   dx,offset msg_load1
                mov   ah,9
                int   21h

                .EXIT

; Первоначальный запуск программы

first_start:

; Запоминаем адрес старого обработчика прерывания 2Fh

                mov   ax,352Fh
                int   21h
                mov   cs:old_int2Fh_off,bx
                mov   cs:old_int2Fh_seg,es

                push  cs
                pop     ds

; Выводим сообщение

                mov   dx,offset msg_load
                mov     ah,9
                int   21h


                mov   dx,OFFSET new_int2Fh
```

```
                mov   ax,252Fh
                int   21h

; Завершаем программу и оставляем резидентно
; в памяти часть программы, содержащую новые
; обработчики прерываний

                mov   dx,OFFSET begin
                int   27h

begin endp

msg_load      db 'Резидентная программа WINTSR загружена$'
msg_load1     db 'Резидентная программа WINTSR уже загружена$'

                end
```

Следующая резидентная программа работает аналогично, но она запрещает запуск WINDOWS. Попробуйте, запустив предварительно программу NOWINTSR, запустить WINDOWS и посмотрите, что из этого получится.

Листинг 23. Запрет запуска WINDOWS
Файл nowintsr.asm

```
---------------------------------------------------------


                .MODEL tiny
                .CODE
                .STARTUP

                jmp begin


old_int2Fh_off  dw 0  ; Адрес старого обработчика
old_int2Fh_seg  dw 0  ; прерывания 2Fh

; Сообщение, которое выдаётся при запуске WINDOWS

msg_win db 'NOWINTSR несовместима с WINDOWS. Нажмите любую клавишу...$'
msg_win_off dw offset msg_win

; Сообщение, которое выдаётся при завершении WINDOWS

msg_win1     db 10,13,'WINDOWS Ended. Press any key...$'
msg_winend_off dw offset msg_win1

; Новый обработчик прерывания 2Fh нужен
; для проверки наличия программы в памяти
; при ее запуске для предохранения
; от повторного запуска

new_int2Fh  proc  far
                cmp   ax,0FF00h
                jz    installed

                cmp   ax,1605h
                jz    winstart

                cmp   ax,1606h
```

```
                jz    winend
                jmp   dword ptr cs:old_int2Fh_off

winstart:

                push ax
                push bx
                push cx
                push dx
                push ds

                mov   dx,cs:msg_win_off
                mov   ah,9
                push  cs
                pop   ds
                int   21h

                mov   ax,0
                int   16h

                pop ds
                pop dx
                pop cx
                pop bx
                pop ax

                mov  cx,0ffh
                jmp   dword ptr cs:old_int2Fh_off

winend:

                push ax
                push bx
                push cx
                push dx
                push ds

                mov   dx,cs:msg_winend_off
                mov   ah,9
                push  cs
                pop   ds
                int   21h

                mov   ax,0
                int   16h

                pop ds
                pop dx
                pop cx
                pop bx
                pop ax

                jmp   dword ptr cs:old_int2Fh_off


; Если код функции 0FF00h, то возвращаем
; в регистре AX значение 00FFh. Это признак
; того, что программа уже загружена в память

installed:
                mov   ax,00FFh
                iret

new_int2Fh  endp
```

```
;==============================

; Точка входа в программу

begin proc

; Проверяем, не загружена ли уже программа
; в память

                mov   ax,0FF00h
                int   2Fh

                cmp   ax,00FFh
                jne   first_start

                mov   dx,offset msg_load1
                mov   ah,9
                int   21h

                .EXIT

; Первоначальный запуск программы

first_start:

; Запоминаем адрес старого обработчика прерывания 2Fh

                mov   ax,352Fh
                int   21h
                mov   cs:old_int2Fh_off,bx
                mov   cs:old_int2Fh_seg,es

                push  cs
                pop     ds

; Выводим сообщение

                mov   dx,offset msg_load
                mov     ah,9
                int   21h


                mov   dx,OFFSET new_int2Fh
                mov   ax,252Fh
                int   21h

