

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

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

Студент гр. 8382

\_\_\_\_\_

Нечепуренко Н.А.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2020

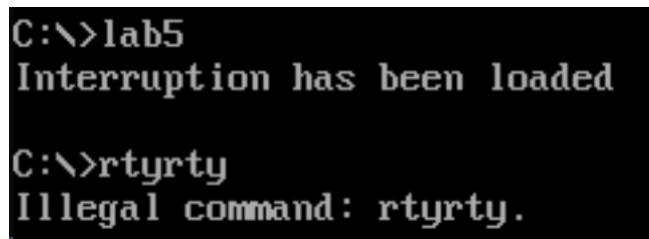
### **Цель работы.**

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

### **Выполнение работы.**

Для реализации пользовательского обработчика прерываний от клавиатуры был частично использован подход, примененный в лабораторной работе 4. Пользовательский обработчик анализирует последнюю нажатую клавишу и, если это одна из клавиш q, w, e или их заглавные аналоги, то заменяет их в буфере на r, t, y соответственно, иначе – передаёт управление стандартному обработчику. Полный исходный код программы находится в Приложении А.

Результат работы программы при введённой строке «qwerty» приведён на рисунке 1.

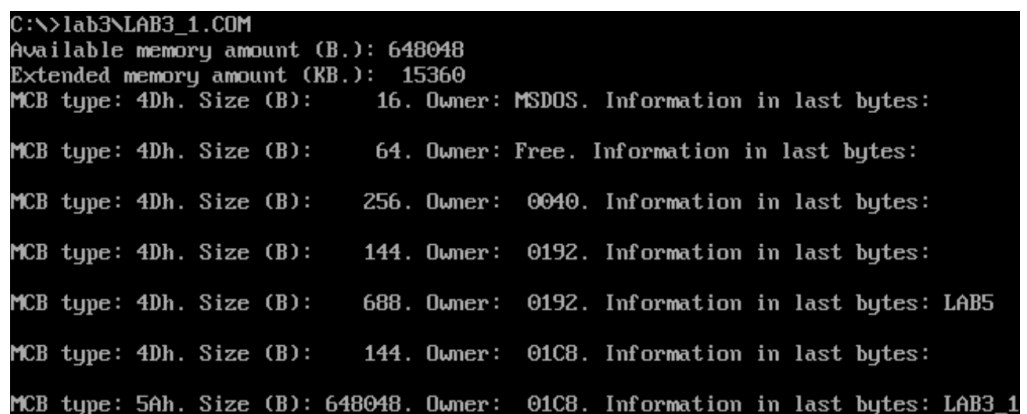


```
C:\>lab5
Interruption has been loaded

C:\>rtyrty
Illegal command: rtyrty.
```

Рисунок 1 – Результат работы программы

На рисунке 2 рассмотрим MCB блоки операционной системы после запуска программы.



```
C:\>lab3\LAB3_1.COM
Available memory amount (B.): 648048
Extended memory amount (KB.): 15360
MCB type: 4Dh. Size (B): 16. Owner: MSDOS. Information in last bytes:
MCB type: 4Dh. Size (B): 64. Owner: Free. Information in last bytes:
MCB type: 4Dh. Size (B): 256. Owner: 0040. Information in last bytes:
MCB type: 4Dh. Size (B): 144. Owner: 0192. Information in last bytes:
MCB type: 4Dh. Size (B): 688. Owner: 0192. Information in last bytes: LAB5
MCB type: 4Dh. Size (B): 144. Owner: 01C8. Information in last bytes:
MCB type: 5Ah. Size (B): 648048. Owner: 01C8. Information in last bytes: LAB3_1
```

