

**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. 0382

Шангичев В. А.

Преподаватель

Ефремов М. А.

Санкт-Петербург

2022

### **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.

### **Задание.**

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.

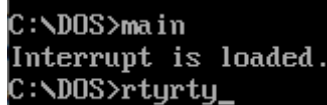
Шаг 5. Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

Шаг 6. Ответьте на контрольные вопросы.

### **Выполнение работы.**

Шаг 1. За основу написанной программы был взят исходный код из предыдущей лабораторной. В исходном коде адрес прерывания таймера был заменен на адрес прерывания клавиатуры. Также была изменена обрабатываемая процедура. В ней теперь происходит считывание скан-кода нажатой клавиши. Если эта клавиша не является символами 'q', 'w' или 'e', то управление передается стандартному прерыванию. В противном случае в память записывается символ, на который должен быть заменен введенный ('r', 't' и 'y' соответственно). Данный символ выводится на экран после отработки аппаратного прерывания.

Шаг 2. Программа была запущена. После этого в консоль была введена строка 'qwerty'. Результат представлен на рис. 1.



```
C:\DOS>main
Interrupt is loaded.
C:\DOS>rtyrty_
```

Рисунок 1 – обработка нажатия клавиш

Шаг 3. Для проверки размещения прерывания в памяти была запущена программа `info` из предыдущей лабораторной работы. Результат представлен на рис. 2.

```

C:\DOS>info
Available memory size:      647888
Extended memory size:      245760
Address: 016F PCP owner:0008 Size:      16 SC/SD:
Address: 0171 PCP owner:0000 Size:      64 SC/SD:
Address: 0176 PCP owner:0040 Size:     256 SC/SD:
Address: 0187 PCP owner:0192 Size:     144 SC/SD:
Address: 0191 PCP owner:0192 Size:     848 SC/SD: MAIN
Address: 01C7 PCP owner:01D2 Size:     144 SC/SD:
Address: 01D1 PCP owner:01D2 Size:    647888 SC/SD: INFO
C:\DOS>

```

Рисунок 2 – размещение прерывания в памяти

Шаг 4. Программа была запущена еще раз, чтобы проверить, распознает ли она загруженное прерывание в памяти. Результат представлен на рис. 3

```

C:\DOS>main
Interrupt is already loaded.
C:\DOS>_

```

Рисунок 3 – распознавание загруженного прерывания

Шаг 5. Программа была вызвана с ключом выгрузки. После этого была введена строка из шага 2. Также было проверено состояние памяти после выгрузки прерывания.

```

C:\DOS>main /un
Interrupt is unloaded.
C:\DOS>qwerty

```

Рисунок 4 – проверка обработки клавиш

```

C:\DOS>info
Available memory size:      648912
Extended memory size:      245760
Address: 016F PCP owner:0008 Size:      16 SC/SD:
Address: 0171 PCP owner:0000 Size:      64 SC/SD:
Address: 0176 PCP owner:0040 Size:     256 SC/SD:
Address: 0187 PCP owner:0192 Size:     144 SC/SD:
Address: 0191 PCP owner:0192 Size:    648912 SC/SD: INFO
C:\DOS>_

```

## Рисунок 5 – проверка памяти

### **Ответы на контрольные вопросы.**

1. Какого типа прерывания использовались в работе?

В работе использовались аппаратные (09h, 16h) и программные (21h) прерывания.

2. Чем отличается скан-код от кода ASCII?

ASCII – это таблица кодировки, в которой некоторым распространенным печатным и непечатным символам присвоены числовые коды. Скан-код – это специальный код, присвоенный каждой клавише, с помощью которого драйвер клавиатуры распознает нажатую клавишу.

