

**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 обработчиков**  
**прерываний**

Студент гр. 7381

Габов Е. С.

Преподаватель

Ефремов М. А.

Санкт-Петербург

## Постановка задачи

**Цель работы:** Исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Пользовательский обработчик прерывания получает управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре. Он обрабатывает скан-код и осуществляет 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 работы программы

- 1) Проверяет, установлено ли пользовательское прерывание;
- 2) Устанавливает пользовательское прерывание и оставляет его резидентным в памяти;
- 3) Восстанавливает системное прерывание, удаляя пользовательское и высвобождая занимаемую им память.

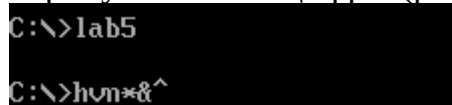
Пользовательское прерывание выполняет следующее действие: при вводе цифр 0-9 с основной клавиатуры заменяет их символами верхнего регистра

### Описание функций

| Название      | Назначение   |
|---------------|--|
| MY_INT        | Осуществляет обработку прерывания  |
| OLD_INT       | Сохраняет сегмент и смещение системного прерывания                               |
| SET_NEW_INT   | Устанавливает вместо системного пользовательское прерывание                      |
| LOAD_MY_INT   | Оставляет прерывание резидентным в памяти  |
| DELETE_MY_INT | Восстанавливает системное прерывание и освобождает память, занимаемую резидентом |
| PRINT         | Вывод строки на экран  |

### Ход работы

Запустим программу и попробуем ввести цифры (рис. 1):



```
C:\>lab5
C:\>h0n*8^
```

Рисунок 1

Проверим память, запустив 3 лабораторную работу (рис. 2):

```

C:\>lab3_2
Amount of available memory:      648096 b
Size of extended memory:        15360 Kb
List of memory control blocks:
MCB type: 4Dh   PSP address: 0008h   Size:      16 b
MCB type: 4Dh   PSP address: 0000h   Size:      64 b
MCB type: 4Dh   PSP address: 0040h   Size:     256 b
MCB type: 4Dh   PSP address: 0192h   Size:     144 b
MCB type: 4Dh   PSP address: 0192h   Size:     640 b      LAB5
MCB type: 4Dh   PSP address: 01C5h   Size:     144 b
MCB type: 4Dh   PSP address: 01C5h   Size:     816 b      LAB3_2
MCB type: 5Ah   PSP address: 0000h   Size:   647264 b      alled!

```

Рисунок 2

После завершения программы, область, установленная резидентной из памяти, не высвобождается.

При повторном вызове программы (рис. 3):

```

C:\>lab5
Interrupt is already installed!

```

Рисунок 3

Выгрузим прерывание и попробуем ввести цифры (рис. 4):

```

C:\>lab5 /un
C:\>122112

```

Рисунок 4

Снова проверим память (рис. 5):

```

C:\>lab3_2
Amount of available memory:      648912 b
Size of extended memory:        15360 Kb
List of memory control blocks:
MCB type: 4Dh   PSP address: 0008h   Size:      16 b
MCB type: 4Dh   PSP address: 0000h   Size:      64 b
MCB type: 4Dh   PSP address: 0040h   Size:     256 b
MCB type: 4Dh   PSP address: 0192h   Size:     144 b
MCB type: 4Dh   PSP address: 0192h   Size:     816 b      LAB3_2
MCB type: 5Ah   PSP address: 0000h   Size:   648080 b      i |  ≡

```

Рисунок 5

Область памяти, ранее выделенная как резидентная, высвобождена.

### Ответы на контрольные вопросы:

- 1) Какого типа прерывания использовались в работе?  
В работе использовались программные (int 21h и int 16h) и аппаратные прерывания (int 09h)
- 2) Чем отличается скан-код от кода ASCII?  
Скан-код – код, присвоенный каждой клавише клавиатуры, с

помощью которого можно опознать, какая клавиша была нажата. Скан-коды жестко привязаны к каждой клавише.

