

**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. 9383

\_\_\_\_\_

Соседков К.С.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2021

### **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.**

При выполнении работы был написан и отлажен программный модуль типа .EXE, который выполняет следующие функции

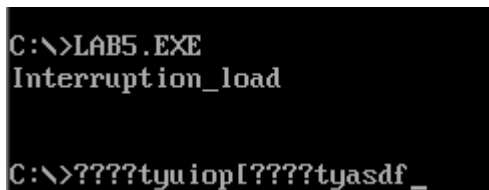
1) Проверяет, установлено ли пользовательское прерывание с вектором 09h.

2) Устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено, и 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.

4) 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.

Пользовательское прерывание заменяет буквы q,w,e,r на вопросительный знак, остальные символы выводятся без изменения. Результат работы пользовательского прерывания представлен на Рисунке 1.



```
C:\>LAB5.EXE
Interruption_load

C:\>????tyuiop[????tyasdf_
```

Рисунок 1: Результат работы пользовательского прерывания

Для проверки размещения прерывания в памяти использовалась программа из Лабораторной работы №3, которая отображает карту памяти в виде списка блоков MCB. Результат представлен на Рисунке 2.

```
C:\>LAB5.EXE
Interruption_load

C:\>LAB3_1.COM
Available memory: 643968
Extended memory: 15360
MCB Type: 4D   PSP Segment Address: 0008   MCB Size:    16   SC/CD:
MCB Type: 4D   PSP Segment Address: 0000   MCB Size:    64   SC/CD:
MCB Type: 4D   PSP Segment Address: 0040   MCB Size:   256   SC/CD:
MCB Type: 4D   PSP Segment Address: 0192   MCB Size:   144   SC/CD:
MCB Type: 4D   PSP Segment Address: 0192   MCB Size:  4768   SC/CD: LAB5
MCB Type: 4D   PSP Segment Address: 02C7   MCB Size:  4144   SC/CD:
MCB Type: 5A   PSP Segment Address: 02C7   MCB Size: 54144   SC/CD: LAB3_1
```

Рисунок 2: Размещение пользовательского прерывания в памяти

Пользовательское прерывание можно загрузить в память только один раз. Для выгрузки прерывания из памяти используется ключ „\un“. Результат представлен на Рисунке 3.

```
C:\>LAB5.EXE
Interruption_load

C:\>LAB5.EXE
Interruption_already_load

C:\>LAB5.EXE /un
Interruption_was_delete
```

Рисунок 3: Повторная  
загрузка и выгрузка  
прерывания

Для проверки того что запуск программы с ключом „\un“ действительно выгружает прерывание из памяти, была использована программа из Лабораторной работы №3.

```

C:\>LAB3_1.COM
Available memory: 648912
Extended memory: 15360
MCB Type: 4D   PSP Segment Address: 0008   MCB Size:    16   SC/CD:
MCB Type: 4D   PSP Segment Address: 0000   MCB Size:    64   SC/CD:
MCB Type: 4D   PSP Segment Address: 0040   MCB Size:   256   SC/CD:
MCB Type: 4D   PSP Segment Address: 0192   MCB Size:   144   SC/CD:
MCB Type: 5A   PSP Segment Address: 0192   MCB Size: 59088   SC/CD: LAB3_1

```

*Рисунок 4: Выгрузка прерывания из памяти*

## **Контрольные вопросы.**

### **1) Какого типа прерывания использовались в работе?**

Программные(21h, 10h) и аппаратные(09h).

### **2) Чем отличается скан код от кода ASCII?**

Скан-код — уникальный код присвоенный каждой клавише. Необходим для того что бы определить какая клавиша была нажата.

ASCII — таблица, в которой многим символам сопоставлены уникальные коды.

## **Выводы.**

При выполнении лабораторной работы был реализован пользовательский обработчик прерывания встроенный в стандартный обработчик от клавиатуры.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММ

Название файла: lab5.asm

ASTACK segment stack

dw 256 dup(?)