Рисунок 2 – MCB блоки после запуска программы

Теперь попытаемся повторно загрузить обработчик и ещё раз рассмотрим МСВ блоки (см. рис. 3).

```
C:\>lab5
Interruption is already loaded

C:\>lab3\LAB3_1.COM
Available memory amount (B.): 648048
Extended memory amount (KB.): 15360
MCB type: 4Dh. Size (B): 16. Owner: MSDOS. Information in last bytes:
MCB type: 4Dh. Size (B): 64. Owner: Free. Information in last bytes:
MCB type: 4Dh. Size (B): 256. Owner: 0040. Information in last bytes:
MCB type: 4Dh. Size (B): 144. Owner: 0192. Information in last bytes:
MCB type: 4Dh. Size (B): 688. Owner: 0192. Information in last bytes: LAB5
MCB type: 4Dh. Size (B): 144. Owner: 01C8. Information in last bytes:
MCB type: 5Ah. Size (B): 648048. Owner: 01C8. Information in last bytes: LAB3_1
```

Рисунок 3 – МСВ блоки после повторного запуска программы

Можно заметить, что программа вывела сообщение о том, что обработчик уже загружен и не стала помещать в резидентную память ещё одну копию кода обработчика.

Выгрузим обработчик, попробуем ввести строку «qwerty» и посмотрим на МСВ блоки (см. рис. 4).

```
C:\>lab5 /un
Interruption has been unloaded

C:\>qwerty
Illegal command: qwerty.

C:\>lab3\LAB3_1.COM
Available memory amount (B.): 648912
Extended memory amount (KB.): 15360
MCB type: 4Dh. Size (B): 16. Owner: MSDOS. Information in last bytes:
MCB type: 4Dh. Size (B): 64. Owner: Free. Information in last bytes:
MCB type: 4Dh. Size (B): 256. Owner: 0040. Information in last bytes:
MCB type: 4Dh. Size (B): 144. Owner: 0192. Information in last bytes:
MCB type: 5Ah. Size (B): 648912. Owner: 0192. Information in last bytes: LAB3_1
```

Рисунок 4 – Выгрузка обработчика

Программа отработала корректно.

### **Контрольные вопросы.**

1. Какого типа прерывания использовались в работе?

В работе использовалось прерывание DOS 21h, прерывание BIOS 16h для работы с клавиатурой, прерывание BIOS 09h, которое обрабатывалось пользовательским кодом. Прерывание 09h генерируется при нажатии клавиши клавиатуры.

2. Чем отличается скан-код от кода ASCII?

Скан-код – это число, с помощью которого драйвер распознаёт какая клавиша была нажата. ASCII код – это числовое значение символа в таблице кодировки. Таким образом, символы «q» и «Q» имеют разные ASCII коды, но один скан-код, так как клавиша используется одна и та же.

### **Выводы.**

В результате выполнения работы был реализован пользовательский обработчик прерываний от клавиатуры, который размещается в резидентной памяти. Обработчик заменяет введённую с помощью клавиатуры букву, либо передаёт управление стандартному обработчику.

## ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ.

```
codeseg segment
    assume cs:codeseg, ss:astack, ds:dataseg

main PROC FAR
    jmp resident_start

resident_data:
    inter_signature dw 1337h
    keep_ip  dw 0
    keep_cs  dw 0
    keep_psp dw 0
    keep_ss   dw 0
    keep_sp   dw 0
    keep_ax   dw 0
    inter_stack dw 100 dup("?")

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

    mov ax, seg inter_stack
    mov ss, ax
    mov sp, offset resident_start

    push bx
    push cx
    push dx
    push si
    push ds
    push es

    in al, 60h
    cmp al, 10h
    je resident_q
    cmp al, 11h
    je resident_w
    cmp al, 12h
```

```

        je resident_e

resident_default:
        pushf
        call dword PTR cs:keep_ip
        jmp resident_final

resident_q:
        mov cl, 'r'
        jmp resident_handler

resident_w:
        mov cl, 't'
        jmp resident_handler

resident_e:
        mov cl, 'y'
        jmp resident_handler

resident_handler:
        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
        mov ch, 00h
        int 16h

resident_final:
        pop es
        pop ds
        pop si
        pop dx
        pop cx
        pop bx

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

