

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №5**  
**по дисциплине «Операционные системы»**  
**Тема: Сопряжение стандартного и пользовательского обработчика**  
**прерываний**

Студент гр. 8381

Преподаватель

\_\_\_\_\_

\_\_\_\_\_

Облизов А.Д.

Ефремов М.А.

Санкт-Петербург

2020

### **Цель работы.**

Исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

### **Выполнение работы.**

Написание работы производилось на базе операционной системы Windows 10 в редакторе Visual Code. Сборка, отладка производились на базе эмулятора DOSBox 0.74-3.

Был написан текст исходного .EXE модуля с именем INT.EXE. Описание процедур в программе представлено в табл. 1.

Таблица 1 – Описание процедур программы

Название	Назначение
PRINT_STRING	Вывод на экран строки, адрес которой содержится в DX
INTERRUPTION	Процедура обработчика прерывания. В процедуре производится: <ul style="list-style-type: none"><li>• Считывание ключа, его обработка или переход на исходный обработчик</li><li>• Записывание символов в буфер клавиатуры (функция 05h прерывания 16h)</li></ul>
INT_CHECK	Процедура проверки установки резидента INTERRUPTION
INT_LOAD	Процедура загрузки резидентной функции INTERRUPTION
INT_UNLOAD	Процедура загрузки резидентной функции INTERRUPTION (восстановление исходного обработчика прерывания системного таймера)
CL_CHECK	Процедура проверки параметра командной строки

Обработка нажатий клавиатуры (изменение записанных символов в буфер) представлена в табл. 2.

Таблица 2 – Обработка нажатий клавиатуры

Нажатая клавиша	«1»	«2»	«3»	«4»	«5»	«6»
Записанный в буфер символ	«S»	«E»	«C»	«R»	«E»	«T»

Зажатие клавиш Shift, CapsLock не влияет на изменение вывода символа.

Вид командной строки после первого запуска программы и последовательного нажатия клавиш «1», «2», «3», «4», «5», «6», «H», «E», «L», «L», «O» представлен на рис. 1.

```
D:\>int
D:\>SECRETHELLO_
```

Рисунок 1 – Выполнение INT.EXE

Вывод программы MEM.COM, которая предназначена для вывода информации о MCB блоках, после запуска INT.EXE, представлен на рис. 2.

```
D:\>mem.com
Available memory: 640 kbytes
Extended memory: 15360 kbytes
MCB number: 1
Block is MS DOS      Area size: 16

MCB number: 2
Block is free        Area size: 64

MCB number: 3
Block is 0040        Area size: 256

MCB number: 4
Block is 0192        Area size: 144

MCB number: 5
Block is 0192        Area size: 4512
INT
MCB number: 6
Block is 02B7        Area size: 144

MCB number: 7
Block is 02B7        Area size: 644224
MEM
D:\>_
```

Рисунок 2 – Выполнение MEM.COM после запуска INT.EXE

На рисунке видно, что процедура прерывания осталась резидентной в памяти и располагается в блоках 4 и 5.

Далее программа INT.EXE была запущена еще раз, в результате чего было выведено сообщение о том, что программа определила уже установленный обработчик прерывания, что представлено на рис. 3.

```
D:\>int
Interruption was already loaded
D:\>
```

Рисунок 3 – Повторный запуск INT.EXE

Далее программа INT.EXE была запущена с параметром командной строки “/un” для выгрузки резидентного обработчика прерываний, а также была запущена программа MEM.COM для вывода блоков MCB. После этого были последовательно нажаты клавиши: «1», «2», «3», «4», «5», «6», «H», «E», «L», «L», «O» Результат выполнения программы представлен на рис. 4.

```
D:\>int /un
D:\>mem.com
Available memory: 640 kbytes
Extended memory: 15360 kbytes
MCB number: 1
Block is MS DOS      Area size: 16
MCB number: 2
Block is free        Area size: 64
MCB number: 3
Block is 0040        Area size: 256
MCB number: 4
Block is 0192        Area size: 144
MCB number: 5
Block is 0192        Area size: 648912
MEM
D:\>123456HELLO_
```

Рисунок 4 – Выгрузка обработчика и выполнение MEM.COM

Из рисунка видно, что память для резидентного обработчика была освобождена (ранее он занимал блоки 4 и 5, теперь их занимает программа MEM.COM). Также видно, что обработчик закончил работу по выведенным символам: замена, которая была реализована в обработчике, не была произведена.

## **Контрольные вопросы.**

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

- Был написан обработчик прерывания от клавиатуры, которое является аппаратным
- В коде программы использовались программные прерывания, например, `int 21h`

### **2. Чем отличается скан-код и ASCII код?**

- ASCII код – код символа, необходимый для хранения символов в памяти и их печати на экран.
- Скан-код – код клавиши на клавиатуре, необходимый для распознавания нажатых клавиш драйвером клавиатуры.

