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. 7381 _____ Минуллин М.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2019

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.

Порядок выполнения работы

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

2) Если прерывание не установлено, то установить резидентную функцию для обработки прерывания и настроить вектор прерываний. Адрес точки входа в стандартный обработчик прерывания находится в теле пользовательского обработчика. Осуществить выход по функции 4Ch прерывания int 21h.

3) Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h.

Ход работы

Состояние памяти до запуска lab5.exe представлено на Рис.1 (использовалась программа lab3_1.com):

```
C:\>LAB3_1.COM
Number of available memory: 648912 bytes
Extended memory size: 15360 Kbytes
Memory control circuitry:
 ADDRESS ¦ OWNER ¦   SIZE  ¦ NAME
 016F      0008        16
 0171      0000        64
 0176      0040       256
 0187      0192       144
 0191      0192      6432    LAB3_1
 0324      0000    642464
```

Рисунок 1 – Результат работы программы lab3_1.com

Запуск программы lab5.exe представлен на Рис.2:

```
C:\>LAB5.EXE
interrupt handler is installed
```

Рисунок 2 – Установка пользовательского обработчика прерывания

Проверим загрузку пользовательского обработчика и его работу. При нажатии клавиши пробела выводится стрелочка (см. Рис.3):

```
C:\>interrupt→handler→is→installed_
```

Рисунок 3 – Результат ввода различных символов

Исходный код программы представлен в приложении А.

Состояние памяти после установки пользовательского обработчика прерывания представлено на рис.4:

```
C:\>LAB3_1.COM
Number of available memory: 647360 bytes
Extended memory size: 15360 Kbytes
Memory control circuitry:
 ADDRESS ¦ OWNER ¦   SIZE  ¦ NAME
 016F      0008        16
 0171      0000        64
 0176      0040       256
 0187      0192       144
 0191      0192      1376    LAB5
 01E8      01F3       144
 01F2      01F3      7984    LAB3_1
 03E6      0000    639360
```

Рисунок 4 – Результат работы программы lab3_1.com после запуска lab5.exe

Выгрузка пользовательского обработчика прерывания с ключом /un (см. Рис.5):

```
C:\>LAB5.EXE/un
interrupt handler is removed
```

Рисунок 5 – Выгрузка пользовательского обработчика прерывания

Состояние памяти после выгрузки пользовательского обработчика прерывания представлено на Рис.6:

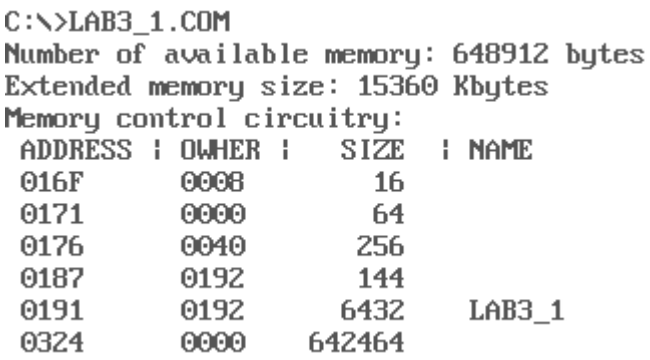

Рисунок 6 – Результат выполнения программы lab3_1.com

Выводы

В процессе выполнения данной лабораторной работы была исследована возможность встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

Контрольные вопросы.

В: Какого типа прерывания использовались в работе?

О: в работе использовались программное (int 21h) и аппаратные (int 09h и 16h) прерывания.

В: Чем отличается скан код от кода ASCII?

О: скан код – в IBM-совместимых компьютерах код, присвоенный каждой клавише, с помощью которого драйвер клавиатуры распознает, какая клавиша была нажата. При нажатии любой клавиши контроллер клавиатуры распознаёт клавишу и посылает её скан-код в порт 60h.

