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 _____ Кормщикова А. О.

Преподаватель _____ Ефремов М. А.

Санкт-Петербург

2020

Цель работы.

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

Ход выполнения.

Был написан код исходного .EXE модуля (LR4.ASM представлен в приложении А), который:

1) Проверяет, установлено ли пользовательское прерывание с вектором 09h

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

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

Код устанавливаемого прерывания сохраняет значения регистров в стеке при входе и восстанавливает их при выходе, обрабатывает нажатия клавиш k, o, j, m, a и вместо них в буфер клавиатуры кладет соответственно символы G, e, n, u, s. Все остальные клавиши обрабатываются стандартным обработчиком.

На рис.1 представлен вывод программы, которая отображает карту памяти до загрузки прерывания.

На рис.2 представлен вывод программы, которая отображает карту памяти после загрузки прерывания.

На рис.3 представлен результат работы прерывание при набранном тексте "Kojima".

На рис.4 представлена повторная загрузка прерывания, выгрузка и повторный набор "Kojima".

На рис. 5 представлен вывод программы, которая отображает карту памяти после выгрузки прерывания

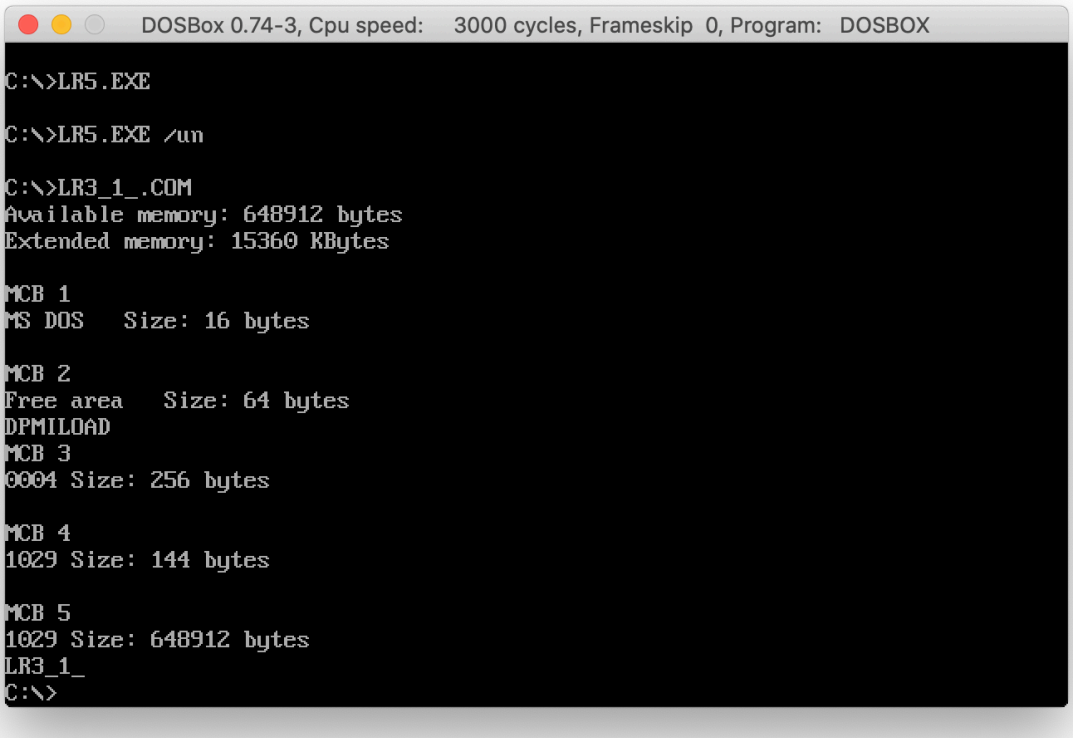

Рисунок 1 - Вывод карты памяти до загрузки прерывания

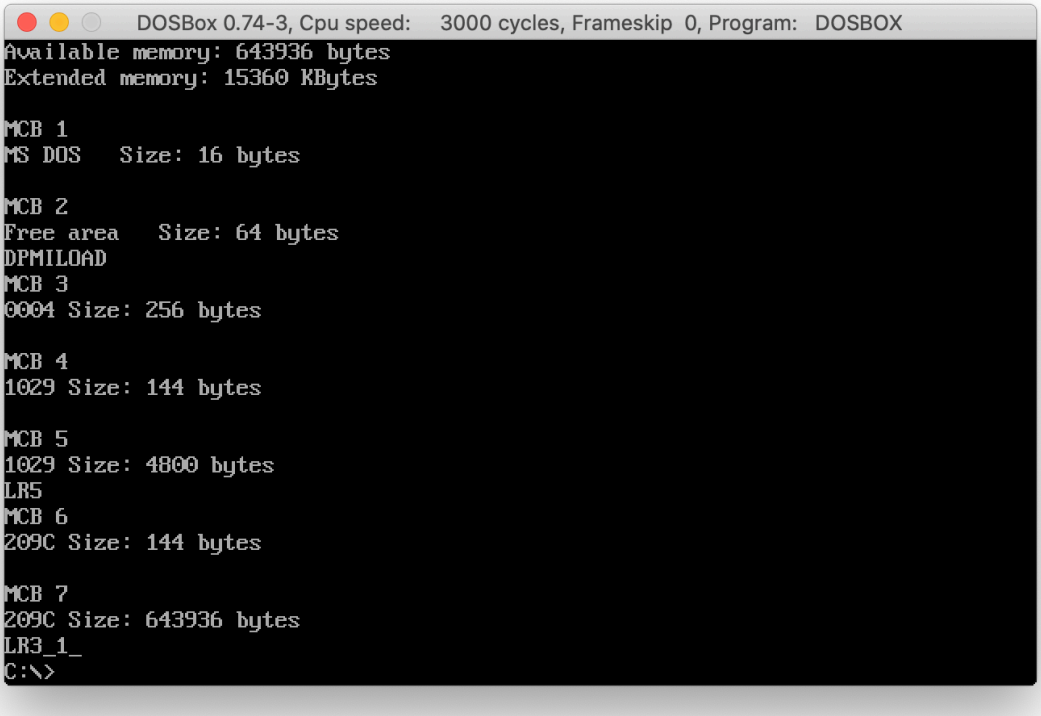

Рисунок 2 - Вывод карты памяти после загрузки прерывания

Рисунок 3 - Ввод строки "Kojima"

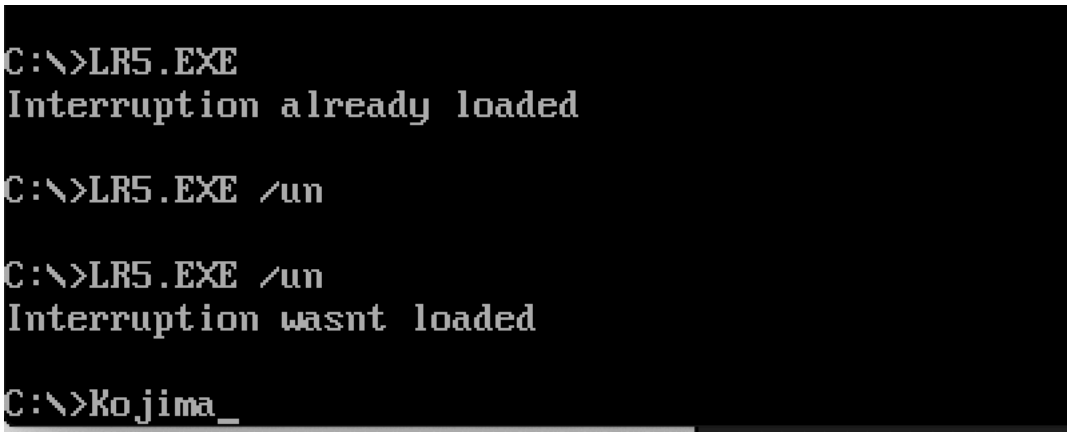

Рисунок 4 - Повторный запуск программы, с ключом выгрузки, повторный с ключом выгрузки, ввод строки "Kojima"