## **Выводы.**

В ходе выполнения лабораторной работы была реализована программа, загружающая и выгружающая пользовательское прерывание от нажатия клавиатуры в память.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ. INT.ASM

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

```
INTERRUPTION  PROC    FAR
                jmp     INT_START
INT_DATA:
                INT_CODE      DW    3158h
```

```
                KEEP_IP      DW    0
                KEEP_CS      DW    0
                KEEP_PSP      DW    0

                SYMB          DB    0
```

```
INT_START:
                push  AX
                push  BX
                push  CX
                push  DX
                push  SI
                push  ES
                push  DS

                mov   AX, SEG SYMB
                mov   DS, AX

                in     AL, 60h
                cmp   AL, 02h
                je     OUT_S
                cmp   AL, 03h
                je     OUT_E
                cmp   AL, 04h
                je     OUT_C
                cmp   AL, 05h
                je     OUT_R
                cmp   AL, 06h
                je     OUT_E
                cmp   AL, 07h
                je     OUT_T

                pushf
                call  DWORD PTR CS:KEEP_IP
                jmp   INT_END
```

```

OUT_S:
    mov     SYMB, 'S'
    jmp     PROCESSING_SYMB
OUT_E:
    mov     SYMB, 'E'
    jmp     PROCESSING_SYMB
OUT_C:
    mov     SYMB, 'C'
    jmp     PROCESSING_SYMB
OUT_R:
    mov     SYMB, 'R'
    jmp     PROCESSING_SYMB
OUT_T:
    mov     SYMB, 'T'

```

```

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

```

```

WRITE_SYMB:
    mov     AH, 05h
    mov     CL, SYMB
    mov     CH, 00h
    int     16h
    or      AL, AL
    jz      INT_END

    mov     AX, 0040h
    mov     ES, AX
    mov     AX, ES:[1Ah]
    mov     ES:[1Ch], AX
    jmp     WRITE_SYMB

```

```

INT_END:
    pop     DS
    pop     ES
    pop     SI
    pop     DX
    pop     CX
    pop     BX
    pop     AX

```

```

        mov     AL, 20h
        out     20h, AL
        IRET

    ret
INTERRUPTION    ENDP
    LAST_BYTE:

INT_CHECK      PROC
    push     AX
    push     BX
    push     SI

    mov     AH, 35h
    mov     AL, 09h
    int     21h
    mov     SI, offset INT_CODE
    sub     SI, offset INTERRUPTION
    mov     AX, ES:[BX + SI]
    cmp     AX, INT_CODE
    jne     INT_CHECK_END
    mov     INT_LOADED, 1

INT_CHECK_END:
    pop     SI
    pop     BX
    pop     AX
    ret
INT_CHECK      ENDP

INT_LOAD      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 INTERRUPTION
    mov     DX, offset INTERRUPTION
    mov     DS, AX
    mov     AH, 25h
    mov     AL, 09h

```



```

        int      21h
        pop      DS

    mov     DX, offset LAST_BYTE
    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
INT_LOAD      ENDP

INT_UNLOAD    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 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

        pop     SI
        pop     ES
        pop     DS
        pop     DX
        pop     BX
        pop     AX

        ret
INT_UNLOAD      ENDP

CL_CHECK      PROC
        push    AX
        push    ES

        mov     AX, KEEP_PSP
        mov     ES, AX
        cmp     byte ptr ES:[82h], '/'
        jne     CL_CHECK_END
        cmp     byte ptr ES:[83h], 'u'
        jne     CL_CHECK_END
        cmp     byte ptr ES:[84h], 'n'
        jne     CL_CHECK_END
        mov     UN_CL, 1

        CL_CHECK_END:
        pop     ES
        pop     AX
        ret
CL_CHECK      ENDP

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

```

MAIN PROC

```
    push    DS
    xor     AX, AX
    push    AX
    mov     AX, DATA
    mov     DS, AX
    mov     KEEP_PSP, ES

    call    INT_CHECK
    call    CL_CHECK
    cmp     UN_CL, 1
    je      UNLOAD
    mov     AL, INT_LOADED
    cmp     AL, 1
    jne     LOAD
    mov     DX, offset WAS_LOADED_INFO
    call    PRINT_STRING
    jmp     MAIN_END

LOAD:
    call    INT_LOAD
    jmp     MAIN_END

UNLOAD:
    cmp     INT_LOADED, 1
    jne     NOT_EXIST
    call    INT_UNLOAD
    jmp     MAIN_END

NOT_EXIST:
    mov     DX, offset NOT_LOADED_INFO
    call    PRINT_STRING

MAIN_END:
    xor     AL, AL
    mov     AH, 4Ch
    int     21h

MAIN ENDP
```

CODE ENDS

```
ASTACK  SEGMENT STACK
        DW  128 dup(0)
ASTACK  ENDS
```

```
DATA    SEGMENT
        WAS_LOADED_INFO      DB  "Interrupt was already loaded", 10, 13, "$"
        NOT_LOADED_INFO      DB  "Interrupt is not loaded", 10, 13, "$"
        INT_LOADED            DB  0
        UN_CL                  DB  0
```

DATA      ENDS

END    MAIN