

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №5**  
**по дисциплине «Операционные системы»**  
Тема: Сопряжение стандартного и пользовательского обработчика  
прерываний

Студент гр. 9383

Нистратов Д.Г.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

## **Постановка задачи.**

Исследовать возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Написать пользовательский обработчик прерывания, который получает управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре и обрабатывает скан-код, осуществляя определенные действия, если скан-код совпадает с определенными кодами, которые он должен обрабатывать. Если скан-код не совпадает с этими кодами, то управление передается стандартному прерыванию.

## **Последовательность действий**

**Шаг 1.** Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет такие же функции, как в программе ЛР 4, а именно:

1) Проверяет, установлено ли пользовательское прерывание с вектором 09h.

2) Если прерывание не установлено то, устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний. Адрес точки входа в стандартный обработчик прерывания находится в теле пользовательского обработчика. Осуществляется выход по функции 4Ch прерывания int 21h.

3) Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h.

Выгрузка прерывания по соответствующему значению параметра в командной строке /un. Выгрузка прерывания состоит в восстановлении стандартного вектора прерываний и освобождении памяти, занимаемой резидентом. Затем осуществляется выход по функции 4Ch прерывания int 21h.

Для того чтобы проверить установку прерывания, можно поступить следующим образом. Прочитать адрес, записанный в векторе прерывания. Предположим, что этот адрес указывает на точку входа в установленный резидент. На определенном, известном смещении в теле резидента

располагается сигнатура, некоторый код, который идентифицирует резидент. Сравнив известное значение сигнатуры с реальным кодом, находящимся в резиденте, можно определить, установлен ли резидент. Если значения совпадают, то резидент установлен. Длину кода сигнатуры должна быть достаточной, чтобы сделать случайное совпадение маловероятным.

Программа должна содержать код устанавливаемого прерывания в виде удаленной процедуры. Этот код будет работать после установки при возникновении прерывания. Он должен выполнять следующие функции:

- 1) Сохранить значения регистров в стеке при входе и восстановить их при выходе.
- 2) При выполнении тела процедуры анализируется скан-код.
- 3) Если этот код совпадает с одним из заданных, то требуемый код записывается в буфер клавиатуры.
- 4) Если этот код не совпадает ни с одним из заданных, то осуществляется передача управления стандартному обработчику прерывания.

**Шаг 2.** Запустите отлаженную программу и убедитесь, что резидентный обработчик прерывания 09h установлен. Работа прерывания проверяется введением различных символов, обрабатываемых установленным обработчиком и стандартным обработчиком.

**Шаг 3.** Также необходимо проверить размещение прерывания в памяти. Для этого запустите программу ЛР 3, которая отображает карту памяти в виде списка блоков МСВ. Полученные результаты поместите в отчет.

**Шаг 4.** Запустите отлаженную программу еще раз и убедитесь, что программа определяет установленный обработчик прерываний. Полученные результаты поместите в отчет.

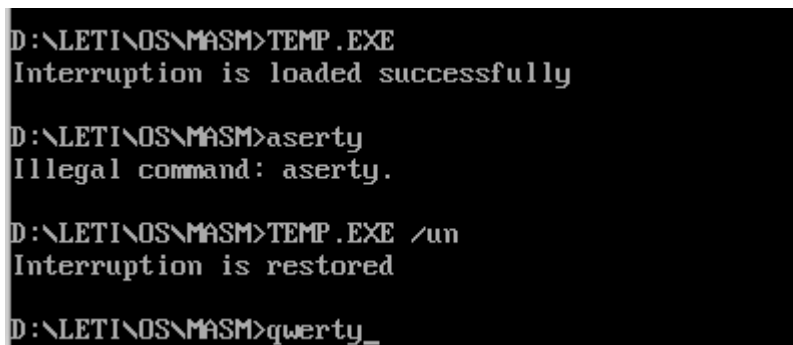
**Шаг 5.** Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

**Шаг 6.** Ответьте на контрольные вопросы.

### **Выполнение работы.**

**Шаг 1.** Была описана программа, выполняющая проверку установление пользовательского прерывания с вектором 09h. А также изменяющие символы q и w на a и s.