ПРИЛОЖЕНИЕ А
ТЕКСТ ИСХОДНОЙ ПРОГРАММЫ

```
ASTACK SEGMENT STACK
    dw 100h dup (?)
ASTACK ENDS

CODE SEGMENT
 ASSUME CS:CODE, DS:DATA, ES:DATA, SS:ASTACK
; ⍰⍰Ž–…„“⍰›
;---------------------------------------
ROUT PROC FAR

    jmp go_
    SIGNATURA   DW 0ABCDh
    KEEP_PSP    DW 0
    KEEP_IP     DW 0
    KEEP_CS     DW 0
    INT_STACK   DW  100 dup (?)
    KEEP_SS     DW 0
    KEEP_AX     DW  ?
    KEEP_SP     DW 0

    go_:
    mov KEEP_SS, SS
    mov KEEP_SP, SP
    mov KEEP_AX, AX
    mov AX, seg INT_STACK
    mov SS, AX
    mov SP, 0
    mov AX, KEEP_AX

    push AX
    push ES
    push DS
    push DX
    push DI
    push CX
    mov AL, 0
    in AL, 60h
    cmp AL, 39h ; space
    je do_req
```

```
pushf
call dword ptr CS:KEEP_IP
jmp skip

do_req:
in AL, 61h
mov AH, AL
or AL, 80h
out 61h, AL
xchg AH, AL
out 61h, AL
mov AL, 20h
out 20h, AL

buf_push:
mov AL,0
mov AH, 05h
mov CL, 1Ah
mov CH, 00h
int 16h
or AL, AL
jz skip
mov AX, 0040h
mov ES, AX
mov AX, ES:[1Ah]
mov ES:[09h], AX
jmp buf_push
skip:

pop CX
pop DI
pop DX
pop DS
pop ES
mov AL, 20h
out 20h, AL
pop AX

mov AX, KEEP_SS
mov SS, AX
mov AX, KEEP_AX
mov SP, KEEP_SP

iret
```

```
ROUT ENDP
LAST_BYTE:
;---------------------------------------
PRINT PROC
    push AX
    mov AH, 09h
    int 21h
    pop AX
    ret
PRINT ENDP
;---------------------------------------
PROV_ROUT PROC
    mov AH, 35h
    mov AL, 09h
    int 21h
    mov SI, offset SIGNATURA
    sub SI, offset ROUT
    mov AX, 0ABCDh
    cmp AX, ES:[BX+SI]
    je ROUT_EST
    call SET_ROUT
    jmp PROV_KONEC
    ROUT_EST:
    call DEL_ROUT
    PROV_KONEC:
    ret
PROV_ROUT ENDP
;---------------------------------------
SET_ROUT PROC
    mov AX, KEEP_PSP
    mov ES, AX
    cmp byte ptr ES:[80h],0
    je UST
    cmp byte ptr ES:[82h],'/'
    jne UST
    cmp byte ptr ES:[83h],'u'
    jne UST
    cmp byte ptr ES:[84h],'n'
    jne UST

    mov DX, offset INT_NOT_INST
    call PRINT
    ret
```

```
    UST:
    call SAVE_STAND

    mov DX, offset INT_INST
    call PRINT

    push DS
    mov DX, offset ROUT
    mov AX, seg ROUT
    mov DS, AX

    mov AH, 25h
    mov AL, 09h
    int 21h
    pop DS

    mov DX, offset LAST_BYTE
    mov CL, 4
    shr DX, CL
    add DX, 1
    add DX, 40h

    mov AL, 0
    mov AH, 31h
    int 21h

    ret
SET_ROUT ENDP
;-----------------------------------------
DEL_ROUT PROC
    push DX
    push AX
    push DS
    push ES


    mov AX, KEEP_PSP
    mov ES, AX
    cmp byte ptr ES:[82h],'/'
    jne UDAL_KONEC
    cmp byte ptr ES:[83h],'u'
    jne UDAL_KONEC
    cmp byte ptr ES:[84h],'n'
    jne UDAL_KONEC
```

```
    mov DX, offset INT_RM
    call PRINT

    CLI

    mov AH, 35h
    mov AL, 09h
    int 21h
    mov SI, offset KEEP_IP
    sub SI, offset ROUT

    mov DX, ES:[BX+SI]
    mov AX, ES:[BX+SI+2]
    mov DS, AX
    mov AH, 25h
    mov AL, 09h
    int 21h

    mov AX, ES:[BX+SI-2]
    mov ES, AX
    mov AX, ES:[2ch]
    push ES
    mov ES, AX
    mov AH, 49h
    int 21h
    pop ES
    mov AH, 49h
    int 21h

    STI
    jmp UDAL_KONEC2

    UDAL_KONEC:
    mov DX, offset INT_ALRD_INST
    call PRINT
    UDAL_KONEC2:

    pop ES
    pop DS
    pop AX
    pop DX
    ret
DEL_ROUT ENDP
```

```
    ;---------------------------------------
    SAVE_STAND PROC
        push AX
        push bx
        push ES
        mov AH,35h
        mov AL,09h
        int 21h
        mov KEEP_CS, ES
        mov KEEP_IP, BX
        pop ES
        pop BX
        pop AX
        ret
    SAVE_STAND ENDP
    ;---------------------------------------
    BEGIN:
        mov AX, DATA
        mov DS, AX
        mov KEEP_PSP, ES
        call PROV_ROUT
        xor AL, AL
        mov AH, 4Ch
        int 21H
    CODE ENDS

    DATA SEGMENT
        INT_INST        db 'interrupt handler is installed',0DH,0AH,'$'
        INT_RM          db 'interrupt handler is removed',0DH,0AH,'$'
        INT_ALRD_INST   db 'interrupt handler is already
installed',0DH,0AH,'$'
        INT_NOT_INST    db 'interrupt handler is not
installed',0DH,0AH,'$'
        STRENDL         db 0DH,0AH,'$'
    DATA ENDS
    END BEGIN
```