### **Выводы.**

Были исследованы возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Была написана программа, загружающая и выгружающая в память обработчик прерывания нажатия клавиш.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла main.asm

```
AStack    SEGMENT STACK
           DB 256 dup (?)
AStack    ENDS

DATA SEGMENT
    not_loaded_msg db 'Interrupt is not loaded.$', 0dh, 0ah
    loaded_msg db 'Interrupt is loaded.$', 0dh, 0ah
    unloaded_msg db 'Interrupt is unloaded.$', 0dh, 0ah
    already_loaded db 'Interrupt is already loaded.$', 0dh, 0ah

DATA ENDS

CODE      SEGMENT
ASSUME    CS:CODE, DS:DATA, SS:AStack

rout proc far
    jmp start
    signature dw 1234h
    keep_psp dw 0
    keep_ip dw 0
    keep_cs dw 0
    keep_ss dw 0
    keep_sp dw 0
    keep_ax dw 0
    key_sym db 0
    IStack db 50 dup(" ")

start:
    mov keep_ax, ax
    mov ax, ss
    mov keep_ss, ax
    mov keep_sp, sp
    mov ax, seg IStack
    mov ss, ax
    mov sp, offset start

    push ax
    push bx
    push cx
    push dx

    in al, 60h
    cmp al, 10h
    je q_key
    cmp al, 11h
    je w_key
    cmp al, 12h
    je e_key

    call dword ptr cs:keep_ip
```

```

        jmp exit_int

q_key:
    mov key_sym, 'r'
    jmp process_hardware_int
w_key:
    mov key_sym, 't'
    jmp process_hardware_int
e_key:
    mov key_sym, 'y'

process_hardware_int:
    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_key:
    mov ah, 05h
    mov cl, key_sym
    mov ch, 00h
    int 16h
    or al, al
    jz exit_int
    mov ax, 40h
    mov es, ax
    mov ax, ES:[1Ah]
    mov ES:[1Ch], ax
    jmp print_key

exit_int:
    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

    route_end:
rout endp

load_rout proc near
    push dx
    push ax
    push cx

    mov ax, 3509h
    int 21h

```



```

    mov keep_ip, bx
    mov keep_cs, es

    push ds
    mov dx, offset rout
    mov ax, seg rout
    mov ds, ax
    mov ax, 2509h
    int 21h
    pop ds

    mov dx, offset route_end
    mov cl, 4
    shr dx, cl
    inc dx
    mov ax, cs
    sub ax, keep_psp
    add dx, ax
    xor ax, ax
    mov ah, 31h
    int 21h

    pop cx
    pop ax
    pop dx

    ret
load_rout endp

unload_rout proc near
    push ax
    push bx

    mov AH, 35h
    mov AL, 09h
    int 21h

    cli
    push ds
    mov ax, es:[keep_cs]
    mov ds, ax
    mov dx, es:[keep_ip]
    mov ah, 25h
    mov al, 09h
    int 21h
    pop ds
    sti

    mov ax, es:[keep_psp]
    mov es, ax
    push es
    mov ax, es:[2Ch]
    mov es, ax
    mov ah, 49h
    int 21h
    pop es
    int 21h

```

```

        pop bx
        pop ax
        ret
unload_rout endp

load_check proc near
    ; return value:
    ; al - nonzero if interrupt is set.
    push si
    push dx
    push bx
    push ax

    mov ax, 3509h
    int 21h
    mov si, offset signature
    sub si, offset rout
    mov dx, es:[bx + si]
    mov al, 1
    cmp dx, 1234h
    je restore
    mov al, 0

    restore:
        mov bl, al
        pop ax
        mov al, bl
        pop bx
        pop dx
        pop si
    ret
load_check endp

cmd_flag_check proc near
    ; return value:
    ; al - nonzero if cmd tail contains flag
    push bx

    mov al, 0
    mov bh, es:[82h]
    cmp bh, '/'
    jne end_
    mov bh, es:[83h]
    cmp bh, 'u'
    jne end_
    mov bh, es:[84h]
    cmp bh, 'n'
    jne end_
    mov al, 1

    end_:
        pop bx
    ret
cmd_flag_check endp

```

```

print proc near
    push ax
    mov ah, 09
    int 21h
    pop ax
    ret
print endp

MAIN proc far
    mov ax, data
    mov ds, ax
    mov keep_psp, es

    call cmd_flag_check
    mov ah, al
    call load_check

    ; ah - is flag setted
    ; al - is interrupt loaded

    cmp ah, 1
    je flag_setted

    flag_not_setted:
        cmp al, 1
        je print_already_loaded
        mov dx, offset loaded_msg
        call print
        call load_rout
        jmp finish_program

    print_already_loaded:
        mov dx, offset already_loaded
        call print
        jmp finish_program

    flag_setted:
        cmp al, 1
        jne print_not_loaded
        call unload_rout
        mov dx, offset unloaded_msg
        call print
        jmp finish_program

    print_not_loaded:
        mov dx, offset not_loaded_msg
        call print

    finish_program:
        xor ax, ax
        mov ah, 4Ch
        int 21h

    main endp
code ends

