

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

Санников 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.

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 ЛРЗ. Полученные результаты поместите в отчет.

6) Ответьте на контрольные вопросы.

Ход работы.

Для выполнения лабораторной работы был написан .EXE модуль, который содержит следующие процедуры:

1)INTERRUPT — обработчик прерывания.

2)IS_LOADED — процедура, которая проверяет загрузку обработчика прерывания в память.

3)LOAD_INT — загружает обработчик прерывания в память.

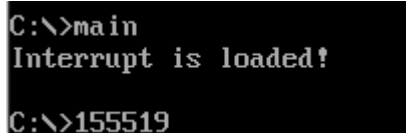
4)UNLOAD_INT — процедура, которая выгружает обработчик прерывания из памяти.

5)CHECK_CMD_KEY — процедура для определения флага /n командной строки.

Выполнение пунктов:

1) Был изменен обработчик прерывания: изменяется вектор прерывания 09h, а не 1Ch. При нажатии на клавишу F1 печатается цифра 1, при нажатии на F2 печатается цифра 5, при нажатии ALT + T печатается цифра 9.

2) Пример запуска модуля см. на Рисунке 1.



```
C:\>main
Interrupt is loaded!
C:\>155519
```

Рисунок 1 — Результат работы обработчика прерываний (два раза нажата F1, три раза F2 и один раз ALT + T).

3) Чтобы выполнить данный шаг, потребовался модуль .com из третьей лабораторной работы. Результат его работы см. на Рисунке 2.

```

C:\>task_1
Available memory: 647840
Extended memory: 246720
MCB: 1 | addr: 016F | owner PSP0008 | size: 16 | SD/SC:
MCB: 2 | addr: 0171 | owner PSP0000 | size: 64 | SD/SC:
MCB: 3 | addr: 0176 | owner PSP0040 | size: 256 | SD/SC:
MCB: 4 | addr: 0187 | owner PSP0192 | size: 144 | SD/SC:
MCB: 5 | addr: 0191 | owner PSP0192 | size: 896 | SD/SC: MAIN
MCB: 6 | addr: 01CA | owner PSP01D5 | size: 144 | SD/SC:
MCB: 7 | addr: 01D4 | owner PSP01D5 | size:647840 | SD/SC: TASK_1

```

Рисунок 2 — Обработчик загружен в память.

Как видно из данного рисунка, обработчик прерывания находится в памяти.

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

```

C:\>main
Interrupt is loaded!

C:\>main
Interrupt is already loaded!

C:\>main
Interrupt is already loaded!

```

Рисунок 3 — Сообщение об уже загруженном обработчике.

5) Теперь выгрузим обработчик и проверим память. См. Рисунок 4.

```

C:\>main
Interrupt is loaded!

C:\>main /un
Interrupt is unloaded!

C:\>task_1
Available memory: 648912
Extended memory: 246720
MCB: 1 | addr: 016F | owner PSP0008 | size: 16 | SD/SC:
MCB: 2 | addr: 0171 | owner PSP0000 | size: 64 | SD/SC:
MCB: 3 | addr: 0176 | owner PSP0040 | size: 256 | SD/SC:
MCB: 4 | addr: 0187 | owner PSP0192 | size: 144 | SD/SC:
MCB: 5 | addr: 0191 | owner PSP0192 | size:648912 | SD/SC: TASK_1

```

Рисунок 4 — Результат выгрузки обработчика из памяти.

Как видно из данного рисунка, обработчик выгружен из памяти.

Исходный код программы см в приложении А.

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?

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)

Вывод.