; Завершаем программу и оставляем резидентно
; в памяти часть программы, содержащую новые
; обработчики прерываний

                mov   dx,OFFSET begin
                int   27h

begin endp

msg_load      db 'Резидентная программа NOWINTSR загружена$'
msg_load1     db 'Резидентная программа NOWINTSR уже загружена$'

                end
```

7.3. Связь с WINDOWS CLIPBOARD

Операционная система Microsoft WINDOWS имеет чрезвычайно удобное средство обмена информацией между программами - CLIPBOARD. Это средство предназначено для обмена как текстовой, так и графической информацией. Что имеется в виду под обменом информацией?

Например, вы подготавливаете рекламный проспект при помощи текстового редактора Microsoft Word for WINDOWS. В проспект необходимо поместить фотографию рекламируемого изделия. Используя сканер, вы считываете фотографию и записываете изображение в файл. Далее полученное изображение может быть отредактировано любым графическим редактором. Выделив в графическом редакторе прямоугольный участок изображения, вы можете скопировать его в CLIBBOARD (как во временную память). Затем, переключившись на текстовый редактор, вы можете вставить в любое место текста изображение, взятое из CLIPBOARD.

Вы можете также выделить часть текста и скопировать её в CLIPBOARD. Затем этот текст может быть вставлен в другое место того же текста или вообще в другой текст, редактируемый другим редактором.

Если WINDOWS работает в расширенном режиме, запустив обычную DOS-программу в окне, вы можете выделить любую часть экрана и скопировать её в CLIPBOARD. Затем содержимое CLIPBOARD можно вставить в другую DOS-программу, если она ожидает ввода с клавиатуры. Таким образом организуется перенос текстовой информации из одной DOS-программы в другую DOS-программу. Заметьте, что обе эти программы могут не знать о том, что они работают в среде WINDOWS.

Однако DOS-программа, работающая в среде WINDOWS, может и сама работать с CLIPBOARD. При этом возможен обмен информацией (текстовой или графической) между DOS-программами и приложениями WINDOWS.

Зная интерфейс DOS-программы с WINDOWS CLIPBOARD, вы легко сможете создавать DOS-программы, обменивающиеся информацией с приложениями WINDOWS. Расскажем о некоторых, наиболее полезных функциях, которые могут быть использованы для работы с WINDOWS CLIPBOARD.

Получить версию WinOldAp

Операционная система WINDOWS содержит специальные средства, предназначенные для работы под её управлением DOS-программ. В терминологии WINDOWS DOS-программы относятся к так называемым старым приложениям WINDOWS (WINDOWS Old Application). Версию драйвера WINDOWS, поддерживающего работу с приложениями WinOldAp, можно узнать с помощью функции 1700h прерывания INT 2Fh:

```
Регистры на входе
AX      1700h
Регистры на выходе:
AX      1700h, если данная версия WinOldAp не поддерживает работу с
CLIPBOARD.

Если AX не равно 1700h, то:
        AL = верхнее значение версии (major version);
        AH = нижнее значение версии (minor version).
```

Открыть CLIPBOARD

Перед выполнением любой операции с CLIPBOARD необходимо открыть CLIPBOARD (по аналогии с обычным файлом):

```
Регистры на входе
AX      1701h
Регистры на выходе:
AX      0, если CLIPBOARD уже открыт;

не равно 0, если операция успешно выполнена.
```

Очистить CLIPBOARD

С помощью этой функции можно удалить данные из CLIPBOARD:

```
Регистры на входе
AX      1702h
Регистры на выходе:
AX      0, если при выполнении операции произошла ошибка;

не равно 0, если операция успешно выполнена.
```

Записать данные в CLIPBOARD

С помощью этой функции DOS-программа может выполнить запись данных в WINDOWS CLIPBOARD.

```
Регистры на входе
AX      1703h
DX      Формат данных, записываемых в CLIPBOARD:
      01h текст;
      02h графика в формате bitmap;
      03h графика в формате metafile picture;
      04h SYLK;
      05h DIF;
      06h графика в формате TIFF;
      07h текст в кодировке OEM.
ES:BX   Указатель на записываемые данные
SI:CX   Длина записываемых данных
Регистры на выходе:
AX      0, если при выполнении операции произошла ошибка;

не равно 0, если операция успешно выполнена.
```

С помощью этой функции можно записывать как текстовые, так и графические данные.

Операционная система WINDOWS использует отличную от принятой в DOS кодировку символов. Кодировка WINDOWS называется ANSI-кодировкой, кодировка DOS - OEM-кодировкой. Если при записи текстовых данных в CLIPBOARD вы зададите кодировку OEM (записав в регистр DX значение 7), одновременно с записью данных будет автоматически выполняться перекодировка из OEM в ANSI.

Пользуясь следующей таблицей, вы можете записывать в CLIPBOARD графические данные в формате bitmap:

Таблица 15. Формат CLIPBOARD для BITMAP-файлов.

Смещение, размер	Описание
00h (2)	тип(0000h)
02h (2)	ширина bitmap в пикселах
04h (2)	высота bitmap в пикселах
06h (2)	количество байт на строку
08h (1)	количество цветовых планов
09h (1)	количество цветовых битов в пикселе
0Ah (4)	указатель на начало данных
0Eh (2)	ширина в 0.1 mm
10h (2)	высота в 0.1 mm
12h	графические данные

Исчерпывающую информацию о форматах графических файлов WINDOWS вы можете получить из документации, поставляемой Microsoft для разработчиков приложений.

Получить размер CLIPBOARD

Размер данных, записанных в CLIPBOARD, можно узнать с помощью следующей функции:

