

**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. 9382

\_\_\_\_\_

Юрьев С.Ю.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2021

## Цель работы.

Исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

## Ход работы.

Был написан и отлажен программный модуль lab.exe.

Процедуры, используемые в программе:

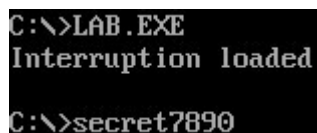
- 1) ROUT - процедура обработчика прерываний.
- 2) CHECK - процедура проверки установки резидента.
- 3) LOADP - процедура загрузки резидента.
- 4) UNLOADP - процедура выгрузки резидента.
- 5) CHECKUN - процедура проверки наличия “/un”.
- 6) PRINT - вывод строки из DX на экран.

В табл. 1 представлена обработка нажатий клавиатуры.

Клавиша	1	2	3	4	5	6
Записанный символ	s	e	c	r	e	t

Табл. 1.

На рис. 1 представлен вид командной строки после запуска программы и нажатия клавиш 1, 2, 3, 4, 5, 6, 7, 8, 9, 0.



```
C:\>LAB.EXE
Interruption loaded
C:\>secret7890_
```

Рис. 1.

На рис. 2 представлен вывод программы memory.com, которая выводит информацию о МСВ блоках, после запуска lab.exe.

```
C:\>MEMORY.COM
Available memory: 644160 b
Extended memory: 15360 kb
MCB: 1 Owner: MS DOS Size: 16
last 8 bytes:
MCB: 2 Owner: free Size: 64
last 8 bytes:
MCB: 3 Owner: 0040 Size: 256
last 8 bytes:
MCB: 4 Owner: 0192 Size: 144
last 8 bytes:
MCB: 5 Owner: 0192 Size: 4576
last 8 bytes: LAB
MCB: 6 Owner: 02BB Size: 4144
last 8 bytes:
MCB: 7 Owner: 02BB Size: 644160
last 8 bytes: MEMORY
```

Рис. 2. Вывод memory.com после запуска lab.exe

По рисунку видно, что процедура прерываний осталась резидентной в памяти и располагается в блоках 4 и 5.

На рис. 3 представлен вывод программы memory.com, которая выводит информацию о МСВ блоках, после запуска lab.exe для выгрузки резидентного обработчика прерываний. После это были нажаты клавиши 1, 2, 3, 4, 5, 6, 7, 8, 9, 0.

```
C:\>LAB.EXE /un
Interruption has been unloaded

C:\>MEMORY.COM
Available memory: 648912 b
Extended memory: 15360 kb
MCB: 1 Owner: MS DOS Size: 16
last 8 bytes:
MCB: 2 Owner: free Size: 64
last 8 bytes:
MCB: 3 Owner: 0040 Size: 256
last 8 bytes:
MCB: 4 Owner: 0192 Size: 144
last 8 bytes:
MCB: 5 Owner: 0192 Size: 648912
last 8 bytes: MEMORY

C:\>1234567890_
```

Рис. 3. Вывод memory.com при вводе '1', '2', '3', ...

По рисунку видно, что память для резидентного обработчика была освобождена. Также видно, что обработчик закончил работу, и замена символов не была произведена.

### **Контрольные вопросы.**

1) Какого типа прерывания использовались в работе?

Ответ: прерывания функций DOS (21h), прерывания функций BIOS (16h, 09h)

2) Чем отличается скан код от кода ASCII?

Ответ: код ASCII – номер символа в таблице ASCII, а скан код – код, присвоенный каждой клавише, с помощью которого драйвер клавиатуры распознает, какая клавиша была нажата.

### **Выводы.**

В ходе выполнения данной работы была реализована программа, загружающая и выгружающая пользовательское прерывание от нажатия клавиатуры в память.

## ПРИЛОЖЕНИЕ А.

### Исходный код программы.

```
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:AStack
; Процедуры
;-----
ROUT PROC FAR
; обработчик прерываний
    jmp starttrout
routdata:
    signature DW 2910h

    keep_ss dw ?
    keep_sp dw ?
    keep_ax dw ?

    KEEP_IP DW 0
    KEEP_CS DW 0
    KEEP_PSP DW 0

    rout_stack dw 16 dup(?)
    end_stack dw ?

    symb db 0

starttrout:
    mov keep_ss, ss
    mov keep_sp, sp
    mov keep_ax, ax
    mov ax, seg rout_stack
    mov ss, ax
```

```

mov    sp, offset end_stack

PUSH AX ; сохранение изменяемых регистров
push bx
push cx
push dx
push si
push es
push ds

mov    ax, seg symb
mov    ds, ax

in     al, 60h
cmp    al, 02h
je     out_s
cmp    al, 03h
je     out_e
cmp    al, 04h
je     out_c
cmp    al, 05h
je     out_r
cmp    al, 06h
je     out_e
cmp    al, 07h
je     out_t

pushf
call   dword ptr cs:keep_ip
jmp    endc

out_s:
mov    symb, 's'
jmp    next

out_e:
mov    symb, 'e'
jmp    next

out_c:
mov    symb, 'c'
jmp    next

out_r:
mov    symb, 'r'
jmp    next

out_t:
mov    symb, 't'

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

