

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. 9383

Рыбников Р.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

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

Постановка задачи.

Требуется написать и отладить .EXE модуль, который выполняет следующие функции:

- 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 и осуществляется выход по функции 4Ch прерывания int 21h.

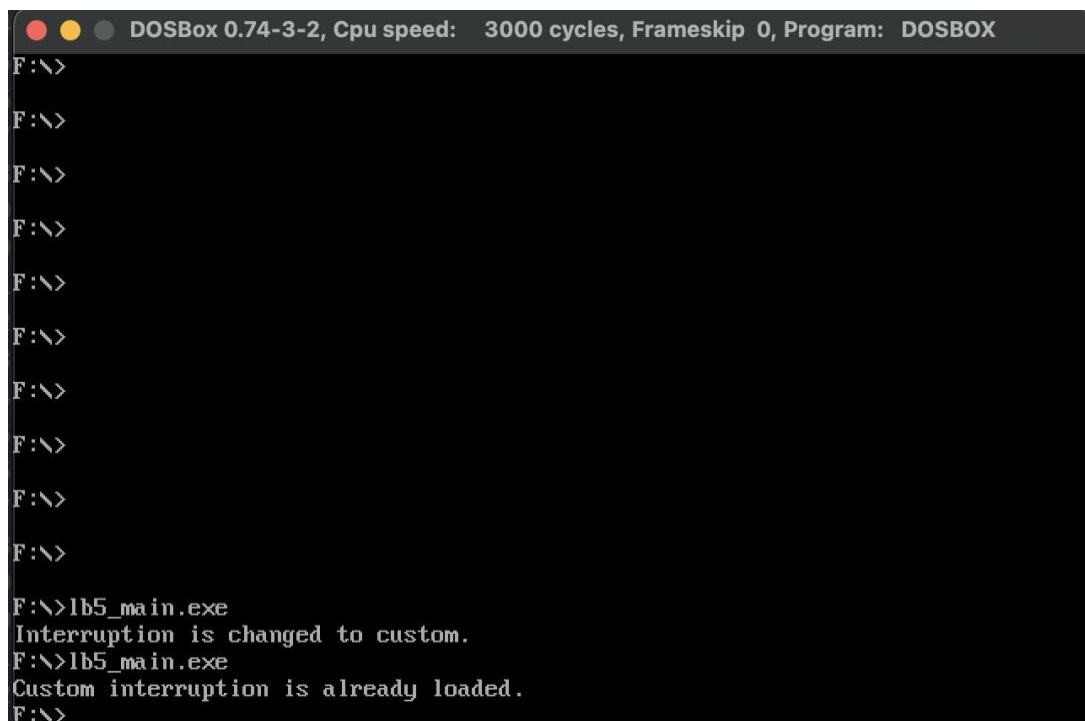
Программа должна содержать код устанавливаемого прерывания в виде удаленной процедуры. Код должен:

- 1) Сохранить значения регистров в стеке при входе и восстановить их при выходе.
- 2) При выполнении тела процедуры анализируется скан-код.
- 3) Если этот код совпадает с одним из заданных, то требуемый код записывается в буфер клавиатуры.

- 4) Если этот код не совпадает ни с одним из заданных, то осуществляется передача управления стандартному обработчику прерывания.

Выполнение работы.

Сначала был написан код для .EXE модуля. Модуль работает следующим образом: если он запущен без ключа /up, то он установит прерывание в память. Далее, при попытках снова установить это прерывание будет выводиться соответствующая строка. Это можно видеть на рисунке 1.



```
DOSBox 0.74-3-2, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
F:\>
F:\>
F:\>
F:\>
F:\>
F:\>
F:\>
F:\>
F:\>
F:\>
F:\>
F:\>lb5_main.exe
Interruption is changed to custom.
F:\>lb5_main.exe
Custom interruption is already loaded.
F:\>
```

Рисунок 1 – Установка прерывания.