```
Регистры на входе
AX      1704h
DX      Формат данных:
      01h текст;
      02h графика в формате bitmap;
      03h графика в формате metafile picture;
      04h SYLK;
      05h DIF;
      06h графика в формате TIFF;
      07h текст в кодировке OEM.
Регистры на выходе:
DX:AX   0, если CLIPBOARD не содержит данных в указанном формате;

размер записанных данных, включая заголовки.
```

Прочитать данные из CLIPBOARD

```
Регистры на входе
AX      1705h
DX      Формат данных, читаемых из CLIPBOARD:
      01h текст;
      02h графика в формате bitmap;
      03h графика в формате metafile picture;
      04h SYLK;
      05h DIF;
      06h графика в формате TIFF;
      07h текст в кодировке OEM.
ES:BX   Указатель на буфер для читаемых данных
Регистры на выходе:
AX      0, если при выполнении операции произошла ошибка;

не равно 0, если операция успешно выполнена.
```

Закрыть CLIPBOARD

После выполнения записи необходимо закрыть CLIPBOARD (точно также, как вы закрываете файл). Для того, чтобы закрыть CLIPBOARD, вы можете использовать следующую функцию:

```
Регистры на входе
AX      1708h
Регистры на выходе:
AX      0, если произошла ошибка;

не равно 0, если операция успешно выполнена.
```

Установить размер данных, записанных в CLIPBOARD

После записи данных в CLIPBOARD программист может ограничить размер CLIPBOARD:

```
Регистры на входе
AX      1709h
SI:CX   Размер данных в байтах
Регистры на выходе:
DX:AX   Размер максимального доступного участка памяти
```

Критическая секция

DOS-программа, работающая на виртуальной машине WINDOWS, может временно запретить переключение задач, захватив процессор в монопольное пользование. Для этого она должна вызвать функцию 1681h прерывания INT 2Fh. Параметры задавать не надо.

Про программу, захватившую процессор, говорят, что она вошла в критическую секцию.

Для выхода из критической секции и возобновления работы диспетчера задач WINDOWS программа должна вызвать функцию 1682h прерывания INT 2Fh.

Пример программы для работы с CLIPBOARD

Приведённая ниже программа демонстрирует запись в CLIPBOARD из DOS-программы, а также вход в критическую секцию и выход из неё.

Вначале программа убеждается в том, что она запущена под управлением WINDOWS, работающем в расширенном режиме. Только в этом случае доступны функции для работы с CLIPBOARD.

Далее программа демонстрирует блокировку механизма переключения задач при входе в критическую секцию. После этого проверяется доступность CLIPBOARD.

Если CLIPBOARD доступен, программа проверяет, есть ли в нём текстовые данные. Если текстовые данные есть, они читаются из CLIPBOARD и выводятся на экран. Затем CLIPBOARD очищается и в него записывается тестовая строка, состоящая из латинских букв и символов кириллицы (для проверки выполнения перекодировки из OEM в ANSI).

После записи строки программа устанавливает размер CLIPBOARD и закрывает его. Далее вы можете запустить приложение WINDOWS "Clipboard" и посмотреть результат!

Листинг 24. Работа с WINDOWS CLIPBOARD и критической секцией
Файл windos.c

```
------------------------------------------------------------


#include <dos.h>
#include <stdio.h>
#include <malloc.h>

char buf[2048], far *fptr = buf;
char msg[] = "String for storing(для записи) to WINDOWS clipboard\n\n\n";

void main(void) {

        union REGS inregs, outregs;
        struct SREGS segregs;
        unsigned long clipbrd_size, i;

        printf("\n\nРабота с WINDOWS CLIPBOARD и критической секцией\n"
                        "© Frolov A. 1992\n\n");

// Проверяем, работает ли программа под управлением
// WINDOWS в расширенном режиме.

         inregs.x.ax = 0x1600;
         int86( 0x2f, &inregs, &outregs);

         if (outregs.h.al == 0) {
```

```
                printf("\nТребуется расширенный режим WINDOWS!\n");
                exit(-1);
        }