В ходе данной лабораторной работы был реализован собственный обработчик прерываний клавиатуры, его установка и выгрузка из памяти компьютера.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.asm

```
ASTACK SEGMENT STACK
    DB 256 DUP(?)
ASTACK ENDS

DATA SEGMENT
    FLAG db 0
    LOADED_MES db 'Interrupt is loaded!', 0DH, 0AH, '$'
    UNLOADED_MES db 'Interrupt is unloaded!', 0DH, 0AH, '$'
    ALREADY_LOADED_MES db 'Interrupt is already loaded!', 0DH,
0AH, '$'
DATA ENDS

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

    INTERRUPT PROC FAR
        jmp int_start
        PSP DW ?
        KEEP_IP DW ?
        KEEP_CS DW ?
        KEEP_SS DW ?
        KEEP_SP DW ?
        KEEP_AX DW ?
        SYMBOL DW ?
        INT_ID DW 1f17h
        INT_STACK DB 128 dup(?)

        int_start:
            mov KEEP_SS, SS
            mov KEEP_SP, SP
            mov SP, SEG INTERRUPT
            mov SS, SP
            mov SP, OFFSET int_start
            push ax
            push bx
            push cx
            push dx

            in al, 60h
            cmp al, 3bh
            je if_f1
            cmp al, 3ch
            je if_f2
            cmp al, 14h
            je if_T
            call dword ptr CS:KEEP_IP
```

```

        jmp int_end

if_f1:
        mov SYMBOL, '1'
        jmp next_key
if_f2:
        mov SYMBOL, '5'
        jmp next_key
if_T:
        mov SYMBOL, '9'

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

int_end:
        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

        if_end_byte:
INTERRUPT ENDP

IS_LOADED PROC NEAR

```



```

    push AX
    push BX
    push DX
    push SI

    mov FLAG, 1
    mov AH, 35h
    mov AL, 1Ch
    int 21h
    mov SI, OFFSET INT_ID
    sub SI, OFFSET INTERRUPT
    mov DX, ES:[BX+SI]
    cmp DX, 1f17h
    je if_loaded
    mov FLAG, 0

    if_loaded:
        pop SI
        pop DX
        pop BX
        pop AX
    ret
IS_LOADED ENDP

LOAD_INT PROC NEAR
    push AX
    push BX
    push CX
    push DX
    push DS
    push ES

    MOV AH, 35h
    MOV AL, 1Ch
    int 21h
    MOV KEEP_IP, BX
    MOV KEEP_CS, ES

    mov DX, offset INTERRUPT
    mov AX, seg INTERRUPT
    mov DS, AX
    mov AH, 25h
    mov AL, 1Ch
    int 21h

    mov DX, offset if_end_byte
    mov CL, 4
    shr DX, CL
    inc DX
    mov AX, CS
    sub AX, PSP
    add DX, AX

```

```

        xor AX, AX
        mov AH, 31h
        int 21h

        pop ES
        pop DS
        pop DX
        pop CX
        pop BX
        pop AX
        ret
LOAD_INT ENDP

UNLOAD_INT PROC NEAR
        push AX
        push BX
        push DX
        push DS
        push ES

        cli
        mov AH, 35h
        mov AL, 1Ch
        int 21h
        mov DX, ES:[offset KEEP_IP]
        mov AX, ES:[offset KEEP_CS]
        mov DS, AX
        mov AH, 25h
        mov AL, 1Ch
        int 21h

        mov AX, ES:[offset PSP]
        mov ES, AX
        mov DX, ES:[2Ch]
        mov AH, 49h
        int 21h
        mov ES, DX
        mov AH, 49h
        int 21h
        sti

        pop ES
        pop DS
        pop DX
        pop BX
        pop AX
        ret
UNLOAD_INT ENDP

CHECK_CMD_KEY PROC NEAR
        push AX

```

```

    mov FLAG, 0
    mov AL, ES:[82h]
    cmp AL, '/'
    jne if_no_key
    mov AL, ES:[83h]
    cmp AL, 'u'
    jne if_no_key
    mov AL, ES:[84h]
    cmp AL, 'n'
    jne if_no_key
    mov FLAG, 1

    if_no_key:
        pop AX
        ret
CHECK_CMD_KEY ENDP

PRINT PROC NEAR
    push AX
    mov AH, 09h
    int 21h
    pop AX
    ret
PRINT ENDP

main PROC FAR
    push DS
    xor AX, AX
    mov AX, DATA
    mov DS, AX

    mov PSP, ES
    call CHECK_CMD_KEY
    cmp FLAG, 1
    je int_unload

    call IS_LOADED
    cmp FLAG, 0
    je int_load
    mov DX, OFFSET ALREADY_LOADED_MES
    call PRINT
    jmp exit

    int_load:
        mov DX, OFFSET LOADED_MES
        call PRINT
        call LOAD_INT
        jmp exit

    int_unload:
        call IS_LOADED
        cmp FLAG, 0

```

```

        je unloaded
        call UNLOAD_INT

unloaded:
        mov DX, OFFSET UNLOADED_MES
        call PRINT

exit:
        pop DS
        mov AX, 4Ch
        int 21h

main ENDP
CODE ENDS
END main

```