

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЁТ
по лабораторной работе №5
по дисциплине «Операционные системы»
Тема: Сопряжение стандартного и пользовательского обработчиков
прерываний

Студент гр.8382

_____ Фильцин И.В.

Преподаватель

_____ Ефремов М.А..

Санкт-Петербург

2020

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.

Ход работы

В ходе лабораторной программы был написан программный модуль типа .EXE, устанавливающий собственный обработчик прерывания клавиатуры.

После запуска программа запоминает старый обработчик прерывания, устанавливает новый и завершается не освобождая занятую память.

Размещение программы в памяти см. на рис. 1

После повторного запуска программа корректно определяет, что обработчик прерывания был изменен, и не делает этого снова. (Результат запуска см. на рис. 2)

При запуске программы с ключом */un* происходит выгрузка обработчика прерывания и восстановления старого. При этом занятая память освобождается.

Состояние памяти см. на рис. 3

Контрольные вопросы

1) Какого типа прерывания использовались в работе?

Аппаратные – 13h

Программные – 21h, 10h

```
HAVE FUN!  
The DOSBox Team http://www.dosbox.com
```

```
Z:\>SET BLASTER=A220 I7 D1 H5 T6  
  
Z:\>MOUNT C "."  
Drive C is mounted as local directory ./  
  
Z:\>C:  
  
C:\>1EXE.EXE  
  
C:\>1COM.COM  
Memory(B): 641616  
Expanded memory(KB): 015360  
MCB type = 4D | Owner = 00080 | Size(B) = 000016 | Last bytes =  
MCB type = 4D | Owner = 00000 | Size(B) = 000064 | Last bytes =  
MCB type = 4D | Owner = 00400 | Size(B) = 000256 | Last bytes =  
MCB type = 4D | Owner = 01920 | Size(B) = 000144 | Last bytes =  
MCB type = 4D | Owner = 01920 | Size(B) = 007120 | Last bytes = 1EXE  
MCB type = 4D | Owner = 035A0 | Size(B) = 007144 | Last bytes =  
MCB type = 5A | Owner = 035A0 | Size(B) = 641616 | Last bytes = 1COM  
  
C:\>_
```

Рис. 1: Запуск программы

```
Z:\>SET BLASTER=A220 I7 D1 H5 T6  
  
Z:\>MOUNT C "."  
Drive C is mounted as local directory ./  
  
Z:\>C:  
  
C:\>1EXE.EXE  
  
C:\>1COM.COM  
Memory(B): 641616  
Expanded memory(KB): 015360  
MCB type = 4D | Owner = 00080 | Size(B) = 000016 | Last bytes =  
MCB type = 4D | Owner = 00000 | Size(B) = 000064 | Last bytes =  
MCB type = 4D | Owner = 00400 | Size(B) = 000256 | Last bytes =  
MCB type = 4D | Owner = 01920 | Size(B) = 000144 | Last bytes =  
MCB type = 4D | Owner = 01920 | Size(B) = 007120 | Last bytes = 1EXE  
MCB type = 4D | Owner = 035A0 | Size(B) = 007144 | Last bytes =  
MCB type = 5A | Owner = 035A0 | Size(B) = 641616 | Last bytes = 1COM  
  
C:\>1EXE.EXE  
Interrupt descriptor already set  
  
C:\>
```

Рис. 2: Повторный запуск программы

```

Memory(B): 641616
Expanded memory(KB): 015360
MCB type = 4D | Owner = 00080 | Size(B) = 000016 | Last bytes =
MCB type = 4D | Owner = 00000 | Size(B) = 000064 | Last bytes =
MCB type = 4D | Owner = 00400 | Size(B) = 000256 | Last bytes =
MCB type = 4D | Owner = 01920 | Size(B) = 000144 | Last bytes =
MCB type = 4D | Owner = 01920 | Size(B) = 007120 | Last bytes = 1EXE
MCB type = 4D | Owner = 035A0 | Size(B) = 007144 | Last bytes =
MCB type = 5A | Owner = 035A0 | Size(B) = 641616 | Last bytes = 1COM

C:\>1EXE.EXE
Interrupt descriptor already set

C:\>1EXE.EXE /un

C:\>1COM.COM
Memory(B): 648912
Expanded memory(KB): 015360
MCB type = 4D | Owner = 00080 | Size(B) = 000016 | Last bytes =
MCB type = 4D | Owner = 00000 | Size(B) = 000064 | Last bytes =
MCB type = 4D | Owner = 00400 | Size(B) = 000256 | Last bytes =
MCB type = 4D | Owner = 01920 | Size(B) = 000144 | Last bytes =
MCB type = 5A | Owner = 01920 | Size(B) = 648912 | Last bytes = 1COM

C:\>

```

Рис. 3: Выгрузка прерывания

2) Чем отличается скан код от кода ASCII.