```

```

        out            20h, al

print_symb:
        mov     ah, 05h
        mov     cl, SYMB
        mov     ch, 00h
        int     16h
        or      al, al
        jz      endc

        mov     ax, 0040h
        mov     es, ax
        mov     ax, es:[1ah]
        mov     es:[1ch], ax
        jmp     print_symb

endc:

        pop     ds
        pop     es
        pop     si
        pop     dx
        pop     cx
        pop     bx
        POP     AX          ; восстановление регистров

        mov     ax, keep_ax
        mov     ss, keep_ss
        mov     sp, keep_sp

        MOV     AL, 20H
        OUT     20H,AL
        IRET

ROUT ENDP
;-----
last_byte:
;-----
CHECK proc
        push    ax
        push    bx
        push    si

        MOV     AH, 35H ; функция получения вектора
        MOV     AL, 09H ; номер вектора
        INT     21H
        mov     si, offset signature
        sub     si, offset rout
        mov     ax, es:[bx+si]
        cmp     ax, signature
        jne     endcheck
        mov     loaded, 1

endcheck:
        pop     si

```

```

        pop    bx
        pop    ax
        ret
CHECK endp
;-----
LOADP proc
        push  ax
        push  bx
        push  cx
        push  dx
        push  es
        push  ds

        MOV AH, 35H ; функция получения вектора
        MOV AL, 09H ; номер вектора
        INT 21H
        MOV KEEP_IP, BX ; запоминание смещения
        MOV KEEP_CS, ES ; и сегмента
        MOV DX, OFFSET ROUT ; смещение для процедуры в DX
        MOV AX, SEG ROUT      ; сегмент процедуры
        MOV DS, AX            ; помещаем в DS
        MOV AH, 25H           ; функция установки вектора
        MOV AL, 09H           ; номер вектора
        INT 21H               ; меняем прерывание
        POP  DS

        mov DX, offset LAST_BYTE ; размер в байтах от начала
        mov CL, 4                 ; перевод в параграфы
        shr DX, CL
        add  dx, 10fh
        inc  DX                   ; размер в параграфах
        mov AH, 31h
        int 21h

        pop  es
        pop  dx
        pop  cx
        pop  bx
        pop  ax
        ret
LOADP endp
;-----
UNLOADP proc
        cli
        push  ax
        push  bx
        push  dx
        push  ds
        push  es
        push  si

        MOV AH, 35H ; функция получения вектора
        MOV AL, 09H ; номер вектора
        INT 21H

```



```

mov     si, offset keep_ip
sub     si, offset rout
mov     dx, es:[bx+si]
mov     ax, es:[bx+si+2]

push    ds
mov     ds, ax
MOV     AH, 25H           ; функция установки вектора
MOV     AL, 09H           ; номер вектора
INT     21H               ; меняем прерывание
POP     DS

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

sti

pop     si
pop     es
pop     ds
pop     dx
pop     bx
pop     ax
ret
UNLOADP   endp
;-----
CHECKUN   proc
push    ax
push    es

mov     ax, keep_psp
mov     es, ax
cmp     byte ptr es:[82h], '/'
jne     endun
cmp     byte ptr es:[83h], 'u'
jne     endun
cmp     byte ptr es:[84h], 'n'
jne     endun
mov     un, 1

endun:
pop     es
pop     ax
ret
CHECKUN   endp
;-----

```

```

PRINT proc near
    mov  ah, 09h
    int 21h
    ret
PRINT endp
;-----
; Код
MAIN PROC
    push DS          ;\ Сохранение адреса начала PSP в стеке
    sub  AX,AX        ; > для последующего восстановления по
    push AX          ;/ команде ret, завершающей процедуру.
    mov  AX,DATA      ; Загрузка сегментного
    mov  DS,AX        ; регистра данных.
    mov  keep_psp, es

    call check
    call checkun
    cmp  un, 1
    je  unload1

    mov  al, loaded
    cmp  al, 1
    jne  load1
    mov  dx, offset loaded_inf
    call print
    jmp  exit

load1:
    mov  dx, offset load_inf
    call print
    call loadp
    jmp  exit

unload1:
    cmp  loaded, 1
    jne  notloaded1
    call UNLOADP
    mov  dx, offset unload_inf
    call print
    jmp  exit

notloaded1:
    mov  dx, offset not_load_inf
    call print

exit:
; Выход в DOS
    xor  AL,AL
    mov  AH,4Ch
    int 21H
MAIN ENDP
CODE ENDS

AStack SEGMENT STACK
    DW 128 DUP(0)
AStack ENDS

```

```

DATA SEGMENT
    load_inf      db    'Interruption loaded',0DH,0AH,'$'
    loaded_inf    db    'Interruption already loaded',0DH,0AH,'$'
    unload_inf    db    'Interruption has been
unloaded',0DH,0AH,'$'
    not_load_inf  db    'Interruption not loaded',0DH,0AH,'$'

    loaded        db    0
    un            db    0
DATA ENDS

        END MAIN

```