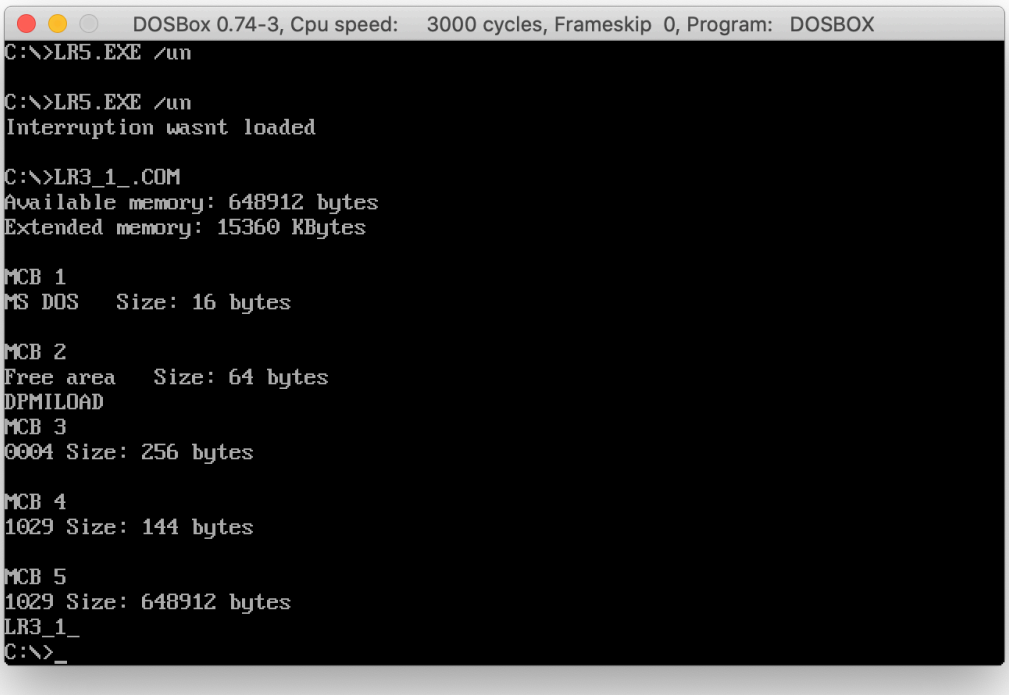

Рисунок 5 - Вывод карты памяти после выгрузки прерывания

Контрольные вопросы:

1) Какого типа прерывания использовались в работе?

Было использовано программное прерывание int 21h - сервис DOS. И аппаратное int 16h - сервис клавиатуры

2) Чем отличается скан код от кода ASCII?

Скан-код— в IBM-совместимых компьютерах код, присвоенный каждой клавише, с помощью которого драйвер клавиатуры распознает, какая клавиша была нажата. Скан-коды жёстко привязаны к каждой клавише на аппаратном уровне и не зависят ни от состояния индикаторов, ни от состояния управляющих клавиш, так же некоторые клавиши могут генерировать более одного скан кода.

ASCI код - код символа в таблице ASCII символов, необходимый для хранения символов в памяти и печати их на экран. Коды ASCII используются в программировании как промежуточные кроссплатформенные коды нажатых клавиш.

Выводы.

В ходе выполнения лабораторной работы были исследованы возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

