

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

Студентка гр.0382

Ильин Д.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.

1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет те же функции, как в программе ЛР 4, а именно:

- проверяет, установлено ли пользовательское прерывание с вектором 1Ch;
- устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено, и осуществляется выход по функции 4Ch прерывания int 21h;
- 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. Длина кода сигнатуры должна быть достаточной, чтобы сделать случайное совпадение маловероятным.

Программа должна содержать код устанавливаемого прерывания в виде удаленной процедуры. Этот код и будет работать после установки при возникновении прерывания. Он должен выполнять следующие функции:

- сохранить значения регистров в стеке при входе и восстановить их при выходе;
- при выполнении тела процедуры анализируется скан-код;
- если этот код совпадает с одним из заданных, то требуемый код записывается в буфер клавиатуры;
- если этот код не совпадает ни с одним из заданных, то осуществляется передача управления стандартному обработчику прерывания.

2. Запустите отлаженную программу и убедитесь, что резидентный обработчик прерывания 09h установлен. Работа прерывания проверяется введением различных символов, обрабатываемых установленным обработчиком и стандартным обработчиком.

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.

4. Запустите отлаженную программу еще раз и убедитесь, что программа определяет установленный обработчик прерываний. Полученные результаты поместите в отчет.

5. Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, т.е. сообщения на экран не выводятся, а память, занятая резидентом, освобождена. Для этого также следует запустить программу ЛР3. Полученные результаты поместите в отчет.

6. Ответьте на контрольные вопросы.

Выполнение работы.

1. Был изменен .EXE модуль ЛР 4 для новых условий, а именно: изменена процедура прерывания MY_INT, изменяется вектор прерывания 09h, а не 1Ch.

2. После запуска модуля были нажаты следующие клавиши: left ALT, e, f, z, space, k. Результат представлен на рисунке 1:

```
C:\>lb5
Interruption was loaded
C:\>_RfX k_
```

Рисунок 1 – результат первого шага

3. Для выполнения данного шага воспользуемся программой lb3_1 из предыдущей лабораторной работы. Результат представлен на рисунке 2. Как можно заметить, обработчик прерывания действительно загружен в память.

```
C:\>lb3_1.com
Available memory: 647872
Extended memory: 245760
MCB num 1, MCB address: 016Fh, PCP address: 0008h, size: 16, SC/SD:
MCB num 2, MCB address: 0171h, PCP address: 0000h, size: 64, SC/SD:
MCB num 3, MCB address: 0176h, PCP address: 0040h, size: 256, SC/SD:
MCB num 4, MCB address: 0187h, PCP address: 0192h, size: 144, SC/SD:
MCB num 5, MCB address: 0191h, PCP address: 0192h, size: 864, SC/SD: LB5
MCB num 6, MCB address: 01C8h, PCP address: 01D3h, size: 144, SC/SD:
MCB num 7, MCB address: 01D2h, PCP address: 01D3h, size: 647872, SC/SD: LB3_1
C:\>_
```

Рисунок 2 – Результаты второго шага

4. При повторном запуске программы действительно выводится сообщение о том, что обработчик уже установлен. Вывод сообщения представлен на рисунке 3.

```
C:\>lb5
Interruption was loaded

C:\>lb5
Interruption is already loaded

C:\>
```

Рисунок 3 – Результаты третьего шага

5. Вызовем программу с параметром /up. Запустим программу прошлой лабораторной работы. Результат представлен на рисунке 4. Как можно заметить, при нажатии клавиш f, left ALT, e, r, z символы выводятся стандартно, а память освобождена.

```

C:\>lb5
Interruption was loaded

C:\>lb5 /un
Interruption was unloaded

C:\>lb3_1
Available memory: 648912
Extended memory: 245760
MCB num 1, MCB adress: 016Fh, PCP adress: 0008h, size: 16, SC/SD:
MCB num 2, MCB adress: 0171h, PCP adress: 0000h, size: 64, SC/SD:
MCB num 3, MCB adress: 0176h, PCP adress: 0040h, size: 256, SC/SD:
MCB num 4, MCB adress: 0187h, PCP adress: 0192h, size: 144, SC/SD:
MCB num 5, MCB adress: 0191h, PCP adress: 0192h, size:648912, SC/SD: LB3_1
C:\>ferz

```

Рисунок 4 – Результаты четвертого шага

Исходный код программы см. в приложении А.

Ответы на 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?