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

ASCII-код – код символа в таблице ASCII

Вывод

В ходе лабораторной работы был реализован обработчик прерываний клавиатуры.

Приложение А. Исходный код программы

```
.model small
```

```
.code
```

```
db 100h dup(0)
```

```
istack label word
```

```
keep_ss dw 0
```

```
keep_sp dw 0
```

```
ssss db 'WOW', 13, 10, '$'
```

```
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
    ret
byte_to_hex endp

```

```

wrd_to_hex proc near
    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

```

```

resident_begin proc far
    push ax ; уж возьмём 2 байта из пользовательского

```

сTeKa

mov keep_ss, **ss**

mov keep_sp, **sp**

mov **ax**, @code

cli

mov **ss**, **ax**

mov **sp**, **offset** istack

sti

push **ds**

push **dx**

push **es**

push **bp**

push **cx**

push **di**

push **bx**

in **al**, 060h

cmp **al**, 02ah

mov **ah**, 0

je do_req

mov **ah**, 035h

mov **al**, vec_csip_info

int 21h

push es

push bx

pushf

mov bp, sp

call dword ptr [bp + 2]

add sp, 4

jmp finish_req

do_req:

in al, 061h

mov ah, al

or al, 080h

out 061h, al

xchg ah, al

out 061h, al

mov ds, ax

retry:

mov ah, 05h

mov cl, 'D'

xor ch, ch

int 016h

or al, al

```
jz finish_req

mov ax, 0040h
mov es, ax
mov si, 001ah
mov ax, es:[si]
mov si, 001ch
mov es:[si], ax
jmp retry
```

```
finish_req:
```

```
pop bx
pop di
pop cx
pop bp
pop es
pop dx
pop ds
```

```
cli
mov ss, keep_ss
mov sp, keep_sp
sti
```

```
mov al, 020h
out 020h, al
```

pop ax

iret

resident_begin endp

last_byte label word

unload_handler proc near

push ds

push ax

push es

push bx

push dx

mov dx, 0

mov ah, 025h

mov al, vec_set_info

int 21h

mov ah, 035h

mov al, 09h

int 21h

mov ah, 035h

mov al, vec_csip_info

int 21h

```
cli
mov dx, es
mov ds, dx
mov dx, bx
mov ah, 025h
mov al, 09h
int 21h
sti

mov ah, 035h
mov al, vec_seg_info
int 21h

mov ah, 049h
int 21h

mov es, bx
mov ah, 049h
int 21h

pop dx
pop bx
pop es
pop ax
pop ds
```

```

    ret
unload_handler endp

load_handler proc near
    push ax
    push dx
    push es
    push bx
    push dx

    mov al, vec_set_info
    mov dx, 1
    mov ah, 025h
    int 21h

    push ds
    mov dx, es:[02ch]
    mov ax, es
    mov ds, ax
    mov al, vec_seg_info
    mov ah, 025h
    int 21h
    pop ds

    mov ah, 035h
    mov al, 09h
    int 21h

```

```

    mov dx, es
    mov ds, dx
    mov dx, bx
    mov ah, 025h
    mov al, vec_csip_info
    int 21h

    cli

    mov ax, @code
    mov ds, ax
    mov dx, offset resident_begin
    mov ah, 025h
    mov al, 09h
    int 21h
    sti

    pop dx
    pop bx
    pop es
    pop dx
    pop ax
    ret
load_handler endp

check_handler proc near
    push bx

```

```

push es

mov ah, 035h
mov al, vec_set_info
int 21h

cmp bx, 1
je handler_is_set
mov ax, 0
jmp finish_check_handler

handler_is_set:
    mov ax, 1

finish_check_handler:

pop es
pop bx
ret
check_handler endp

already_label db 'Interrupt descriptor already set', 13,
    10, '$'
unset_option db '/un'
unset_size equ 3
vec_set_info equ 0ffh ; в этом векторе сохраним инфу
    стоит ли наше прерывание

```

vec_csip_info **equ** 0feh ; в этом векторе сохраним инфу
о прошлом хендлере

vec_seg_info **equ** 0fdh ; а в этом адрес среды для
освобождения

transient_begin:

mov ax, @code

cli

mov ss, ax

mov sp, offset istack

sti

mov ax, @code

mov ds, ax

mov cl, es:[080h]

cmp cl, 4

mov cx, unset_size

mov si, offset unset_option

mov di, 082h

repe cmpsb

jne unset_command_not_set

unset_command_set:


```

    call check_handler
    cmp ax, 1
    jne finish
    call unload_handler
    jmp finish

unset_command_not_set:
    call check_handler
    cmp ax, 1
    jne handler_unloaded

handler_loaded:
    mov dx, offset already_label
    mov ah, 09h
    int 21h
    jmp finish

handler_unloaded:
    call load_handler
    mov dx, offset last_byte
    mov ah, 031h
    int 21h

finish:
    mov ah, 04ch
    int 21h

```

```
end transient_begin
```