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. 9381 Шахин Н.С

Преподаватель Ефремов М. А.

Санкт-Петербург

2021

Цель работы.

Исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Пользовательский обработчик прерываний получает управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре. Он обрабатывает скан-код и осуществляет определенные действия, если скан-код совпадает с определёнными кодами, которые он должен обрабатывать. Если скан-код не совпадает с этими кодами, то управление передается стандартному прерыванию.

Ход работы.

1) Написан и отлажен программный модуль типа .EXE, который выполняет такие же функции, как и в программе лабораторной работы №4, а именно:

- Проверяет, установлено ли пользовательское прерывание с вектором 09h.
- 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 int21h.
- Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h.

2) Запустил отлаженную программу и убедился, что резидентный обработчик прерывания 09h установлен. Работа прерывания была проверена введением ! .

```
F:\>lab5
Interruption is loading now!

F:\>!!!!!!!
```

3) Проверил размещение прерывания в памяти. Для этого запустил программу лабораторной работы №3, которая отображает карту памяти в виде списка блоков MCB.

```
F:\>lab3
Amount of available memory:     648160 b
Size of extended memory:      15360 Kb
List of memory control blocks:
MCB type: 4Dh   PSP adress: 0008h       Size:       16 b
MCB type: 4Dh   PSP adress: 0000h       Size:       64 b
MCB type: 4Dh   PSP adress: 0040h       Size:      256 b
MCB type: 4Dh   PSP adress: 0192h       Size:      144 b
MCB type: 4Dh   PSP adress: 0192h       Size:      576 b        LAB5
MCB type: 4Dh   PSP adress: 01C1h       Size:      144 b
MCB type: 5Ah   PSP adress: 01C1h       Size:   648160 b        LAB3
```

Как видно, резидент находится в памяти и успешно используется.

4) Запустил отлаженную программу с ключом выгрузки и убедился, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также запустил программу лабораторной работы №3.

```
F:\>lab5 /un
Interruption was restored!

F:\>lab3
Amount of available memory:     648912 b
Size of extended memory:      15360 Kb
List of memory control blocks:
MCB type: 4Dh   PSP adress: 0008h       Size:       16 b
MCB type: 4Dh   PSP adress: 0000h       Size:       64 b
MCB type: 4Dh   PSP adress: 0040h       Size:      256 b
MCB type: 4Dh   PSP adress: 0192h       Size:      144 b
MCB type: 5Ah   PSP adress: 0192h       Size:   648912 b        LAB3
```

Резидент был успешно выгружен из памяти.

Вывод.

В ходе данной работы производилось исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Было написано пользовательское прерывание от клавиатуры, которое анализирует скан-коды, выполняет вывод сообщения

результата нажатия, а при несовпадении скан-кода передает управление стандартному обработчику.

Ответы на контрольные вопросы:

1. Какого типа прерывания использовались в работе?

Использовались следующие типы прерываний:

Аппаратные (прерывание от клавиатуры 09h)

Программные (прерывания вызываемые командой int 21h)

2. Чем отличается скан код от кода ASCII?

Скан-код – это код, который клавиатура передаёт системе. Тем самым система определяет, какая клавиша (или комбинация клавиш) была нажата. ASCII-код – это таблица кодировок для печатных символов.

Таким образом скан-код – это номер клавиши, а ASCII-код – код, соответствующий обозначению на этой клавише

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
LAB5 SEGMENT
        ASSUME CS:LAB5, DS:DATA, SS:STACK

ROUT PROC FAR
        jmp START_ROUT

        PSP_ADDRESS_0 dw ?
        PSP_ADDRESS_1 dw ?
        KEEP_IP dw 0
        KEEP_CS dw 0
        ROUT_SET dw 8f17h

        VEC dd 0

        REQ_KEY db 02h

        KEEP_SS dw ?
      KEEP_SP dw ?
      KEEP_AX dw ?
      INT_STACK dw 64 dup (?)
      END_INT_STACK dw ?

START_ROUT:
        mov KEEP_SS, ss
        mov KEEP_SP, sp
        mov KEEP_AX, ax
        mov ax, cs
        mov ss, ax
        mov sp, offset END_INT_STACK

        push ax
        push bx
        push cx
        push dx
        push es

        in al, 60h
        cmp al, REQ_KEY
        jne NO_MATCH

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

        mov ah,05h

        push ax
      push es
      xor ax, ax
      mov es, ax
      mov al, es:[417h]
      pop es
      and al, 00000011b
      jnz SHIFT
```

```
      pop ax
      mov cl, '1'
        jmp WRITING_CHAR

SHIFT:
        pop ax
        mov cl, '!'