Также, прерывание можно выгрузить, запустив программу с ключом /up. Если запустить программу так же еще раз, после выгрузки, то выведется соответствующее сообщение. Это представлено на рисунке 2.

```
DOSBox 0.74-3-2, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
F:\>
F:\>
F:\>
F:\>
F:\>
F:\>
F:\>
F:\>
F:\>lb5_main.exe
Interruption is changed to custom.
F:\>lb5_main.exe
Custom interruption is already loaded.
F:\>
F:\>/un
F:\>lb5_main.exe /un
Custom interruption was unloaded.
F:\>
```

Рисунок 2 – Выгрузка прерывания.

Далее на рисунках 3-6 представлены соответственно: работа прерывания, расположение в памяти после загрузки прерывания, расположение в памяти после повторной загрузки, расположение в памяти после выгрузки.

Чтобы увидеть работу прерывания, необходимо зажать сочетание клавиш ctrl+a.

```
DOSBox 0.74-3-2, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
F:\>
F:\>
F:\>
F:\>lb5_main.exe
Interruption is changed to custom.
F:\>lb5_main.exe
Custom interruption is already loaded.
F:\>
F:\>/un
F:\>lb5_main.exe /un
Custom interruption was unloaded.
F:\>
F:\>
F:\>lb5_main.exe
Interruption is changed to custom.
F:\>lb5_main.exe
Custom interruption is already loaded.
F:\>aaaaaaaa
```

Рисунок 3 — Работа прерывания (нажатие ctrl+a).

```
DOSBox 0.74-3-2, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
F:\>lb5_main.exe /un
Custom interruption was unloaded.
F:\>
F:\>
F:\>lb5_main.exe
Interruption is changed to custom.
F:\>lb5_main.exe
Custom interruption is already loaded.
F:\>
F:\>lab3_1.com
Size of accessed memory: 644304 byte
Size of extended memory: 245760 byte
MCB:01 Address: 016F PSP address: 0008 Size: 16 SD/SC:
MCB:02 Address: 0171 PSP address: 0000 Size: 64 SD/SC:
MCB:03 Address: 0176 PSP address: 0040 Size: 256 SD/SC:
MCB:04 Address: 0187 PSP address: 0192 Size: 144 SD/SC:
MCB:05 Address: 0191 PSP address: 0192 Size: 4432 SD/SC: LB5_MAIN
MCB:06 Address: 02A7 PSP address: 02B2 Size: 1442 SD/SC:
MCB:07 Address: 02B1 PSP address: 02B2 Size: 644304 SD/SC: LAB3_1
F:\>_
```

Рисунок 4 – Расположение в памяти после загрузки прерывания.

```
DOSBox 0.74-3-2, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
Size of accessed memory: 648912 byte
Size of extended memory: 245760 byte
MCB:01 Address: 016F PSP address: 0008 Size: 16 SD/SC:
MCB:02 Address: 0171 PSP address: 0000 Size: 64 SD/SC:
MCB:03 Address: 0176 PSP address: 0040 Size: 256 SD/SC:
MCB:04 Address: 0187 PSP address: 0192 Size: 144 SD/SC:
MCB:05 Address: 0191 PSP address: 0192 Size: 648912 SD/SC: LAB3_1
F:\>

F:\>lb5_main.exe
Interruption is changed to custom.
F:\>lb5_main.exe
Custom interruption is already loaded.
F:\>lab3_1.com

Size of accessed memory: 644304 byte
Size of extended memory: 245760 byte
MCB:01 Address: 016F PSP address: 0008 Size: 16 SD/SC:
MCB:02 Address: 0171 PSP address: 0000 Size: 64 SD/SC:
MCB:03 Address: 0176 PSP address: 0040 Size: 256 SD/SC:
MCB:04 Address: 0187 PSP address: 0192 Size: 144 SD/SC:
MCB:05 Address: 0191 PSP address: 0192 Size: 4432 SD/SC: LB5_MAIN
MCB:06 Address: 02A7 PSP address: 02B2 Size: 1442 SD/SC:
MCB:07 Address: 02B1 PSP address: 02B2 Size: 644304 SD/SC: LAB3_1
F:\>
```