ASCII код – код, используемый для представления символов в памяти компьютера (код может различаться в зависимости от таблицы кодировки или ее разновидности).

### **Вывод**

В результате выполнения данной лабораторной работы были исследованы работа и возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры обработчиков прерываний.

## Приложение А

### 1) Lab5.asm

code segment

assume cs:code, ds:data, ss:Tstack

start\_mem:

PSP dw 0

KEEP\_CS dw 0

KEEP\_IP dw 0

Int9 dd 0

Scancode\_1 db 2

Scancode\_2 db 3

Scancode\_3 db 4

Scancode\_4 db 5

Scancode\_5 db 6

Scancode\_6 db 7

Scancode\_7 db 8

Scancode\_8 db 9

Scancode\_9 db 10

Scancode\_0 db 11

KEEP\_AX dw 0

KEEP\_SS dw 0

KEEP\_SP dw 0

MY\_STACK dw 100 DUP(?)

end\_of\_my\_stack:

push\_main macro

mov CS:KEEP\_AX, AX

mov CS:KEEP\_SS, SS

mov CS:KEEP\_SP, SP

mov AX, SEG MY\_STACK

```
mov    SS, AX
```

```
mov    SP, offset end_of_my_stack
```

```
push   ax
```

```
push   bx
```

```
push   cx
```

```
push   dx
```

```
endm
```

```
pop_main macro
```

```
pop    dx
```

```
pop    cx
```

```
pop    bx
```

```
pop    ax
```

```
mov    ax, CS:KEEP_SS
```

```
mov    ss, ax
```

```
mov    SP, CS:KEEP_SP
```

```
mov    ax, CS:KEEP_AX
```

```
endm
```

```
my_int proc far
```

```
    jmp    body
```

```
    Int_Tag dw 1234h
```

```
body:
```

```
    push_main
```

```
    in     al, 60h
```

```
    mov    ch, 00h
```

```
    cmp    al, cs:[Scancode_1]
```

```
        jne        next1
        mov        cl, '!'
        jmp        do_req
next1:
        cmp        al, cs:[Scancode_2]
        jne        next2
        mov        cl, '@'
        jmp        do_req
next2:
        cmp        al, cs:[Scancode_3]
        jne        next3
        mov        cl, '#'
        jmp        do_req
next3:
        cmp        al, cs:[Scancode_4]
        jne        next4
        mov        cl, '$'
        jmp        do_req
next4:
        cmp        al, cs:[Scancode_5]
        jne        next5
        mov        cl, '%'
        jmp        do_req
next5:
        cmp        al, cs:[Scancode_6]
        jne        next6
        mov        cl, '^'
        jmp        do_req
next6:
        cmp        al, cs:[Scancode_7]
        jne        next7
        mov        cl, '&'
        jmp        do_req
next7:
        cmp        al, cs:[Scancode_8]
        jne        next8
        mov        cl, '*'
```

```

        jmp      do_req
next8:
        cmp      al, cs:[Scancode_9]
        jne      next9
        mov      cl, '('
        jmp      do_req
next9:
        cmp      al, cs:[Scancode_0]
        jne      int9do
        mov      cl, ')'
        jmp      do_req

int9do:
        pop_main
        jmp      cs:[Int9]

do_req:
        in        al, 61h
        mov  ah, al
        or        al, 80h
        out  61h, al
        xchg  ah, al
        out  61H, al
        mov  al, 20h
        out  20h, al

        mov  ah, 05h
        int  16h
        or        al, al
        jnz  skip
        jmp  to_quit

;Aey i?enoe aooa?a iaai i?inoi onoaiaaeou cia?aiea
;y?aeee 0040:001A ?aaiui cia?aie? y?aeee 0040:001C.
skip:
        push  es
        push  si
        mov  ax, 0040h