**Шаг 2.** Была отлажена и запущена написанная программа, см. Изображение 1



```
D:\LETI\OS\MASM>TEMP.EXE
Interruption is loaded successfully

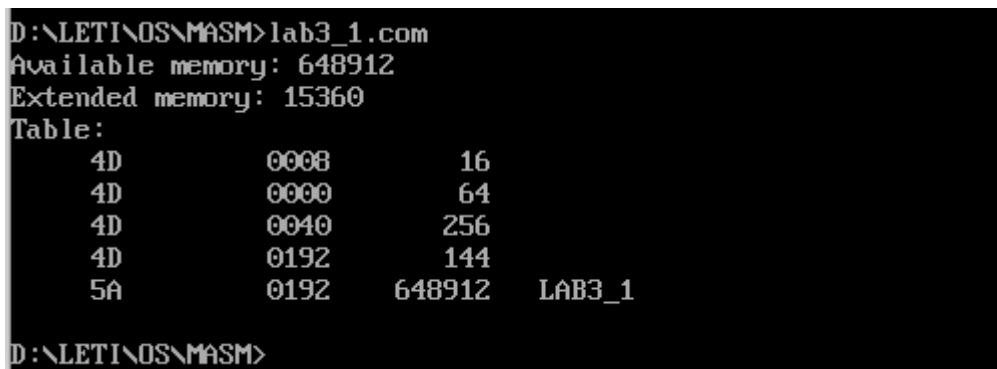
D:\LETI\OS\MASM>aserty
Illegal command: aserty.

D:\LETI\OS\MASM>TEMP.EXE /un
Interruption is restored

D:\LETI\OS\MASM>qwerty_
```

Изображение 1 – Пример работы программы.

**Шаг 3.** Запустим программу из ЛР3 для отображения карты памяти в виде списка блоков MCB, см. Изображение 2



```
D:\LETI\OS\MASM>lab3_1.com
Available memory: 648912
Extended memory: 15360
Table:
  4D      0008      16
  4D      0000      64
  4D      0040     256
  4D      0192     144
  5A      0192   648912   LAB3_1

D:\LETI\OS\MASM>
```

Изображение 2 – таблица MCB из ЛБ3

**Шаг 4.** Запустим программу и убедимся, что установщик прерываний работает, см. Изображение 3

```
D:\LETI\OS\MASM>lab3_1.com
Available memory: 644368
Extended memory: 15360
Table:
  4D      0008      16
  4D      0000      64
  4D      0040     256
  4D      0192     144
  4D      0192    4368   LAB5
  4D      02AE    4144
  5A      02AE   644368  LAB3_1
D:\LETI\OS\MASM>
```

Изображение 3 – Установщик прерываний в таблице MCB

**Шаг 5.** Запустим программу и убедимся, что выгрузка произошла успешно, см.

Изображение 4

```
D:\LETI\OS\MASM>LAB5.EXE /un
Interruption was unloaded.

D:\LETI\OS\MASM>lab3_1.com
Available memory: 648912
Extended memory: 15360
Table:
  4D      0008      16
  4D      0000      64
  4D      0040     256
  4D      0192     144
  5A      0192   648912  LAB3_1
D:\LETI\OS\MASM>
```

Изображение 4 – Выгрузка установщика прерываний

**Шаг 6.** Ответы на вопросы:

1. Какого типа прерывания использовались в работе?

Были использованы аппаратные прерывания 09h и 16h, а также пользовательские прерывания 21h и 10h.

2. Чем отличается скан-код от кода ASCII

Скан-код – это код клавиши клавиатуры, который преобразуется обработчиком в код символа. ASCII код – это код самого символа в таблице.

### **Заключение.**

В ходе лабораторной работы был изучен метод взаимодействия клавиатуры с ОС DOS, изучен принцип работы скан-кодов, а также реализация собственных резидентных прерываний.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab5.asm

ASTACK SEGMENT STACK

DW 200 DUP(?)

ASTACK ENDS

DATA SEGMENT

ROUT\_LOADED db 'Interruption is already loaded', 0DH, 0AH, '\$'

ROUT\_CHANGED db 'Interruption is loaded successfully', 0DH, 0AH, '\$'

ROUT\_IS\_NOT\_LOADED db 'Interruption is not loaded', 0DH, 0AH, '\$'

ROUT\_UNLOADED db 'Interruption is restored', 0DH, 0AH, '\$'

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA, SS:ASTACK

WRITE PROC NEAR

push ax

mov ah, 9

int 21h

pop ax

ret

WRITE ENDP

start:

ROUT proc far

jmp START\_ST

KEEP\_PSP DW 0

KEEP\_IP DW 0

KEEP\_CS DW 0

KEEP\_SS DW 0

KEEP\_SP DW 0

KEEP\_AX DW 0

INT\_SIG DW 7777h

INTSEG DW 64 DUP(?)