WRITING_CHAR:
        mov ch, 00h
        int 16h
        or al, al
        jnz SKIP
        jmp END_ROUT

SKIP:
        mov ax, 0C00h
        int 21h
        jmp WRITING_CHAR

NO_MATCH:
        mov     AX, KEEP_IP
        mov     word ptr VEC, AX
        mov     AX, KEEP_CS
        mov     word ptr VEC + 2, AX
        pop es
        pop dx
        pop cx
        pop bx
        pop ax
        mov     AX, KEEP_SS
        mov     SS, AX
        mov     AX, KEEP_AX
        mov     SP, KEEP_SP
        jmp     CS:[VEC]

END_ROUT:
        pop es
      pop dx
      pop cx
      pop bx
        pop ax

      mov ss, KEEP_SS
      mov ax, KEEP_AX
      mov sp, KEEP_SP

        mov al, 20H
        out 20h, al
        iret
NEED_MEM_AREA:
ROUT ENDP

IS_UNLOADING_STARTED PROC NEAR

        mov PSP_ADDRESS_1, es
        push es
        push cx

        xor cx, cx
        mov cl, es:[80h]
        cmp cl, 0
        je NULL_CMD
```

```
      mov ax, PSP_ADDRESS_0
      mov es, ax

      mov bx, 0082h

      mov al, es:[bx]
      cmp al, '/'
      jne NULL_CMD

      mov al, es:[bx+1]
      cmp al, 'u'
      jne NULL_CMD

      mov al, es:[bx+2]
      cmp al, 'n'
      jne NULL_CMD
      mov al, 1
        jmp NULL_CMD

NULL_CMD:
        pop cx
        pop es
        ret
IS_UNLOADING_STARTED ENDP

IS_LOADING PROC NEAR
        push dx
        push bx
        push si

        mov ah,35h
        mov al,1Ch
        int 21h

        mov si, offset ROUT_SET
        sub si, offset ROUT
        mov dx, es:[bx+si]
        cmp dx, 8f17h
        je INT_IS_SET
        mov al, 0
        jmp POP_REG

INT_IS_SET:
      mov al, 1

POP_REG:
        pop si
        pop bx
        pop dx
        ret
IS_LOADING ENDP

ADD_ROUT PROC NEAR

        push ds
        mov dx, offset ROUT
        mov ax, seg ROUT
        mov ds, ax
        mov ah, 25H
        mov al, 1CH
        int 21H
        pop ds
```

```
        ret

ADD_ROUT ENDP

LOADING_STARTED PROC NEAR
        push ax
        push dx
        push bx
        push es

        mov AH,35h
      mov AL,1Ch
      int 21h

      mov KEEP_IP, bx
        mov KEEP_CS, es

        call ADD_ROUT

        mov dx, offset [INT_ISLOADING]
        call WRITE

        call RESIDENT

        pop es
        pop bx
        pop dx
        pop ax

        ret
LOADING_STARTED ENDP

UNLOADING_STARTED PROC NEAR
        push ax
        push bx
        push dx
        push si
        push es

        mov ah, 35h
        mov al, 1Ch
        int 21h

        cli
        push ds
        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, 1CH
        int 21H
        pop ds
        sti

        mov dx, offset [INT_RESTORED]
        call WRITE

        mov cx, es:[bx+si-2]
        mov es, cx
        push es
        mov cx, es:[2ch]
```

```
        mov es, cx
        mov ah, 49h
        int 21h
        mov ah, 49h
        pop es
        int 21h

        pop es
        pop si
        pop dx
        pop bx
        pop ax
        ret
UNLOADING_STARTED ENDP

RESIDENT PROC NEAR

        push dx
        push cx
        push ax
        lea dx, NEED_MEM_AREA
        mov CL, 4h
        shr DX, CL
        inc DX
        mov ax, cs
        sub ax, PSP_ADDRESS_1
        add dx, ax
        xor ax, ax
        mov AH, 31h
        int 21h
        pop ax
        pop cx
        pop dx
        ret

RESIDENT ENDP

WRITE PROC NEAR
        mov AH,09h
        int 21h
        ret
WRITE ENDP

MAIN PROC  FAR

        mov bx, 02Ch
      mov ax, [bx]
      mov PSP_ADDRESS_0, ds
      sub ax, ax
      xor bx, bx

        mov ax, DATA
        mov ds, ax

        call IS_UNLOADING_STARTED
        cmp al, 1h
        je UNLOAD_START

        call IS_LOADING
        cmp al, 1
        je INTERRUPTI0N_IS_LOADED
        call LOADING_STARTED
```

```
UNLOAD_START:
        call IS_LOADING
        cmp al, 0
        je INT_IS_NOT_SET
        call UNLOADING_STARTED
        jmp EXIT

INT_IS_NOT_SET:
      mov dx, offset [INT_NOT_SET]
      call WRITE
        jmp EXIT

INTERRUPTI0N_IS_LOADED:
        mov dx, offset [INT_ISLOADED]
      call WRITE

EXIT:
        mov ah,4ch
        int 21h
MAIN ENDP
LAB5 ENDS
STACK SEGMENT  STACK
        DW 64 DUP(?)
STACK ENDS
DATA SEGMENT
        INT_NOT_SET DB "Interruption didn't load!",0DH,0AH,'$'
        INT_RESTORED DB "Interruption was restored!",0DH,0AH,'$'
        INT_ISLOADED DB "Interruption has already loaded!",0DH,0AH,'$'
        INT_ISLOADING DB "Interruption is loading now!",0DH,0AH,'$'
DATA ENDS
        END MAIN
```