```

```

        mov ax, keep_ax

        mov al, 20h
        out 20h, al

        iret
main ENDP
resident_part_end:

init proc far
    mov ax, dataseg
    mov ds, ax
    mov keep_psp, es
    call check_un
    mov ax, un_flag
    cmp ax, 0
    jne init_reset
    call set_inter
    jmp init_final
init_reset:
    call reset_inter
init_final:
    mov ax, 4c00h
    int 21h
    ret
init endp

set_inter proc near
    push ax
    push bx
    push dx
    push di
    push si
    push cx
    push ds
    push es

set_inter_get_prev:
    mov ax, 3509h
    int 21h

```

```

        mov keep_cs, es
        mov keep_ip, bx

set_inter_check:
        mov si, offset inter_signature
        sub si, offset main
        mov ax, es:[bx+si] ; get resident_data
        cmp ax, 1337h ; check signature to be equal 1337h
        jne set_inter_set_new

set_inter_already_set:
        mov di, offset msg_inter_already
        call print
        jmp set_inter_final

set_inter_set_new:
        push ds
        mov dx, offset main
        mov ax, seg main
        mov ds, ax
        mov ax, 2509h
        int 21h
        pop ds

        mov di, offset msg_inter_loaded
        call print

set_inter_make_resident:
        mov dx, offset resident_part_end
        xor cx, cx
        mov cl, 4
        shr dx, cl
        add dx, 16h
        inc dx
        mov ah, 31h
        int 21h
set_inter_final:
        pop es
        pop ds
        pop cx
        pop si
        pop di

```



```

    pop dx
    pop bx
    pop ax
    ret
set_inter endp

```

```

reset_inter proc near

```

```

    push ax
    push bx
    push dx
    push ds
    push es
    push si
    push di

```

```

reset_inter_get_prev:

```

```

    mov ax, 3509h
    int 21h
    mov si, offset inter_signature
    sub si, offset main
    mov ax, es:[bx+si] ; get resident_data
    cmp ax, 1337h ; check signature to be equal 1337h
    jne reset_inter_final

```

```

reset_inter_restore:

```

```

    cli
    push ds
    mov dx, es:[bx+si+2]; cs
    mov ax, es:[bx+si+4]; ip
    mov ds, ax
    mov ax, 2509h
    int 21h
    pop ds
    sti

```

```

reset_inter_free_memory:

```

```

    mov ax, es:[bx+si+6]; keep_psp
    mov es, ax
    push es
    mov ax, es:[2ch]; there is adr in psp of memory needs to be free
    mov es, ax
    mov ah, 49h
    int 21h

```

```

    pop es
    mov ah, 49h
    int 21h
    mov di, offset msg_inter_unloaded
    call print

reset_inter_final:
    pop di
    pop si
    pop es
    pop ds
    pop dx
    pop bx
    pop ax
    ret
reset_inter ENDP

check_un proc near
    push ax
    push es
    mov ax, keep_psp
    mov es, ax
    ; check cmd tail
    cmp byte ptr es:[81h+1], "/"
    jne check_un_final
    cmp byte ptr es:[81h+2], "u"
    jne check_un_final
    cmp byte ptr es:[81h+3], "n"
    jne check_un_final
    cmp byte ptr es:[81h+4], 13
    jne check_un_final
    mov ax, 1
    mov un_flag, ax
check_un_final:
    pop es
    pop ax
    ret
check_un endp

print proc near
    ; prints di content
    push dx

```

```

        push ax
        mov ah, 9h
        mov dx, di
        int 21h
        pop ax
        pop dx
        ret
print endp

codeseg ends

astack segment stack
        dw 100 dup("?")
astack ends

dataseg segment
        un_flag dw 0
        msg_inter_loaded db "Interruption has been loaded", 13, 10, "$"
        msg_inter_unloaded db "Interruption has been unloaded", 13, 10, "$"
        msg_inter_already db "Interruption is already loaded", 13, 10, "$"
dataseg ends

end init

```