

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 обработчика
прерываний

Студентка гр. 0382

Здобнова К.Д.

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.

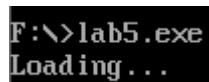
Шаг 5. Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

Шаг 6. Ответьте на контрольные вопросы.

Выполнение работы.


В первую очередь напишем и отладим программный модуль для лабораторной работы, выполняющий функции, аналогичные, как в лабораторной работе №4: проверка установки прерывания, установка прерывания, очистка прерывания, вывод информации о нем.

Запустим модуль и убедившись в установке прерывания - проверим его работоспособность, для этого введем клавиши 'q', 'w' и 'e', и при попытке увидим, что вместо них в строке записано 123.



```
F:\>lab5.exe
Loading...
```

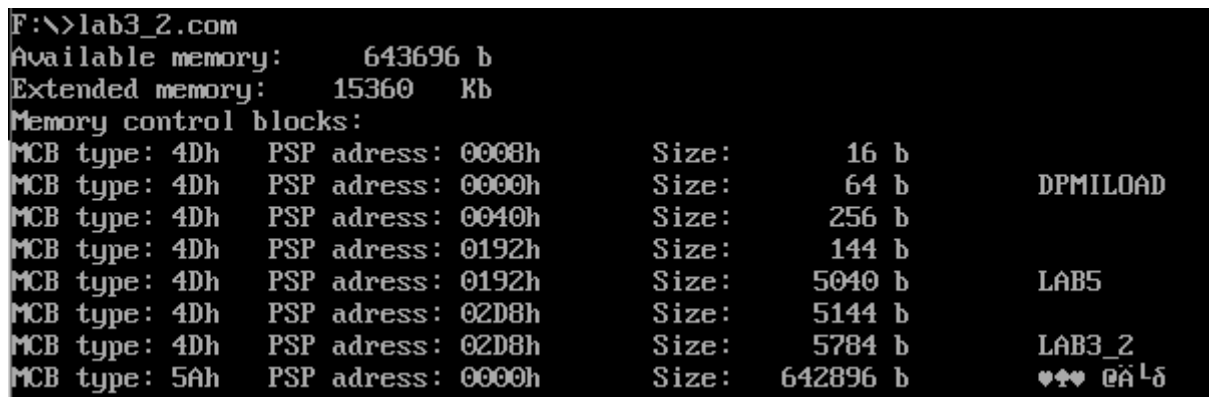
Рисунок 1.



```
F:\>123_
```

Рисунок 2.

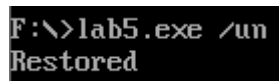
Размещение в памяти определим, благодаря лабораторной работе №3 и увидим наш работающий резидент.



```
F:\>lab3_2.com
Available memory: 643696 b
Extended memory: 15360 Kb
Memory control blocks:
MCB type: 4Dh PSP adress: 0008h Size: 16 b
MCB type: 4Dh PSP adress: 0000h Size: 64 b DPMILOAD
MCB type: 4Dh PSP adress: 0040h Size: 256 b
MCB type: 4Dh PSP adress: 0192h Size: 144 b
MCB type: 4Dh PSP adress: 0192h Size: 5040 b LAB5
MCB type: 4Dh PSP adress: 02D8h Size: 5144 b
MCB type: 4Dh PSP adress: 02D8h Size: 5784 b LAB3_2
MCB type: 5Ah PSP adress: 0000h Size: 642896 b ♥♥♥ @ÄLδ
```

Рисунок 3.

Выгрузим резидентный обработчик прерывания и проверим, что 'q', 'w', 'e' выводятся изначальным образом.



```
F:\>lab5.exe /un
Restored
```

Рисунок 4.



```
F:\>qwe_
```

Рисунок 5.

Также, воспользуемся снова программой из лабораторной работы №3 для проверки.

```

F:\>lab3_2.com
Available memory:      648912 b
Extended memory:      15360  Kb
Memory control blocks:
MCB type: 4Dh  PSP adress: 0008h      Size:      16 b
MCB type: 4Dh  PSP adress: 0000h      Size:       64 b      DPMILOAD
MCB type: 4Dh  PSP adress: 0040h      Size:     256 b
MCB type: 4Dh  PSP adress: 0192h      Size:     144 b
MCB type: 4Dh  PSP adress: 0192h      Size:     784 b      LAB3_2
MCB type: 5Ah  PSP adress: 0000h      Size:   648112 b      &P .î_П

```

Рисунок 6.

Как видим, обработчик выгружен.

Ответы на контрольные вопросы:

1) Какого типа прерывания использовались в работе?

Аппаратное прерывание int 09h, при нажатиях и отпусканиях клавиш
(прерывание функций BIOS)

Программное прерывание int 21h – для предоставления программисту
различных услуг со стороны DOS (прерывание функций DOS)

2) Чем отличается скан-код от кода ASCII

Код ASCII – это код символа из таблицы ASCII. Скан-код – код, присвоенный
каждой клавише с помощью которого драйвер распознает, какая клавиша была
нажата.

