Реентерабельность и проблемы резидентных программ в DOS

С понятием прерывания тесно связано такое свойство программ (модулей, функций) как **РЕЕНТЕРАБЕЛЬНОСТЬ** (буквально, " повторновходимость"). Предположим, что имеется функция, которая вызывается как из функции обработки прерываний, так и из основного (прерываемого) процесса.

```
void fun() {}

void interrupt intr(...) { ... fun(); ...}

void main() { ... fun(); ...}
```

Тогда вполне вероятна ситуация, что во время вызова функции **fun** из основной программы может произойти прерывание, и функция **fun** будет вызвана еще раз уже из обработчика прерываний, то есть вызов ее произойдет " как бы" рекурсивно. Для обозначения работоспособности программы в такой ситуации предусмотрен специальный термин - **реентерабельность.**

> РЕЕНТЕРАБЕЛЬНОСТЬ - - свойство программы, корректно выполняться при рекурсивном вызове из прерывания

С точки зрения механизма обработки прерываний, введенного для Си-программ, который обеспечивает прозрачность прерывания, а также возможности рекурсивного вызова любых функций, можно утверждать, что любая функция, которая не использует глобальных переменных или ресурсов, является реентерабельной, то есть ее смело можно вызывать из прерывания. Другое дело, что при использовании глобальных (общих) данных или разделении общих ресурсов могут возникнуть проблемы - одна и та же переменная может использоваться как из основной программы, так и из прерывания. При проектировании все эти случаи должны отслеживаться и процесс обработки данных, инициируемый из прерывания, должен " откладываться" до завершения аналогичных действий в основной программе. Все эти проблемы лучше всего рассмотреть на примере обеспечения реентерабельности функций DOS.

> Операционная система DOS -- нереентерабельная программа, поэтому для вызова его функций из резидентных программ необходимо использование специальных средств синхронизации (проверки " занятости" DOS).

Заметим, что DOS была спроектирована исключительно как однозадачная операционная система, вернее, понятие задачи или процесса в ней вообще не предусмотрено. Это значит, что при выполнении конкретной программы в DOS ее текущее состояние и характеристики занимаемых ресурсов " размазаны" по различным переменным. Эти разрозненные переменные и составляют **контекст** текущей выполняемой программы (задачи). В то же время резидентная программа, хотя и вызывается по прерыванию, при выполнении функций DOS, должна быть заявлена как отдельный процесс (задача). Самым естественным для нее является наследование состояния собственного основного процесса (**main**) перед его завершением. Тогда в резидентную программу должны быть включены средства переключения задач, создающих более или менее полный режим мультипрограммирования, и она должна самостоятельно:

- в функции main() определить ряд ячеек сегмента данных DOS, которые содержат параметры текущей выполняемой программы (указатель на PSP, указатель на DTA, индикатор занятости DOS и др.);

- определить значения перечисленных параметров для собственной программы;

- при необходимости организовать собственную область стека в программе;

- перед выполнением функции DOS в программе обработки прерывания необходимо убедиться в том, что DOS не занята обработкой программного прерывания от текущей выполняемой (прерванной) программы. Если это так, то необходимо отложить вызов до появления программного прерывания 28h (DOS свободна) или проверять занятость DOS по таймеру. Аналогичные действия необходимы также для ROM BIOS;

- если DOS свободна, то перед вызовом необходимо сохранить параметры текущей (прерванной) программы и установить соответствующие параметры резидентной. Таким образом "вручную" сделать резидентную программу текущей. После выполнения функции DOS необходимо установить в DOS состояние прерванной программы;

- аналогичные действия необходимо выполнять и со стеком.

