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

Студентка гр. 8381 _____ Ивлева О.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

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

Выполнение работы.

Сборка, отладка производились на базе эмулятора DOSBox 0.74-3.

Таблица 1 – Описание процедур программы

Название	**Назначение**
WRITE	Вывод на экран строки, адрес которой содержится в DX
INTERRUPTION	Процедура обработчика прерывания.
INT_CHECK	Проверка установки резидента INTERRUPTION
INT_LOAD	Загрузка резидентной функции INTERRUPTION
INT_UNLOAD	Загрузка резидентной функции INTERRUPTION (восстановление исходного обработчика прерывания системного таймера)
CL_CHECK	Проверка параметра командной строки(“/UN”)

Обработка нажатий клавиатуры представлена в табл. 2.

Таблица 2 – Обработка нажатий клавиатуры

Нажатая клавиша	Q	W	E	R	T	Y	U	I	O	P	[
Записанный в буфер символ	P	R	I	M	I	T	E	L	A	B	U

Вид командной строки после первого запуска программы и последовательного нажатия клавиш [poiuytrewqasd представлен на рис. 1.

Рисунок 1 – Выполнение программы

На рис. 2 видно, что процедура прерывания осталась резидентной в памяти и располагается в блоке 4 и 5.

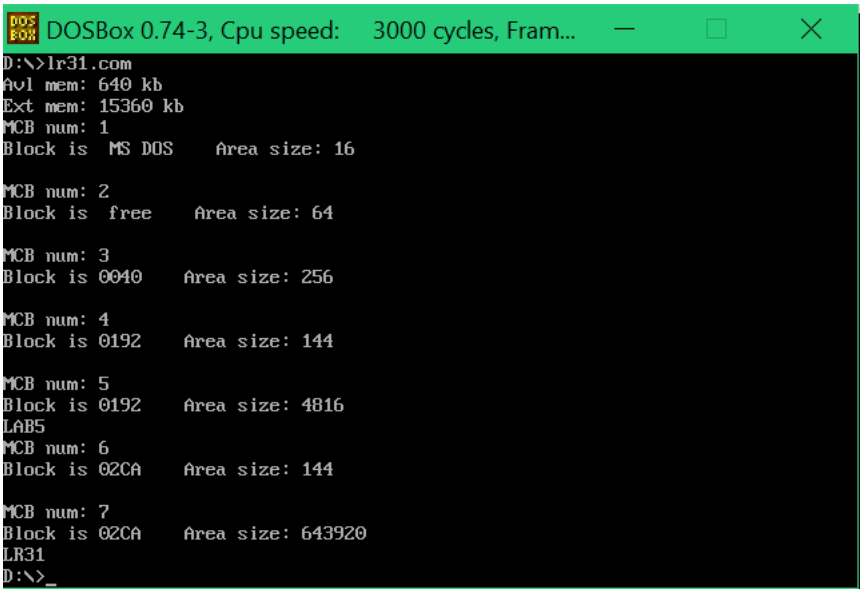

Рисунок 2 – Выполнение lr31.com после запуска lab5.EXE

Далее программа lab5.exe была запущена с параметром “/UN” для выгрузки резидентного обработчика прерываний, а таже была запущена программа lr31.com для вывода блоков MCB. Результат выполнения программы представлен на рис. 3. Видно, что память для резидентного обработчика была освобождена (ранее он занимал блок 4 и 5) и обработчик прерывания прекратил работу.

Рисунок 3 – Выгрузка обработчика

Контрольные вопросы.

1. Какого типа прерывания использовались в работе?

Было использовано аппаратное прерывание от клавиатуры 25h. Также использовались программные прерывания, например, int 21h.

2. Чем отличается скан-код и ASCII код?

Если ASCII код – это код символа для хранения символов и печати на экран, то скан-код – это код клавиши на клавиатуре, используемый 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. LAB5.ASM

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

INTERRUPTION    PROC    FAR
        jmp     INT_START
    INT_DATA:
        INT_CODE        DW  3158h

        KEEP_IP   DW  0
        KEEP_CS   DW  0
            KEEP_SS           DW  0
            KEEP_SP     DW  0
            KEEP_AX     DW  0
            KEEP_PSP    DW    0
            INT_STACK   DW    100 dup (?)

            SYMB        DB    0

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

            mov   AX, SEG SYMB
            mov   DS, AX

            in          AL, 60h
            cmp   AL, 10h
            je          OUT_P
            cmp   AL, 11h
            je          OUT_R
            cmp   AL, 12h
```

```
      je          OUT_I
      cmp   AL, 13h
      je          OUT_M
      cmp   AL, 14h
      je          OUT_I
      cmp   AL, 15h
      je          OUT_T
      cmp   AL, 16h
      je          OUT_E
      cmp   AL, 17h
      je          OUT_L
      cmp   AL, 18h
      je          OUT_A
      cmp   AL, 19h
      je          OUT_B
      cmp   AL, 1Ah
      je          OUT_U

      pushf
      call  DWORD PTR CS:KEEP_IP
      jmp   INT_END

OUT_P:
      mov         SYMB, 'P'
      jmp         PROCESSING_SYMB
OUT_R:
      mov         SYMB, 'R'
      jmp         PROCESSING_SYMB
OUT_I:
      mov         SYMB, 'I'
      jmp         PROCESSING_SYMB
OUT_M:
      mov         SYMB, 'M'
      jmp         PROCESSING_SYMB
OUT_T:
      mov         SYMB, 'T'
      jmp         PROCESSING_SYMB
OUT_E:
      mov         SYMB, 'E'
      jmp         PROCESSING_SYMB
OUT_L:
      mov         SYMB, 'L'
      jmp         PROCESSING_SYMB
OUT_A:
      mov         SYMB, 'A'
      jmp         PROCESSING_SYMB
OUT_B:
```

```
      mov         SYMB, 'B'
      jmp         PROCESSING_SYMB
OUT_U:
      mov         SYMB, 'U'

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


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

      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

WRITE    PROC    NEAR
        push    AX
        mov     AH, 09h
        int     21h
        pop     AX
    ret
WRITE    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    WRITE
            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    WRITE
      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  "Interruption was already loaded", 10, 13,"$"
      NOT_LOADED_INFO         DB  "Interruption is not loaded", 10, 13,"$"
    INT_LOADED          DB  0
    UN_CL               DB  0
DATA    ENDS

END   MAIN
```