Рисунок 5 – Расположение прерывания в памяти после повторной загрузки.

```
DOSBox 0.74-3-2, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
F:\>lab3_1.com

Size of accessed memory: 644304 byte
Size of extended memory: 245760 byte
MCB:01 Address: 016F PSP address: 0008 Size: 16 SD/SC:
MCB:02 Address: 0171 PSP address: 0000 Size: 64 SD/SC:
MCB:03 Address: 0176 PSP address: 0040 Size: 256 SD/SC:
MCB:04 Address: 0187 PSP address: 0192 Size: 144 SD/SC:
MCB:05 Address: 0191 PSP address: 0192 Size: 4432 SD/SC: LB5_MAIN
MCB:06 Address: 02A7 PSP address: 02B2 Size: 1442 SD/SC:
MCB:07 Address: 02B1 PSP address: 02B2 Size: 644304 SD/SC: LAB3_1
F:\>

F:\>lb5_main.exe /un
Custom interruption was unloaded.
F:\>lab3_1.com

Size of accessed memory: 648912 byte
Size of extended memory: 245760 byte
MCB:01 Address: 016F PSP address: 0008 Size: 16 SD/SC:
MCB:02 Address: 0171 PSP address: 0000 Size: 64 SD/SC:
MCB:03 Address: 0176 PSP address: 0040 Size: 256 SD/SC:
MCB:04 Address: 0187 PSP address: 0192 Size: 144 SD/SC:
MCB:05 Address: 0191 PSP address: 0192 Size: 648912 SD/SC: LAB3_1
F:\>
```

Рисунок 6 - Расположение в памяти после выгрузки прерывания.

Ответы на вопросы.

1. Какого типа прерывания использовались в работе?

Программные прерывания int 21h и int 21h, а так же аппаратные прерывания 09h и 16h.

2. Чем отличается скан код от кода ASCII?

Скан-код – уникальный числовой код, присвоенный клавише клавиатуры, который определяет нажатую клавишу. Код ASCII – уникальный код каждого символа таблицы ASCII.

Выводы.

Был реализован пользовательский обработчик прерываний, который был встроен в стандартный обработчик клавиатуры.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

lb5_main.asm:

```
AStack  SEGMENT STACK
        DW 64 DUP(?)
AStack  ENDS
```

```
DATA SEGMENT
    SECOND_INFO db "Custom interruption is already loaded.$"
    THIRD_INFO  db "Interruption is changed to custom.$"
    FIRST_INFO  db "Default interruption is set and can't be unloaded.$"
    FOURTH_INFO db "Custom interruption was unloaded.$"
DATA ENDS
```

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

```
PRINT_BUFFER proc near
    push ax
    mov ah, 9h
    int 21h
    pop ax
    ret
PRINT_BUFFER endp
```

```
START:
ROUT proc far
    jmp start1
    key_value db 0
    KEEP_PSP dw 0
    KEEP_IP dw 0
    KEEP_CS dw 0
    KEEP_SS DW 0
    KEEP_SP DW 0
    KEEP_AX DW 0
    ROUT_INDEX dw 6666h
    TIMER_COUNTER db 'Timer: 0000$'
    BStack DW 64 DUP(?)
start1:
    mov KEEP_SP, sp
    mov KEEP_AX, ax
    mov KEEP_SS, ss

    mov ax, seg BStack
    mov ss, ax
    mov ax, offset start1
    mov sp, ax
```



```
mov ax, KEEP_AX
```

```
push ax  
push bx  
push cx  
push dx  
push si  
push es  
push ds
```

```
    mov cx, 040h  
    mov es, cx  
    mov cx, es:[0017h]  
mov ax, SEG key_value  
mov ds, ax
```

```
and cx, 0100b  
jz defolt_int
```

```
in al, 60h  
cmp al, 1Eh  
je key
```

```
defolt_int:  
    call dword ptr cs:[KEEP_IP]  
    jmp re_reg
```

```
key:  
    mov key_value, '&'  
    jmp next
```

```
next:  
    in al, 61h  
    mov ah, al  
    or al, 80h  
    out 61h, al  
    xchg ah, al  
    out 61h, al  
    mov al, 20H  
    out 20h, al
```

```
print_key:  
  
    mov ah, 05h  
    mov cl, key_value
```

```

mov ch, 00h
int 16h
or al,al
jz re_reg
mov ax, 040h
mov es, ax
mov ax, es:[1Ah]
mov es:[1Ch], ax
jmp print_key