Приведенный ниже пример демонстрирует реальные трудности оформления процесса обработки прерывания в виде полноправной задачи в DOS.

```
 //-----------------------------------------------------bk63-02.cpp
#include <stdio.h>
#include <io.h>
#include <dos.h>
#include <alloc.h>
#include <string.h>
                                  // Макроопределения переключения стека
#define SAVE if (LEVEL++ == 0) {CS=cs; SSI=_SS;SPI=_SP;
_SS=SS;_SP=(unsigned)SP;}
#define RESTORE if (LEVEL-- == 1) {CS=0; _SS=SSI,_SP=SPI;}


char fname[]="a.txt";
unsigned CtrlBrk;
int NKEY;                                    // Скан-код горячей клавиши
int DS=0;                  // Значение DS в момент выполнения программы
int NSTACK=1000;                             // Размер внутреннего стека
char *SP;                                  // Указатель внутреннего стека
int FORKH=0;                                 // Флаг выполнения процесса
struct SREGS SG;                        // Значения сегментных регистров.
                                         // Используются для определения
                                  // размера резидентной части программы.
```

```
union REGS rg;                          // Значения регистров процессора.
                                       // Используются при инициализации.

unsigned CS,SS,SPI,SSI,LEVEL=0,PIDS,PSP,PSPI,DOSSEG,DOSBUSY,DISKFLAG;

unsigned far *PSPADR;                // far адрес PSP сетевого резидента
char far *PDOSBUSY;                        // Адрес флага занятости DOS
char far *DTA, far *DTAI;                // Адрес DTA сетевого резидента
                                        // и DTA прерванного процесса
unsigned SIZE;                      // Размер резидентной части программы
int WAITDOS=0,                                  // Флаги занятости: DOS,
WAITDISK=0,                                      // дисковой подсистемы,
WAITBIOS=0;                                                  // BIOS.

                        //Сохраненные значения перехваченных прерываний:
void interrupt (*TIMSAV)(...),                        // Таймер (INT 8h),
interrupt (*KBSAV)(...),                         // Клавиатура (INT 9h),
interrupt (*DOSOKSAV)(...),           // Прерывание DOS O.K. (INT 28h),
interrupt (*DISKSAV)(...);            // Дисковое прерывание (INT 13h),

 //-----------Вывод данных в видеопамять------------------------------
void putstr(char *s, int x, int y)
{
char (far *p)[25][80][2]=(char (far*)[25][80][2])0xB8000000;
while(*s !='\0')
     { (*p)[y][x ][0] = *s++; (*p)[y][x++][1] = 0x71; }
}

char *to10(int n,char *s)
{ char *p = s+2;
for (; p>=s; p--, n/=10) *s-- = n % 10;
s+=3; *s++ = ' '; return s;
}
```

```
                  // Сохранение регистра флагов в точке вызова newscrit
int cflag;
void interrupt newscrit(bp,di,si,ds,es,dx,cx,bx,ax,ip,cs,flgs)
{ ax=0; cflag=flgs; }

static void interrupt (*old24)(...);

void SETCONTEXT()                // Переключение контекста с прерванной
{                                          // задачи на сетевой резидент
PSPI=*PSPADR;                       // Запоминаем идентификатор процесса
                                          // (PSP прерванного процесса).
*PSPADR=PSP;                                // Устанавливаем свой PSP
DTAI=getdta();                   // Запоминаем DTA прерванного процесса.
setdta(DTA);                                // Устанавливаем свой DTA
old24=getvect(0x24);     // Запоминаем 24h вектор (адрес подпрограммы )
                                          // реакции на фатальную ошибку)
setvect(0x24,(void interrupt(far*)(...))newscrit);
                                                    // Ставим свой вектор
rg.x.ax=0x3300;                     // Проверить реакцию на CTRL/BREAK,
intdos(&rg,&rg);                                                    //
CtrlBrk=rg.h.dl;                            // и запомнить эту реакцию
rg.x.ax=0x3301;                     // Установить реакцию на CTRL/BREAK:
rg.h.dl=0;              // запретить остановку программы по CTRL/BREAK.
intdos(&rg,&rg);                                                    //
}

void RESTORECONTEXT()      //Восстановление контекста прерванной задачи
{
setdta(DTAI);                   //Восстановить DTA прерванной программы,
*PSPADR=PSPI;                                                 //PSP,
setvect(0x24,old24);        //вектор 24h (реакция на фатальную ошибку),
rg.x.ax=0x3301;                   //восстановить реакцию на CTRL/BREAK.
```

```
rg.h.dl=CtrlBrk;
intdos(&rg,&rg);
}