```

end main

## Название файла: info.asm

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING

ORG 100h

start:

jmp begin

data:

available\_memory\_msg db "Available memory size: ",  
0dh, 0ah, '\$'

extended\_memory\_msg db "Extended memory size: ",  
0dh, 0ah, '\$'

mcb\_msg db "Address: PCP owner: Size:  
SC/SD: ", 0dh, 0ah, '\$'

tetr\_to\_hex PROC near

and AL,0Fh

cmp AL,09

jbe next

add AL,07

next:

add AL,30h

ret

tetr\_to\_hex ENDP

byte\_to\_hex PROC near

push CX

mov AH,AL

call tetr\_to\_hex

xchg AL,AH

mov CL,4

shr AL,CL

call tetr\_to\_hex

pop CX

ret

```
byte_to_hex ENDP
```

```
wrd_to_hex PROC near
```

```
    push BX
    mov BH,AH
    call byte_to_hex
    mov [DI],AH
    dec DI
    mov [DI],AL
    dec DI
    mov AL,BH
    call byte_to_hex
    mov [DI],AH
    dec DI
    mov [DI],AL
    pop BX
    ret
```

```
wrd_to_hex ENDP
```

```
byte_to_dec PROC near
```

```
    push CX
    push DX
    push ax
    xor AH,AH
    xor DX,DX
    mov CX,10
```

```
loop_bd:
```

```
    div CX
    or DL,30h
    mov [SI],DL
    dec SI
    xor DX,DX
    cmp AX,10
    jae loop_bd
    cmp AL,00h
    je end_1
    or AL,30h
    mov [SI],AL
```

```
end_1:
```

```

    pop ax
    pop DX
    pop CX
    ret
byte_to_dec ENDP

convert_to_decimal proc near
    ; ax - paragraph
    ; si - low digit of result
    push bx
    push dx

    mov bx, 16
    mul bx ; convert to num of bytes

    mov bx, 10
convert:
    div bx
    add dl, '0'
    mov [si], dl
    dec si
    xor dx, dx
    cmp ax, 0000h
    jne convert

    pop dx
    pop bx
    ret
convert_to_decimal endp

print proc near
    ; dx - offset of message
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
print endp

```

```

print_available_memory proc near
    mov ah, 4Ah
    mov bx, 0ffffh
    int 21h ; now bx contains size of available memory

    mov si, offset available_memory_msg
    add si, 33

    mov ax, bx
    call convert_to_decimal

    mov dx, offset available_memory_msg
    call print
    ret
print_available_memory endp

print_extended_memory proc near
    mov AL,30h
    out 70h,AL
    in AL,71h
    mov BL,AL

    mov AL,31h
    out 70h,AL
    in AL,71h
    mov bh, al

    mov ax, bx
    mov si, offset extended_memory_msg
    add si, 33

    call convert_to_decimal

    mov dx, offset extended_memory_msg
    call print
    ret
print_extended_memory endp

```

```

set_mcb_address proc near
    ; ax - address
    push di
    mov di, offset mcb_msg
    add di, 12
    call wrd_to_hex
    pop di
    ret
set_mcb_address endp

set_pcp_owner proc near
    ; es - address of mcb
    push ax
    push di
    mov ax, es:[1]
    mov di, offset mcb_msg
    add di, 27
    call wrd_to_hex
    pop di
    pop ax
    ret
set_pcp_owner endp

set_size proc near
    ; es - address of mcb
    push si
    push ax

    mov si, offset mcb_msg
    add si, 45
    mov ax, es:[3]
    call convert_to_decimal

    pop ax
    pop si
    ret
set_size endp

```



```

set_sc proc near
    push di
    push ax
    push bx

    mov di, offset mcb_msg
    add di, 54
    mov si, 8
    sc_write:
        mov bx, es:[si]
        mov [di], bx
        add si, 2
        add di, 2
        cmp si, 16
        jb sc_write

    pop bx
    pop ax
    pop di
    ret
set_sc endp

print_mcb proc near
    ; es - address of mcb
    push ax
    push dx

    mov ax, es
    call set_mcb_address
    call set_pcp_owner
    call set_size
    call set_sc
    mov dx, offset mcb_msg
    call print

    pop dx
    pop ax
    ret
print_mcb endp

```

```

print_memory_control_blocks proc near
    ; get address of first block
    mov ah, 52h
    int 21h
    mov es, es:[bx-2]

    print_msbs:
        call print_mcb
        mov ah, es:[0]
        cmp ah, 5Ah
        je end_
        mov ax, es
        add ax, es:[3]
        inc ax
        mov es, ax
        jmp print_msbs

    end_:

    ret
print_memory_control_blocks endp

```

```

begin:
    call print_available_memory
    call print_extended_memory
    call print_memory_control_blocks

```

```

    xor al, al
    mov ah, 4Ch
    int 21h

```

```

finish_program:

```

```

TESTPC ENDS

```

```

END start

```