```

```

skip:
    mov al, es:[1Ah]
    mov es:[1Ch], al
    jmp print_key

```

```

re_reg:
    pop ds
    pop es
    pop si
    pop dx
    pop cx
    pop bx
    pop ax

```

```

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

```

```

mov al, 20H
out 20H, al

```

```

    iret
end_rout:
ROUT endp

```

```

UNLOAD proc near
    push ax
    push es

    mov al,es:[81h+1]
    cmp al,'/'
    jne need_unload

    mov al,es:[81h+2]
    cmp al,'u'
    jne need_unload

    mov al,es:[81h+3]

```

```

        cmp al,'n'
        jne need_unload

        mov cl,1h

need_unload:
        pop es
        pop ax
        ret
UNLOAD endp


LOAD PROC near
        push ax
        push dx

        mov KEEP_PSP, es

        mov ah,35h
        mov al,09h
        int 21h
        mov KEEP_IP, bx
        mov KEEP_CS, es

        push ds
        lea dx, ROUT
        mov ax, SEG ROUT
        mov ds,ax
        mov ah,25h
        mov al,09h
        int 21h
        pop ds

        lea dx, end_rout
        mov cl,4h
        shr dx,cl
        inc dx
        add dx,100h
        xor ax, ax
        mov ah,31h
        int 21h

        pop dx
        pop ax
        ret
LOAD ENDP


UNLOAD_ROUT PROC near
        push ax
        push si

        cli

```

```

        push ds
        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]
mov ds,ax
mov ah,25h
mov al,09h
int 21h
pop ds

mov ax,es:[bx+si-2]
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 ax
ret
UNLOAD_ROUT endp

CHECK_LOAD proc near
    push ax
    push si

    push es
    push dx

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

    mov si, offset ROUT_INDEX
    sub si, offset ROUT
    mov dx,es:[bx+si]
    cmp dx, ROUT_INDEX
    jne end_check
    mov ch,1h

```

```

end_check:
    pop dx
    pop es
    pop si
    pop ax
    ret
CHECK_LOAD ENDP

MAIN proc far
    push DS
    push AX
    mov AX,DATA
    mov DS,AX

    call UNLOAD
    cmp cl, 1h
    je start_unload

    call CHECK_LOAD
    cmp ch, 1h
    je marker_1
    mov dx, offset THIRD_INFO
    call PRINT_BUFFER
    call LOAD
    jmp exit

start_unload:
    call CHECK_LOAD
    cmp ch, 1h
    jne marker_2
    call UNLOAD_ROUT
    mov dx, offset FOURTH_INFO
    call PRINT_BUFFER
    jmp exit

marker_2:
    mov dx, offset FIRST_INFO
    call PRINT_BUFFER
    jmp exit
marker_1:
    mov dx, offset SECOND_INFO
    call PRINT_BUFFER
    jmp exit

exit:
    mov ah, 4ch
    int 21h
MAIN endp
CODE ends
END Main

```

lab3_1.asm:

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START: JMP BEGIN
ACCESSED_MEMORY db 13,10,'Size of accessed memory:    $'
EXTENDED_MEMORY db 13,10,'Size of extended memory:    $'
STR_BYTE db ' byte $'
STR_MCB db 13,10,'MCB:0 $'
ADRESS db 'Adress:    $'
ADRESS_PSP db 'PSP address:    $'
STR_SIZE db 'Size:    $'
MCB_SD_SC db ' SD/SC: $'
STR_ERROR db 13,10,'Memory Error!$'
STR_SUCCECC db 13,10,'Success!$'

TETR_TO_HEX PROC near
    and AL,0Fh
    cmp AL,09
    jbe next
    add AL,07
next:
    add AL,30h
    ret
TETR_TO_HEX ENDP

BYTE_TO_HEX PROC near
    push CX
    mov AH,AL
    call TETR_TO_HEX
    xchg AL,AH
    mov CL,4
    shr AL,CL
    call TETR_TO_HEX
    pop CX ;в AH младшая
    ret
BYTE_TO_HEX ENDP
;.....
WRD_TO_HEX PROC near
;перевод в 16 с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
    push BX
    mov BH,AH
    call BYTE_TO_HEX
    mov [DI],AH
    dec DI
    mov [DI],AL
    dec DI
    mov AL,BH
    call BYTE_TO_HEX
    mov [DI],AH
```