void PROCESS1()                        // Пример процесса, создающего файл
{                                                          // средствами DOS
int fd;
if((fd=_creat(fname,0)) == -1) return;
fname[0]++;
char *p="Это строка из резидента\n\r";
_write(fd,(void*)p,strlen(p+1));
_close(fd);
}

void PROCKEY()                                  // Обработка горячих клавиш
{                                    // с проверкой всех возможных условий
if (NKEY==0) return;                     // Нет кода - нечего обрабатывать
if (FORKH) return;    // Исключение повторного входа (с потеря запроса)
if (WAITDOS !=-1)                              // Проверка на занятость DOS
     {
     if (*PDOSBUSY)            //Если установлен флаг занятости DOS ...
          { putstr("Wait DOS ",0,1);
          WAITDOS=1;                       // Взводим свой флаг и выходим
          return;
          }
     }
if (WAITDISK !=-1)        // Проверка занятости диска по прерыванию 13h
     {
     if (DISKFLAG)               // Если дисковая подсистема занята ...
          { putstr("Wait DISK",20,1);
          WAITDISK=1;                      // Взводим свой флаг и выходим
          return;
          }
```

```
     }
if (WAITBIOS !=-1)              // Проверка занятости BIOS - прерывание
     {                            // основного процесса в сегменте BIOS
     if ((CS & 0xFF00) >= 0xC000)
          {                                 // Если был прерван BIOS ...
          putstr("Wait BIOS",40,1);
          WAITBIOS=1;                         // Взводим флаг и выходим
          return;
          }
     }
FORKH++;                           // Взводим флаг выполнения процесса
SETCONTEXT();                      // Переключаемся на контекст процесса
switch(NKEY)                      // обработчика прерывания, оставленного
     {                              // от main() резидентной программы
case 53: PROCESS1();
     break;                                          // Реакция на ALT+?
     }
RESTORECONTEXT();                           // Восстанавливаем контекст
NKEY=0;                     // Сбрасываем скан-код - обработка выполнена
FORKH--;                          // Сбрасываем флаг выполнения процесса
}



int i;
#define ALT 8
int kbval;
char SCAN[]={53,80,72,75,77,71,78,74,82,79,0};

                                  // Обработчик прерываний от клавиатуры
void interrupt KB(bp,di,si,ds,es,dx,cx,bx,ax,ip,cs,flgs)
{
SAVE;                                                 // Переключить стек
kbval=peekb(0,0x417);               // Байт состояния клавиатуры из DOS
```

```
for (i=0; SCAN[i] !=0; i++)                          // Проверка скан-кода
if (inportb(0x60)==SCAN[i]) break;
if (SCAN[i] !=0)
     {                         // Это скан-код из списка при нажатой ALT
     if ((kbval & ALT) !=0)       // Снять код с клавиатуры - он " наш"
          {
          kbval=inportb(0x61);
          outportb(0x61,kbval | 0x80);
          outportb(0x61,kbval);
          outportb(0x20,0x20);
          switch(SCAN[i])
               {
          case 71: break;          // Моментальная реакция на скан-коды
          case 80: break;                 // без переключения процессов
          case 72: break;
          case 75: break;
          case 77: break;
          case 78: break;
          case 74: break;
          case 53: if (NKEY!=0) break;  // Старый код еще не обработан -
                                              // выйти с потерей запроса
                                    // Иначе - обработка с переключением
               NKEY=SCAN[i]; PROCKEY(); break;
               }
          RESTORE;
          return;
          }
     }
RESTORE;                                             // Восстановить стек
(*KBSAV)();                       // Обработчик прерывания DOS по цепочке
}
```

```
 //---------------------------------------------------------------------------
void interrupt DOSOK(bp,di,si,ds,es,dx,cx,bx,ax,ip,cs,flgs)
               // Прерывание DOS O.K., которое вызывается DOS-ом, когда
               // разрешено использование всех функций DOS, несмотря на
                 // взведенный флаг занятости DOS (т.е. DOS находится в
               // состоянии ожидания, например при вводе с клавиатуры).
{
(*DOSOKSAV)();                     //Вызвать по старому вектору DOS O.K.
if((*PDOSBUSY==0) && (WAITDOS> 0))
                           //Если флаг занятости DOS сброшен, а наш флаг
     {                            // взведен, то сбрасываем свой флаг...
     putstr(" ",0,1);
     WAITDOS=0;
     SAVE; PROCKEY(); RESTORE;