Скан-код – это специальный код, присвоенный каждой клавише, с помощью которого драйвер клавиатуры распознает нажатую клавишу. А ASCII код – это численное представление символов в таблице ASCII (например, ‘a’, ‘b’ и др. имеют и скан-коды, и ASCII-коды, т. к. это и символы, и клавиши, но ‘space’, ‘enter’ и др. – лишь клавиши, у них есть только скан-коды)

Выводы.

В ходе работы был написан собственный обработчик прерываний клавиатуры, а также была реализована установка и выгрузка данного обработчика.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb5.asm

```
AStack SEGMENT STACK
    DW 128 DUP(?)
AStack ENDS

DATA SEGMENT
    NOT_LOAD db 'Interruption did not load', 0DH, 0AH, '$'
    LOAD db 'Interruption was loaded', 0DH, 0AH, '$'
    UNLOAD db 'Interruption was unloaded', 0DH, 0AH, '$'
    ALREADY_LOAD db 'Interruption is already loaded', 0DH,
0AH, '$'
DATA ENDS

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

;-----
; ПРОЦЕДУРЫ
;-----
PRINT PROC near
    push AX
    mov AH, 09h
    int 21h
    pop AX
    ret
PRINT ENDP
;-----
MY_INT PROC far
    jmp handle
    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
    signature dw 9871h
    IStack db 50 dup(" ")
handle:
    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 handle
```

```

    push AX
    push BX
    push CX
    push DX

    in AL, 60h
    cmp AL, 38h
    je space
    cmp AL, 2Ch
    je z_key
    cmp AL, 12h
    je e_key

    pushf
    call dword ptr CS:KEEP_IP
    jmp exit_int

space:
    mov KEY_SYM, '_'
    jmp next_key
z_key:
    mov KEY_SYM, 'X'
    jmp next_key
e_key:
    mov KEY_SYM, 'R'

next_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 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
end_int:
MY_INT ENDP
;-----
MY_INT_LOAD PROC near
    mov PSP, ES
    mov AH, 35h
    mov AL, 09h
    int 21h
    mov KEEP_IP, BX
    mov KEEP_CS, ES

    push DS
    mov DX, offset MY_INT
    mov AX, seg MY_INT
    mov DS, AX
    mov AH, 25h
    mov AL, 09h
    int 21h
    pop DS

    mov DX, offset end_int
    mov CL, 4
    shr DX, CL
    inc DX
    mov AX, CS
    sub AX, PSP
    add DX, AX
    mov AL, 0
    mov AH, 31h
    int 21h
    ret
MY_INT_LOAD ENDP
;-----
MY_INT_UNLOAD PROC near
    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:[PSP]
        mov ES, AX
        push ES
        mov AX, ES:[2Ch]
        mov ES, AX
        mov AH, 49h
        int 21h
        pop ES
        int 21h
        ret
MY_INT_UNLOAD ENDp
;-----
IS_LOADED PROC near
        push BX
        push ES
        mov AH, 35h
        mov AL, 09h
        int 21h

        mov AX, ES:[signature]
        cmp AX, 9871h
        je loaded
        mov AL, 0h
        jmp end_isloaded
loaded:
        mov AL, 01h
end_isloaded:
        pop ES
        pop BX
        ret
IS_LOADED ENDP
;-----
IS_FLAG PROC near
        push BP
        mov BP, 0082h

        mov AL, ES:[BP]
        cmp AL, '/'
        jne not_good

        mov AL, ES:[BP+1]
        cmp AL, 'u'
        jne not_good

        mov AL, ES:[BP+2]
        cmp AL, 'n'

```

```

        jne not_good

        mov AL, 01h
        jmp good
not_good:
        mov AL, 0h
good:
        pop BP
        ret
IS_FLAG endp
;-----
MAIN PROC far
        mov ax, data
        mov ds, ax

        call IS_FLAG
        mov BX, AX

        call IS_LOADED
        cmp AL, 0h
        je not_loaded
        cmp BL, 0h
        jne int_unload
        mov DX, offset ALREADY_LOAD
        call PRINT
        jmp end_main

not_loaded:
        cmp BL, 0h
        je int_load
        mov DX, offset NOT_LOAD
        call PRINT
        jmp end_main

int_load:
        mov DX, offset LOAD
        call PRINT
        call MY_INT_LOAD
        jmp end_main

int_unload:
        mov AH, 35h
        mov AL, 09h
        int 21h
        mov DX, offset UNLOAD
        call PRINT
        call MY_INT_UNLOAD

end_main:
        xor AL, AL
        mov AH, 4Ch
        int 21h
MAIN ENDP

```

TESTPC ENDS
END MAIN