

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. 8383

Дейнега В.Е.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

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

Ход работы.

Был написан программный модуль типа .EXE, который выполняет следующие функции:

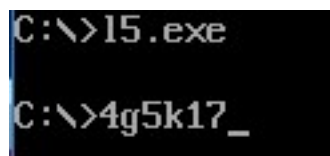
- 1) Проверяет, установлено ли пользовательское прерывание с вектором 09h.
- 2) Устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено, и осуществляет выход по функции 4Ch прерывания int 21h.
- 3) Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h.
- 4) 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.

Обработчик прерываний обрабатывает нажатия таким образом, что заменяет написанные буквы в стиль 1337 т.е. заменяет буквы на похожие цифры. В табл. 1 приведены примеры таких замен.

Таблица 1 – Обработка нажатий

Нажатая клавиша	a	t	i	s
Записанный в буфер символ	4	7	1	5

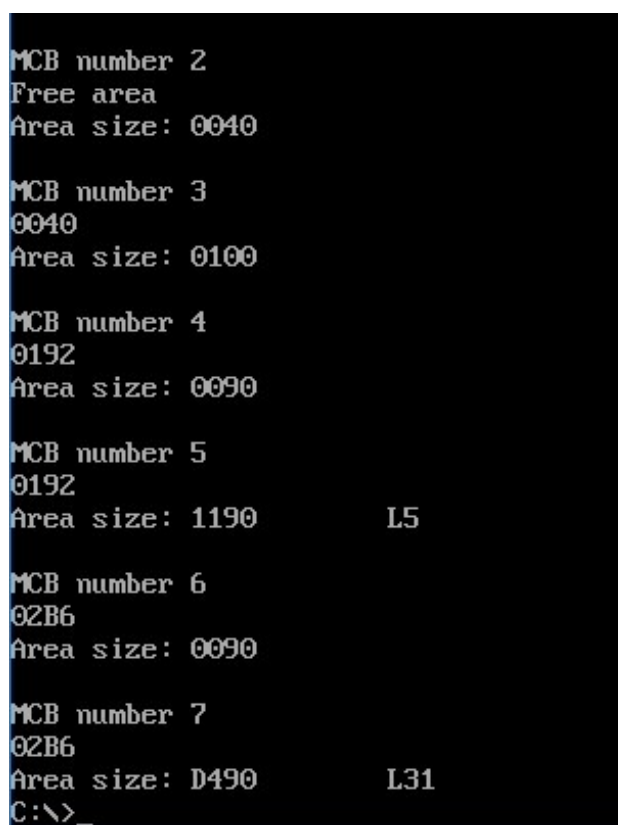
Демонстрация работы программы представлена на рис. 1.



```
C:\>l5.exe  
C:\>4g5k17_
```

Рисунок 1 – Ввод строки “agskit”

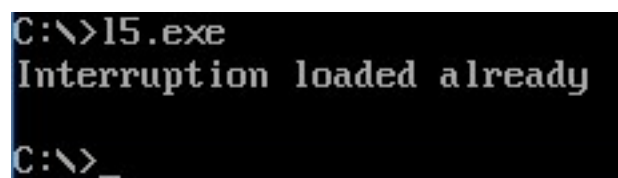
Вывод программы l31.com, предоставляющей информацию о МСВ блоках представлен на рис. 2.



```
MCB number 2  
Free area  
Area size: 0040  
  
MCB number 3  
0040  
Area size: 0100  
  
MCB number 4  
0192  
Area size: 0090  
  
MCB number 5  
0192  
Area size: 1190          L5  
  
MCB number 6  
02B6  
Area size: 0090  
  
MCB number 7  
02B6  
Area size: D490          L31  
C:\>_
```

Рисунок 2 – Выполнение l31.com после выполнения l5.exe

Попытка еще раз запустить l5.exe представлена на рис. 3.



```
C:\>l5.exe  
Interruption loaded already  
C:\>_
```

Рисунок 3 – Запуск l5.exe во второй раз

Далее программа была запущена с параметром /un, результат представлен на рис. 4.

```
C:\>l5.exe /un  
C:\>_
```

Рисунок 4 – Запуск l5.exe с параметром /un

С помощью программы из лабораторной №3 посмотрим информацию о блоках MCB, результат выполнения l31.com представлен на рис. 5.

```
C:\>l31.com  
Available memory: 648912 B  
Extended memory : 15360 KB  
  
MCB number 1  
Area belongs to MS DOS  
Area size: 0010  
  
MCB number 2  
Free area  
Area size: 0040  
  
MCB number 3  
0040  
Area size: 0100  
  
MCB number 4  
0192  
Area size: 0090  
  
MCB number 5  
0192  
Area size: E6D0      L31  
C:\>_
```

Рисунок 5 – Выполнение l31.com после выгрузки резидента

На рисунке 5 действительно видно, что память резидентного обработчика была освобождена, ранее он занимал блок 5, что видно на рис. 2.