START\_ST:

```
mov KEEP_SP, sp
mov KEEP_AX, ax
mov KEEP_SS, ss
```

```
mov ax, seg INTSEG
mov ss, ax
mov ax, offset START_ST
mov sp, ax
```

```
mov ax, KEEP_AX
```

```
push bx
push cx
push dx
push si
push cx
push ds
push ax
```

```
in al, 60h
cmp al, 10h
je K_Q
cmp al, 11h
je K_W
```

```
call dword ptr cs:[KEEP_IP]
jmp END_ROUT_P
```

K\_Q:

```
mov al, 'a'
jmp DO
```

K\_W:

```
mov al, 's'
```

DO:

```
push ax
in al, 61h
mov ah, al
```



```
or al, 80h
out 61h, al
xchg ah, al
out 61h, al
mov al, 20H
out 20h, al
pop ax
```

READS:

```
mov ah, 05h
mov cl, al
mov ch, 00h
int 16h
or al, al
jz END_ROUT_P
mov ax, 40h
mov es, ax
mov ax, es:[1ah]
mov es:[1ch], ax
jmp READS
```

END\_ROUT\_P:

```
pop ds
pop es
pop si
pop dx
pop cx
pop bx
pop ax
mov sp, KEEP_SP
mov ax, KEEP_SS
mov ss, ax
mov ax, KEEP_AX
mov al, 20h
out 20h, al
iret
```

ROUT endp

CHECK\_UNLOAD PROC near

push ax

push es

mov cl, 0h

mov al,es:[81h+1]

cmp al,'/'

jne WRONG\_ARG

mov al,es:[81h+2]

cmp al,'u'

jne WRONG\_ARG

mov al,es:[81h+3]

cmp al,'n'

jne WRONG\_ARG

mov cl,1h

WRONG\_ARG:

pop es

pop ax

ret

CHECK\_UNLOAD ENDP

CHECK\_LOADED PROC NEAR

push ax

push dx

push es

push si

mov cl, 0h

mov ah, 35h

mov al, 09h

int 21h

```
mov si, offset INT_SIG
sub si, offset ROUT
mov dx, es:[bx + si]
cmp dx, INT_SIG
jne NOT_LOADED
```

```
mov cl, 1h
```

NOT\_LOADED:

```
pop si
pop es
pop dx
pop ax
ret
```

CHECK\_LOADED ENDP

LOAD\_ROUT PROC near

```
push ax
push cx
push dx
```

```
mov KEEP_PSP, es
```

```
mov ah, 35h
mov al, 09h
int 21h
```

```
mov KEEP_CS, es
mov KEEP_IP, bx
```

```
push es
push bx
push ds
```

```
lea dx, ROUT
mov ax, SEG ROUT
mov ds, ax
```

```

mov ah, 25h
mov al, 09h
int 21h

pop ds
pop bx
pop es

mov dx, offset ROUT_CHANGED
call WRITE

lea dx, END_ROUT_P
mov cl, 4h
shr dx, cl
inc dx

add dx, 100h

xor ax, ax

mov ah, 31h
int 21h

pop dx
pop cx
pop ax
ret
LOAD_ROUT ENDP

UNLOAD_ROUT PROC near
push ax
push si

call CHECK_LOADED
cmp cl, 1h
jne ROUT_ISNOT_LOADED

cli

```

push ds

push es

mov ah, 35h

mov al, 09h

int 21h

mov si, offset KEEP\_IP

sub si, offset ROUT

mov dx, es:[bx + si]

mov ax, es:[bx + si + 2]

mov ds, ax

mov ah, 25h

mov al, 09h

int 21h

mov ax, es:[bx + si + 4]

mov es, ax

push es

mov ax, es:[2ch]

mov es, ax

mov ah, 49h

int 21h

pop es

mov ah, 49h

int 21h

pop es

pop ds

sti

mov dx, offset ROUT\_UNLOADED

call WRITE

```
jmp UNLOAD_END
```

ROUT\_ISNOT\_LOADED:

```
mov dx, offset ROUT_IS_NOT_LOADED
call WRITE
```

UNLOAD\_END:

```
pop si
pop ax
ret
```

UNLOAD\_ROUT ENDP

MAIN PROC FAR

```
mov ax, DATA
mov ds, ax
```

```
call CHECK_UNLOAD
cmp cl, 1h
je START_UNLOAD
```

```
call CHECK_LOADED
cmp ch, 1h
je ALREADY_LOADED
```

```
call LOAD_ROUT
jmp EXIT
```

START\_UNLOAD:

```
call UNLOAD_ROUT
jmp EXIT
```

ALREADY\_LOADED:

```
mov dx, offset ROUT_LOADED
call WRITE
jmp EXIT
```

EXIT:

    xor al, al

    mov ah, 4ch

    int 21h

MAIN endp

CODE ends

END Main