ASTACK ends

DATA segment

int\_already\_loaded db 'Interruption\_already\_load',0dh,0ah,0dh,0ah,'\$'

interruption\_loaded db 'Interruption\_load',0dh,0ah,0dh,0ah,'\$'

interruption\_delete db 'Interruption\_was\_delete',0dh,0ah,0dh,0ah,'\$'

DATA ends

CODE segment

assume cs:CODE, ds:DATA, ss:ASTACK

CUSTOM\_INTERRUPTION proc far

jmp start

int\_seg dw 256 dup(0)

int\_sig dw 0ffffh

keep\_ip dw 0

keep\_cs dw 0

```
keep_psp dw 0
keep_ax dw 0
keep_ss dw 0
keep_sp dw 0
```

start:

```
mov keep_ax, ax
mov keep_sp, sp
mov keep_ss, ss
mov ax, seg int_seg
mov ss, ax
mov ax, offset int_seg
add ax, 256
mov sp, ax
```

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

```
in al, 60h
    cmp al, 10h
    jl default_int
    cmp al, 13h
    jg default_int
```

```
    mov cl, '?'
    jmp change_key
```

default\_int:

```
    pushf
```



```
call dword ptr cs:keep_ip  
jmp end_interruption
```

```
change_key:  
    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 ch, 00h  
    int 16h  
    or  al, al  
    jz  end_interruption  
    mov ax, 40h  
    mov es, ax  
    mov ax, es:[1ah]  
    mov es:[1ch], ax  
    jmp print_key
```

```
end_interruption:  
    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
CUSTOM_INTERRUPTION endp
LAST:

```

```

UNLOAD_CUSTOM_INTERRUPTION proc
cli
    push ax
    push bx
    push dx
    push ds
    push es
    push si

    mov ah, 35h
    mov al, 09h
    int 21h
    mov si, offset keep_ip
    sub si, offset CUSTOM_INTERRUPTION
    mov dx, es:[bx + si]
    mov ax, es:[bx + si + 2]

    push ds
    mov ds, ax
    mov ah, 25h

```

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

```
    push dx
    mov dx,offset interruption_delete
    call PRINT
    pop dx
```

```
pop si
pop es
pop ds
pop dx
pop bx
pop ax
```

```
ret
```

```
UNLOAD_CUSTOM_INTERRUPTION endp
```

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

```
CHECK_CMD proc far

    push es
        mov ax, keep_psp
    mov es, ax

    mov al, es:[81h+1]
        cmp al, '/'
        jne set_zero

    mov al, es:[81h+2]
        cmp al, 'u'
        jne set_zero

    mov al, es:[81h+3]
        cmp al, 'n'
        jne set_zero

    mov ax, 1h
```

```

        jmp check_cmd_exit

set_zero:
        mov ax, 0h

check_cmd_exit:
        pop es
        ret
CHECK_CMD endp


IS_LOADED proc far
        push bx
        push si

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

        mov si, offset int_sig
        sub si, offset CUSTOM_INTERRUPTION
        mov dx, es:[bx + si]
        cmp dx, int_sig
        jne not_loaded
        mov ax, 1h
        jmp is_loaded_exit

not_loaded:
        mov ax, 0h

is_loaded_exit:

```

```
    pop si
    pop bx

    ret
IS_LOADED endp
```

```
LOAD_CUSTOM_INTERRUPTION proc far
```

```
    push ax
    push bx
    push cx
    push dx
    push es
    push ds
```

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

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

```
    mov dx, offset CUSTOM_INTERRUPTION
    mov ax, seg CUSTOM_INTERRUPTION
    mov ds,ax
```

```
    mov ah,25h
    mov al,09h
```

int 21h

pop ds

mov dx,offset interruption\_loaded  
call PRINT

mov dx, offset LAST  
mov cl,4h  
shr dx,cl  
inc dx

add dx,100h  
xor ax,ax

mov ah,31h  
int 21h

pop es  
pop dx  
pop cx  
pop bx  
pop ax  
ret

LOAD\_CUSTOM\_INTERRUPTION endp

MAIN proc far  
mov ax, DATA  
mov ds, ax  
mov KEEP\_PSP, es

```
push es
call IS_LOADED
cmp ax, 0h
jne check_cmd_un
```

```
call LOAD_CUSTOM_INTERRUPTION
pop es
jmp exit
```

```
check_cmd_un:
    pop es
    call CHECK_CMD
    cmp ax, 0h
    je already_loaded
    call UNLOAD_CUSTOM_INTERRUPTION
    jmp exit
```

```
already_loaded:
    mov dx,offset int_already_loaded
    call PRINT
```

```
exit:
    xor al,al
    mov ah,4ch
    int 21h
MAIN endp
```

```
CODE ends
end main
```