Контрольные вопросы.

1. Какого типа прерывания использовались в работе?

- 1) Программные (int 21h).
- 2) Аппаратные (16h).

2. Чем отличается скан-код от ASCII кода?

Скан-код — код, присвоенный каждой клавише, с помощью которого драйвер клавиатуры распознает, какая клавиша была нажата.

ASCII код – код символа в таблице ASCII символов, необходимый для хранения символов в памяти и печати их на экран.

Выводы.

В ходе выполнения лабораторной работы была реализована программа, загружающая и выгружающая пользовательское прерывание от нажатия клавиш на клавиатуре.

ПРИЛОЖЕНИЕ

КОД ПРОГРАММЫ

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

```
INTER   PROC      FAR
        jmp      INTER_START
        SYMB      DB    0
        SIGNATURE DW    1234h
        KEEP_IP   DW    0
        KEEP_CS   DW    0
        KEEP_PSP  DW    0
```

```
INTER_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, 14h
        je      OUT_T

        cmp     AL, 17h
        je      OUT_I

;        cmp     AL, 18h
;        je      OUT_O

        cmp     AL, 1eh
        je      OUT_A

        cmp     AL, 1fh
        je      OUT_S

        pushf
        call    DWORD PTR CS:KEEP_IP
        jmp     INTER_REAL_END

OUT_A:
        mov     SYMB, '4'
        jmp     PROCESSING
OUT_I:
        mov     SYMB, '1'
```

```

        jmp PROCESSING
OUT_0:
        mov SYMB, '0'
        jmp PROCESSING
OUT_5:
        mov SYMB, '5'
        jmp PROCESSING
OUT_T:
        mov SYMB, '7'

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

INTER_REAL_END:
        pop     DS
        pop     ES
        pop     SI
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        mov     AL, 20h
        out     20h, AL
        IRET

        ret
INTER    ENDP
INTER_END:

INTER_CHECK    PROC
        push    AX
        push    BX
        push    SI

```

```

        mov     AH, 35h
        mov     AL, 09h
        int     21h
        mov     SI, offset SIGNATURE
        sub     SI, offset INTER
        mov     AX, ES:[BX + SI]
        cmp     AX, SIGNATURE
        jne     INTER_CHECK_END
        mov     IS_LOADED, 1

INTER_CHECK_END:
        pop     SI
        pop     BX
        pop     AX
    ret
INTER_CHECK     ENDP

INTER_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 INTER
        mov     DX, offset INTER
        mov     DS, AX
        mov     AH, 25h
        mov     AL, 09h
        int     21h
        pop     DS

    mov     DX, offset INTER_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

```



```

INTER_LOAD          ENDP

INTER_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 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
INTER_UNLOAD        ENDP

UN_CHECK            PROC
    push    AX
    push    ES

```

```

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

UN_CHECK_END:
        pop     ES
        pop     AX
        ret
UN_CHECK     ENDP

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

MAIN PROC
        push    DS
        xor     AX, AX
        push    AX
        mov     AX, DATA
        mov     DS, AX
        mov     KEEP_PSP, ES

        call    INTER_CHECK
        call    UN_CHECK
        cmp     IS_UN, 1
        je      UNLOAD
        mov     AL, IS_LOADED
        cmp     AL, 1
        jne     LOAD
        mov     DX, offset STR_LOADED_ALREADY
        call    WRITE_STR
        jmp     MAIN_END

LOAD:
        call    INTER_LOAD
        jmp     MAIN_END

UNLOAD:
        cmp     IS_LOADED, 1
        jne     NOT_EXIST
        call    INTER_UNLOAD
        jmp     MAIN_END

NOT_EXIST:
        mov     DX, offset STR_NOT_LOADED
        call    WRITE_STR

```

```

        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
        STR_LOADED_ALREADY DB "Interruption loaded already ",10,13,"$"
        STR_NOT_LOADED     DB "Interruption isn't loaded",10,13,"$"
        IS_LOADED          DB 0
        IS_UN              DB 0
DATA     ENDS

END      MAIN

```