Выводы.

В процессе выполнения лабораторной работы были получены знания о
встраивании собственного прерывания в обработчик клавиатуры. Была
реализована программа, анализирующая скан-коды, изменяющая результат
нажатия на клавиатуру.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab5.asm

```
CODE SEGMENT
ASSUME  CS:CODE, DS:DATA, SS:MY_STACK

MY_STACK SEGMENT STACK
    dw 256 dup(0)
MY_STACK ENDS

DATA SEGMENT
    NotSet db "Did not load", 0dh, 0ah, '$'
    IsRestored db "Restored", 0dh, 0ah, '$'
    IsLoaded db "Already load", 0dh, 0ah, '$'
    Loading db "Loading...", 0dh, 0ah, '$'
    IS_LOAD db 0
    IS_UN db 0
DATA ENDS

;Начало прерывания
inter PROC FAR
    jmp start

interdata:
;Здесь будем хранить информацию об измененных значениях
    keyInf db 0
    SIGN dw 6666h
    KEEP_IP dw 0
    KEEP_CS dw 0
    KEEP_PSP dw 0
    KEEP_AX dw 0
    KEEP_SS dw 0
    KEEP_SP dw 0
    BUFF dw 256 dup(0)

start:
    mov KEEP_AX, ax
    mov KEEP_SP, sp
    mov KEEP_SS, ss
    mov ax, seg BUFF
    mov ss, ax
    mov ax, offset BUFF
    add ax, 256
    mov sp, ax

    push ax
    push bx
    push cx
    push dx
    push si
    push es
    push ds
    mov ax, seg keyInf
    mov ds, ax

    in al, 60h
```

```

        cmp al, 10h
        je key_1
        cmp al, 11h
        je key_2
        cmp al, 12h
        je key_3

        pushf
        call dword ptr cs:KEEP_IP
        jmp inter_end

key_1:
        mov keyInf, '1'
        jmp next
key_2:
        mov keyInf, '2'
        jmp next
key_3:
        mov keyInf, '3'

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:
        mov ah, 05h
        mov cl, keyInf
        mov ch, 00h
        int 16h
        or     al, al
        jz     inter_end
        mov ax, 0040h
        mov es, ax
        mov ax, es:[1ah]
        mov es:[1ch], ax
        jmp print

inter_end:
        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

```

```

inter ENDP
_end:

isSet PROC
    push ax
    push bx
    push si

    mov ah, 35h
    mov al, 09h
    int 21h
    mov si, offset SIGN
    sub si, offset inter
    mov ax, es:[bx + si]
    cmp ax, SIGN
    jne isSetCheck
    mov is_load, 1

    isSetCheck:
        pop si
        pop bx
        pop ax
        ret
isSet ENDP

loadInt PROC
    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 ax, seg inter
    mov dx, offset inter
    mov ds, ax
    mov ah, 25h
    mov al, 09h
    int 21h
    pop ds

    mov dx, offset _end
    mov cl, 4h
    shr dx, cl
    add dx, 10fh
    inc dx
    xor ax, ax
    mov ah, 31h
    int 21h

    pop es
    pop dx
    pop cx
    pop bx
    pop ax

```



```

        ret
loadInt  ENDP

UNloadInt 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 inter
    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

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

    ret
UNloadInt ENDP

checkUN  PROC
    push ax
    push es

    mov ax, KEEP_PSP
    mov es, ax
    cmp byte ptr es:[82h], '/'
    jne UNend
    cmp byte ptr es:[83h], 'u'
    jne UNend

```

```

        cmp byte ptr es:[84h], 'n'
        jne UNend
        mov is_un, 1

UNend:
        pop es
        pop ax
        ret
checkUN ENDP

printRes PROC near
        push ax
        mov ah, 09h
        int 21h
        pop ax
        ret
printRes ENDP

main PROC
        push ds
        xor ax, ax
        push ax
        mov ax, data
        mov ds, ax
        mov KEEP_PSP, es

        call isSet

        call checkUN

        cmp is_un, 1
        je UnloadCheck
        mov al, is_load
        cmp al, 1
        jne NotSetCheck
        mov dx, offset IsLoaded

        call printRes

        jmp ExitCheck

NotSetCheck:
        mov dx, offset Loading

        call printRes

        call loadInt

        jmp ExitCheck

UnloadCheck:
        cmp is_load, 1
        jne NotLoadedCheck
        mov dx, offset IsRestored

        call printRes

        call UNloadInt

        jmp ExitCheck

```

```
NotLoadedCheck:
    mov  dx, offset NotSet

    call printRes

ExitCheck:
    xor  al, al
    mov  ah, 4ch
    int  21h
main ENDP

CODE ENDS

    END main
```