```

    dec DI
    mov [DI],AL
    pop BX
    ret
WRD_TO_HEX ENDP
;.....
BYTE_TO_DEC PROC near
; перевод в 10с/с, SI - адрес поля младшей цифры
    push CX
    push DX
    xor AH,AH
    xor DX,DX
    mov CX,10
loop_bd:
    div CX
    or DL,30h
    mov [SI],DL
    dec SI
    xor DX,DX
    cmp AX,10
    jae loop_bd
    cmp AL,00h
    je end_l
    or AL,30h
    mov [SI],AL
end_l:
    pop DX
    pop CX
    ret
BYTE_TO_DEC ENDP
;.....
WRITE_STRING PROC near
    push AX
    mov AH,09h
    int 21h
    pop AX
    ret
WRITE_STRING ENDP
;.....

WRITE_SIZE PROC
    push ax
    push bx
    push cx
    push dx
    push si

        mov bx,10h
        mul bx
        mov bx,0ah
        xor cx,cx
delenie:

```

```

        div bx
        push dx
        inc cx
        xor dx,dx
        cmp ax,0h
        jnz delenie

write_symbol:
        pop dx
        or dl,30h
        mov [si], dl
        inc si
        loop write_symbol

        pop si
        pop dx
        pop cx
        pop bx
        pop ax
        ret
WRITE_SIZE ENDP
;.....
PRINT_MCB PROC
        push ax
        push bx
        push cx
        push dx
        push si

        mov ah,52h
        int 21h
        mov ax,es:[bx-2]
        mov es,ax
        xor cx,cx

        inc cx
pargraph_MCB:
        lea si, STR_MCB
        add si, 7
        mov al,cl
        push cx
        call BYTE_TO_DEC
        lea dx, STR_MCB
        call WRITE_STRING

        mov ax,es
        lea di,ADRESS
        add di,12
        call WRD_TO_HEX
        lea dx,ADRESS
        call WRITE_STRING

```



```

xor ah,ah
mov al,es:[0]
push ax
mov ax,es:[1]
lea di, ADRESS_PSP
add di, 15
call WRD_TO_HEX
lea dx, ADRESS_PSP
call WRITE_STRING
mov ax,es:[3]
lea si,STR_SIZE
add si, 6
call WRITE_SIZE
lea dx,STR_SIZE
call WRITE_STRING
xor dx, dx
lea dx , MCB_SD_SC
call WRITE_STRING
mov cx,8
xor di,di

```

write_char:

```

mov dl,es:[di+8]
mov ah,02h
int 21h
inc di
loop write_char

```

```

mov ax,es:[3]
mov bx,es
add bx,ax
inc bx
mov es,bx
pop ax
pop cx
inc cx
cmp al,5Ah
je exit
cmp al,4Dh
jne exit
jmp pargraph_MCB

```

exit:

```

pop si
pop dx
pop cx
pop bx
pop ax

```

ret

PRINT_MCB ENDP

```

BEGIN:
    mov ah,4ah
        mov bx,0ffffh
        int 21h
    mov ax,bx
    lea si, ACCESSED_MEMORY
    add si, 27
    call WRITE_SIZE
    lea dx, ACCESSED_MEMORY
    call WRITE_STRING
    lea dx,STR_BYTE
    call WRITE_STRING

; расширенная память
    mov AL,30h
    out 70h,AL
    in AL,71h
    mov BL,AL
    mov AL,31h
    out 70h,AL
    in AL,71h

        mov bh,al
        mov ax,bx
        lea si,EXTENDED_MEMORY
        add si, 27
        call WRITE_SIZE
        lea dx,EXTENDED_MEMORY
        call WRITE_STRING
        lea dx,STR_BYTE
        call WRITE_STRING

;MCB
call PRINT_MCB

xor AL,AL
mov AH,4Ch
int 21H
TESTPC ENDS
END START

```