```



```
    mov  es, ax
    mov  si, 001ah
    mov  ax, es:[si]
    mov  si, 001ch
    mov  es:[si], ax
    pop      si
    pop      es
```

to\_quit:

```
    pop_main
    mov  al, 20h
    out  20h, al
    iret
```

my\_int endp

end\_mem:

old\_int\_save proc near

```
    push_main
    push  es
    push  di
    mov      ah, 35h
    mov      al,  09h
    int     21h
    mov  cs:KEEP_IP, bx
    mov  cs:KEEP_CS, es
    mov word ptr Int9+2, es
    mov word ptr Int9, bx
    pop  di
    pop  es
    pop_main
    ret
```

old\_int\_save endp

set\_new\_int proc near

```
    push_main
    push  ds
```

```

        mov  dx, offset my_int
        mov  ax, seg my_int
        mov  ds, ax
        mov     ah, 25h
        mov     al, 09h
        int    21h
        pop  ds
        pop_main
        ret
set_new_int endp

```

```

load_my_int proc near
        mov  dx, seg code
        add  dx, (end_mem-start_mem)
        mov  cl, 4
        shr  dx, cl ;div 16
        inc  dx
        mov  ah, 31h
        int  21h
        ret
load_my_int endp

```

```

delete_my_int proc near
        cli
        push_main
        push  ds
        push  es
        push  di
        mov     ah,35h
        mov     al,09h
        int    21h
        mov  ax, es:[2]
        mov  cs:KEEP_CS, ax
        mov  ax, es:[4]
        mov  cs:KEEP_IP, ax
        mov  ax, es:[0]
        mov  cx, ax

```

```

mov  es, ax
mov  ax, es:[2Ch]
mov  es, ax
xor  ax, ax
mov  ah, 49h
int  21h
mov  es, cx
xor  ax, ax
mov  ah, 49h
int  21h
mov  dx, cs:KEEP_IP
mov  ax, cs:KEEP_CS
mov  ds, ax
mov      ah, 25h
mov      al, 09h
int  21h
pop  di
pop  es
pop  ds
pop_main
sti
ret

```

delete\_my\_int endp

main proc near

```

push  ds
mov  ax, seg data
mov  ds, ax
pop  cs:PSP

mov  es, cs:PSP
mov  al, es:[80h]
cmp  al, 4
jne  Empty_Tail

mov  al, byte PTR es:[82h]

```

```

    cmp  al, '/'
    jne      Empty_Tail
    mov  al, byte PTR es:[83h]
    cmp  al, 'u'
    jne      Empty_Tail
    mov  al, byte PTR es:[84h]
    cmp  al, 'n'
    jne      Empty_Tail

    mov  IsDelete, 1

```

Empty\_Tail:

```

    mov     ah, 35h
    mov     al, 09h
    int     21h
    mov  ax, es:[bx+3]
    cmp  ax, 1234h
    je      already_inst

    cmp  IsDelete, 1
    je      not_inst

    call  old_int_save
    call  set_new_int
    call  load_my_int

    jmp  exit

```

already\_inst:

```

    cmp  IsDelete, 1
    je      delete_my_int_main_m
    mov  dx, offset Inst_Mess
    mov  ah, 09h
    int  21h
    jmp  exit

```

delete\_my\_int\_main\_m:

```

    call  delete_my_int

```

```
    jmp    exit
```

```
not_inst:
```

```
    mov     dx, offset Not_Inst_Mess
```

```
    mov    ah, 09h
```

```
    int    21h
```

```
    jmp     exit
```

```
exit:
```

```
    xor    al, al
```

```
    mov    ah, 4Ch
```

```
    int    21h
```

```
    ret
```

```
main endp
```

```
code ends
```

```
data segment
```

```
    IsDelete      db 0
```

```
    Inst_Mess     db 'Interrupt is already installed!', 10, 13, '$'
```

```
    Not_Inst_Mess db 'Interrupt is not installed!', 10, 13, '$'
```

```
data ends
```

```
Tstack segment stack
```

```
    dw 128 dup (?)
```

```
Tstack ends
```

```
end main
```