ПРИЛОЖЕНИЕ А

```
SSTACK SEGMENT STACK
               DW 128
SSTACK  ENDS

DATA SEGMENT

      LOADED db 'Interruption already loaded', 0DH, 0AH, '$'
      NOTLOAD db 'Interruption wasnt loaded', 0DH, 0AH, '$'
      LOAD  db 0
      UN db 0


DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA, SS:SSTACK


ROUT PROC FAR ; обработчик прерываний

      jmp INTERRUPT
      SYMBOL db  0
      SIGN dw  1000h
      KEEP_IP dw  0
      KEEP_CS dw  0
      KEEP_PSP dw  0
      KEEP_SS dw  0
      KEEP_SP dw  0
      KEEP_AX dw  0
      INT_STACK dw 128 dup(0)

INTERRUPT:
    mov     KEEP_AX, AX
      mov   KEEP_SP, SP
      mov   KEEP_SS, SS
      mov   AX, SEG INT_STACK
      mov   SS, AX
      mov   AX, offset INT_STACK
      add   AX, 256
      mov   SP, AX

      push  AX
      push    BX
      push    CX
      push    DX
      push    SI
    push    ES
    push    DS
      mov   AX, seg SYMBOL
      mov   DS, AX

      in al, 60h

      cmp al, 25h
      je K     ;Kojima genius
      cmp al, 18h
      je O
      cmp al, 24h
      je J
      cmp al, 32h
      je M
      cmp al, 1eh
```

```
      je A

      pushf
      call DWORD PTR CS:KEEP_IP
      jmp END_

K:
      mov SYMBOL, 'G'
      jmp PROC_
O:
      mov SYMBOL, 'e'
      jmp PROC_
J:
      mov SYMBOL, 'n'
      jmp PROC_
M:
      mov SYMBOL, 'u'
      jmp PROC_
A:
      mov SYMBOL, 's'
      jmp PROC_


PROC_:
      in al, 61h
      mov ah, al
      or al, 80h
      out 61h, al
      xchg ah, al
      out 61h, al
      mov al, 20h
      out 20h, al

WRITE_:
      mov ah, 05h
      mov cl, SYMBOL
      mov ch, 00h
      int 16h
      or al, al
      je END_
      mov ax, 0040h
      mov es, ax
      mov ax, es:[1ah]
      mov es:[1ch], ax
      jmp WRITE_

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
ROUT ENDP
END_ROUT:
```

```
CHECK PROC
      push ax
      push bx
      push si

      mov ah, 35h
      mov al, 09h
      int 21h
      mov si, offset SIGN
      sub si, offset ROUT
      mov ax, es:[bx+si]
      cmp ax, SIGN
      jnz check_end
      mov LOAD, 1

check_end:
      pop si
      pop bx
      pop ax
      ret
CHECK ENDP

CHECK_UN PROC
      push ax
      push es

      mov ax, KEEP_PSP
      mov es, ax
      cmp byte ptr ES:[82H], '/'
      jnz CHECK_UN_END
      cmp byte ptr ES:[83H], 'u'
      jnz CHECK_UN_END
      cmp byte ptr ES:[84H], 'n'
      jnz CHECK_UN_END

      mov UN, 1

CHECK_UN_END:

      pop es
      pop ax
      ret
CHECK_UN ENDP



LOAD_I PROC
      push ax
      push bx
      push cx
      push dx
      push es
      push ds

      mov ah, 35h
      mov al, 09h
      int 21h

      mov KEEP_IP, bx
      mov KEEP_CS, es

      mov ax, seg ROUT
      mov dx, offset ROUT
      mov ds, ax
      mov ah, 25h
```

```
      mov al, 09h
      int 21h

      pop ds

      mov dx, offset END_ROUT
      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
LOAD_I ENDP


LOAD_UN PROC
      CLI
      push ax
      push bx
      push cx
      push dx
      push ds
      push es
      push si

      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]
      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 cx
```

```
      pop bx
      pop ax
      ret
LOAD_UN ENDP

MAIN PROC FAR
      push ds
      xor ax,ax
      push ax
      mov ax, DATA
      mov ds, ax
      mov KEEP_PSP, es
      call CHECK
      call CHECK_UN
      cmp UN, 1
      je UNLOAD
      cmp LOAD, 1
      jnz LOAD_
      mov dx, offset LOADED

      push ax
      mov ah, 09h
      int 21h
      pop ax

      jmp MEND
LOAD_:
      call LOAD_I
      jmp MEND

UNLOAD:
      cmp LOAD, 1
      jnz NOT_LOAD
      call LOAD_UN
      jmp MEND

NOT_LOAD:
      mov dx, offset NOTLOAD
      push ax
      mov ah, 09h
      int 21h
      pop ax

MEND:
      xor al, al
      mov ah, 4ch
      int 21h


MAIN      ENDP
CODE ENDS
END MAIN
```