// Выводим на экран версию WINDOWS

        printf("Версия WINDOWS - %d.%d\n",
                outregs.h.al, outregs.h.ah);

// Определяем и выводим на экран идентификатор
// виртуальной машины, на которой работает
// данная программа.

        inregs.x.ax = 0x1683;
        int86( 0x2f, &inregs, &outregs);
        printf("Виртуальная машина - VM%d\n",
                outregs.x.bx);

// Входим в критическую секцию. До выхода из нее
// переключение задач в WINDOWS блокировано.

        inregs.x.ax = 0x1681;
        int86( 0x2f, &inregs, &outregs);

        printf("\n\nВошли в критическую секцию.\n"
                        "Попробуйте переключить задачу клавишами <ALT-
TAB>,\n"
                        "затем нажмите любую другую клавишу для выхода\n"
                        "из критической секции\n");

// После нажатия на любую клавишу выходим
// из критической секции

        getch();

        inregs.x.ax = 0x1682;
        int86( 0x2f, &inregs, &outregs);

        printf("Вышли из критической секции\n");

// Проверяем доступность CLIPBOARD. Если доступен,
// выводим версию драйвера WINDOWS, использующегося
// для поддержки DOS-программ - WINOLDAP.

        inregs.x.ax = 0x1700;
        int86( 0x2f, &inregs, &outregs);
        if(outregs.x.ax == 0x1700) {
                printf("\nClipboard недоступна");
                exit(-1);
        }
        else printf("\nВерсия WINOLDAP - %d.%d",
                outregs.h.al, outregs.h.ah);

// Открываем CLIPBOARD

        inregs.x.ax = 0x1701;
        int86( 0x2f, &inregs, &outregs);
        if(outregs.x.ax == 0x0000) {
                printf("\nОшибка при открытии Clipboard");
                exit(-1);
        }

// Получаем объем данных, находящихся в CLIPBOARD.
// Регистр DX равен 1, следовательно, мы будем работать с
```

```
// текстовыми данными.

        inregs.x.ax = 0x1704;
        inregs.x.dx = 0x01;
        int86( 0x2f, &inregs, &outregs);

// Вычисляем объем данных

        clipbrd_size = outregs.x.dx << 16l;
        clipbrd_size += outregs.x.ax;
        if(clipbrd_size == 0L)
               printf("\nClipboard пуст");

        else printf("\nОбъем данных в Clipboard: %lu\n", clipbrd_size);

// Получаем данные из CLIPBOARD. В регистре DX
// задаем значение 7, что соответствует тексту
// в кодировке OEM. При этом в процессе передачи
// данных выполняется перекодировка из представления
// ANSI (используется в WINDOWS) в представление
// OEM (используется в DOS)

        inregs.x.ax = 0x1705;
        inregs.x.dx = 0x07;
        segregs.es  = FP_SEG(fptr);
        inregs.x.bx = FP_OFF(fptr);

        int86x( 0x2f, &inregs, &outregs, &segregs);

// Выводим содержимое CLIPBOARD, если
// там что-нибудь есть.

        if(outregs.x.ax == 0) printf("\nВ Clipboard ничего нет!");
        else {
               printf("Содержимое Clipborad:\n");

               for(i=0l; i < clipbrd_size; i++) {
                       putch(buf[i]);
               }
        }

// Очищаем CLIPBOARD

        inregs.x.ax = 0x1702;
        int86( 0x2f, &inregs, &outregs);

// Записываем в CLIPBOARD текстовые данные
// в кодировке OEM

        inregs.x.ax = 0x1703;
        inregs.x.dx = 0x07;
        inregs.x.si = 0x00;
        inregs.x.cx = strlen(msg);

        fptr = msg;
        segregs.es  = FP_SEG(fptr);
        inregs.x.bx = FP_OFF(fptr);

        int86x( 0x2f, &inregs, &outregs, &segregs);
        if(outregs.x.ax == 0) {
               printf("\nОшибка при записи в Clipboard");
               exit(-1);
        }
```

```
// Устанавливаем размер CLIPBOARD, равный
// длине записанной в него строки

        inregs.x.ax = 0x1709;
        inregs.x.si = 0x00;
        inregs.x.cx = strlen(msg);
        int86( 0x2f, &inregs, &outregs);

// Закрываем CLIPBOARD

        inregs.x.ax = 0x1708;
        int86( 0x2f, &inregs, &outregs);

        exit(0);
}
```