     }
}
 //---------------------------------------------------------------------------
void interrupt BIOSDISK(bp,di,si,ds,es,dx,cx,bx,ax,ip,cs,flgs)
{                           //Обработка перехваченного прерывания INT 13h
DISKFLAG++;              //Взводим признак занятости дисковой подсистемы
(*DISKSAV)();                   //Вызываем прерывание по старому вектору
newscrit();                                          //Сохраняем флаги ...
ax=_AX;                                               //... и регистры ...
cx=_CX;
dx=_DX;
flgs=cflag;                                        //... возвращаем флаги.
DISKFLAG --;         //Сбрасываем признак занятости дисковой подсистемы.
if (WAITDISK> 0)             //Если наш флаг взведен, то сбрасываем его,
     {
     WAITDISK=0;
     putstr(" ",20,1);
     SAVE; PROCKEY(); RESTORE;
     }
```

```
}
 //---------------------------------------------------------------------------
void interrupt TIMER(bp,di,si,ds,es,dx,cx,bx,ax,ip,cs,flgs)
{                    //Обработка перехваченного прерывания от таймера.
if (LEVEL==0)
     {
     SAVE;                          //Переключиться на внутренний стек
     if (WAITBIOS> 0)  //Если установлен флаг занятости BIOS, то сбрасываем его
          {                 // и пытаемся выполнить отложенные действия
          putstr(" ",40,1);
          WAITBIOS=0;
          PROCKEY();
          }
     if ((WAITDOS> 0) && (*PDOSBUSY==0))
                              //Если взведен наш флаг занятости DOS, а
          {                 // системный флаг занятости DOS сброшен, то
          putstr(" ",0,1);
          WAITDOS=0;                          // сбрасываем свой флаг
          PROCKEY();        // и пытаемся выполнить отложенные действия
          }
     RESTORE;                          //Восстанавливаем внешний стек.
     }
(*TIMSAV)();    //Вызвать сохраненный обработчик прерывания от таймера.
}
 //---------------------------------------------------------------------------
void main()
{ int i; unsigned n , far *b; char sx[3];
if ((SP = new char[NSTACK])==NULL) goto fatal;
printf("Синхронизация DOS(y/n):"); scanf("%s",sx);
if (sx[0]!='y') WAITDOS=-1;
printf("Синхронизация DISK(y/n):"); scanf("%s",sx);
if (sx[0]!='y') WAITDISK=-1;
printf("Синхронизация BIOS(y/n):"); scanf("%s",sx);
```

```
if (sx[0]!='y') WAITBIOS=-1;

SP+=NSTACK-10;        //Вычисляем значение указателя внутреннего стека.
DS=SS=_DS;                    //Запоминаем значения сегментных регистров.
SIZE=(DS-_CS+0x10)+((unsigned)(malloc(200)) >> 4);
                                              //Вычисляем размер резидента
printf("Объем резидента %d KB\n\r",SIZE >> 6);
rg.x.ax=0x5100;                      //Получить у DOS адрес PSP программы
intdos(&rg,&rg);
PSP=rg.x.bx;
rg.x.ax=0x3400;              //Спросить у DOS адрес флага занятости DOS,
intdos(&rg,&rg);                           //который лежит в сегменте DOS.
DOSSEG=_ES;                     //Попутно запомнить адрес этого сегмента
DOSBUSY=rg.x.bx;                                    //и адрес этого флага
PDOSBUSY=(char far*)MK_FP(DOSSEG,DOSBUSY);
                           //Собрать far указатели на флаг занятости DOS,
PSPADR=(unsigned far*)MK_FP(DOSSEG,0);
                                                 // и свой собственный PSP
while(1)                     //В этом цикле находим в сегменте DOS адрес
     {                         // идентификатора процесса (проще говоря,
                         // где DOS хранит адрес PSP текущего процесса).
     if (*PSPADR==PSP)
          {
          rg.x.ax=0x5000;
          rg.x.bx=PSP+1;
          intdos(&rg,&rg);
          i=(*PSPADR==(PSP+1));
          rg.x.ax=0x5000;
          rg.x.bx=PSP;
          intdos(&rg,&rg);
          if(i) break;
          }
     PSPADR++;
```

```
     }
DTA=getdta();                    //Читаем и запоминаем адрес своего DTA
KBSAV=getvect(0x9);               //Переустанавливаем вектор прерывания
setvect(0x9,(void interrupt (far*)(...))KB);
TIMSAV=getvect(0x1C);
setvect(0x1C,(void interrupt (far*)(...))TIMER);
DOSOKSAV=getvect(0x28);
setvect(0x28,(void interrupt (far*)(...))DOSOK );
DISKSAV=getvect(0x13);
setvect(0x13,(void interrupt (far*)(...))BIOSDISK);
b=(unsigned far*)MK_FP(_psp,0x2C);
_ES=*b;                                //Освободить в памяти ENVIROMENT.
_AH=0x49;
geninterrupt(0x21);
keep(0,SIZE);                          // Выйти и остаться резидентным.
fatal: printf("Не хватает памяти\n");
}                   // А если были ошибки, то резидентным не оставаться
```