

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

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

Студент гр. 9383

Мосин К.К.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

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

Задание.

Шаг 1. Написать и отладить модуль типа .EXE, который выполняет такие же функции, как в программе ЛР4.

Шаг 2. Убедиться, что резидентный обработчик установлен.

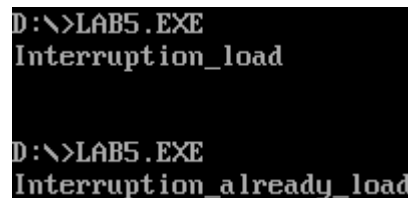
Шаг 3. Проверка размещения прерывания в памяти.

Шаг 4. Проверка установленного обработчика.

Шаг 5. Освобождение памяти.

Выполнение работы.

Загрузка резидентной функции и повторного запуска программы представлены на изображении 1.

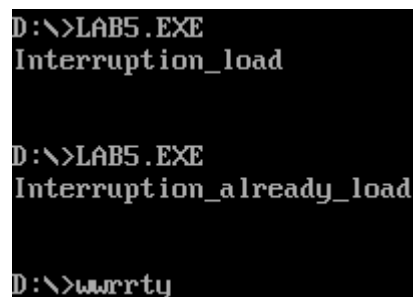


```
D:\>LAB5.EXE
Interruption_load

D:\>LAB5.EXE
Interruption_already_load
```

Рис. 1 - загрузка резидентной функции

Прерывание описано так, что при написании букв q и e они сменяются на w и r соответственно. Проверка проиллюстрирована на изображении 2.



```
D:\>LAB5.EXE
Interruption_load

D:\>LAB5.EXE
Interruption_already_load

D:\>wrrty
```

Рис. 2 - проверка работы

Проверка размещения обработчика в памяти представлена на рисунке 3.

```

D:\>LAB3.COM
available_memory: 644384 bytes
extended_memory: 960 Kbytes
memory_allocate error
table:
    4D      0088      16
    4D      0000      64
    4D      0000     256
    4D      0122     144
    4D      0122    4352   LAB5
    4D      02DD    1442
    5A      02DD   644384  LAB3

```

Рис. 3 - размещение в памяти

Контрольные вопросы.

1) Какого типа прерывания использовались в работе?

21h - программное прерывание выхода в DOS

10h - программное прерывание о получении информации о курсоре

09h - аппаратное прерывание клавиатуры

2) Чем отличается скан-код от кода ASCII?

Скан-код определяет код клавиши данной клавиатуры, подключенной к устройству, в то время как ASCII-код является общим кодом символа для всех компьютеров. Данный подход обеспечивает стандарты работы с символами для всех устройств.

Выводы.

В ходе выполнения лабораторной работы был построен пользовательский обработчик прерывания от клавиатуры.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab5.asm

```
stack_ segment stack
```

```
dw 128 dup(?)
```

```
stack_ ends
```

```
data_ segment
```

```
is_interruption_load db 'Interruption_already_load',0dh,0ah,0dh,0ah,'$'
```

```
interruption_load db 'Interruption_load',0dh,0ah,0dh,0ah,'$'
```

```
interruption_not_load db 'Interruption_not_load',0dh,0ah,0dh,0ah,'$'
```

```
interruption_delete db 'Interruption_was_delete',0dh,0ah,0dh,0ah,'$'
```

```
data_ ends
```

```
code_ segment
```

```
assume cs:code_, ds:data_, ss:stack_
```

```
rout proc far
```

```
jmp body
```

```
keep_cs dw 0
```

```
keep_ip dw 0
```

```
keep_psp dw 0
```

```
keep_ss dw 0
```

```
keep_sp dw 0
```

```
keep_ax dw 0
```

```
int_sig dw 9999h
```

```
int_seg dw 64 dup(?)
```

```
body:
```

```
mov keep_ss,ss
```

```

mov keep_sp,sp
mov keep_ax,ax

mov ax,seg int_seg
mov ss,ax
mov ax,offset body
mov sp,ax

mov ax,keep_ax

in al,60h
cmp al,10h
je input_w
cmp al,12h
je input_r

call dword ptr cs:[keep_ip]
jmp rout_end_ptr
input_w:
mov al,'w'
jmp do_req
input_r:
mov al,'r'

do_req:
push ax
in al, 61h
    mov ah, al
    or al, 80h
    out 61h, al
    xchg ah, al

```

```
    out 61h, al
    mov al, 20H
    out 20h, al
pop ax
```

```
loop_:
mov ah, 05h
    mov cl, al
    mov ch, 00h
    int 16h
    or al, al
    jz rout_end_ptr
    mov ax, 40h
    mov es, ax
    mov ax, es:[1ah]
    mov es:[1ch], ax
    jmp loop_
```

```
rout_end_ptr:
mov sp,keep_sp
mov ax,keep_ss
mov ss,ax
mov ax,keep_ax
mov al,20h
out 20h,al
iret
rout endp
```

```
setCurs proc
mov ah,02h
mov bh,0
```

```
mov dh,22
mov dl,0
int 10h
ret
setCurs endp
```

```
getCurs proc
mov ah,03h
mov bh,0
int 10h
ret
getCurs endp
```

```
deleteRout proc
cli
push ds
push es
```

```
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
```

```
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
```

```
pop es
```

```
pop ds
```

```
sti
```

```
mov dx,offset interruption_delete
```

```
call print
```

```
ret
```

```
deleteRout endp
```

```
print proc near
```

```
    push ax
```

```
    mov ah,09h
```

```
    int 21h
```

```
    pop ax
```

```
    ret
```

```
print endp
```



```

main proc far
mov ax,data_
mov ds,ax

push es

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

mov si,offset int_sig
sub si,offset rout
mov dx,es:[bx + si]
cmp dx,int_sig
jne interruption_not_loaded

pop es

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

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

mov al,es:[81h+3]
cmp al,'n'
jne bad_cfg

call deleteRout

```

jmp exit

interruption_not_loaded:

mov keep_psp,es

mov ah,35h

mov al,09h

int 21h

mov keep_cs,es

mov keep_ip,bx

push es

push bx

push ds

lea dx,rout

mov ax,seg rout

mov ds,ax

mov ah,25h

mov al,09h

int 21h

pop ds

pop bx

pop es

mov dx,offset interruption_load

call print

lea dx,rout_end_ptr

```
mov cl,4h
shr dx,cl
inc dx

add dx,100h
xor ax,ax

mov ah,31h
int 21h
jmp exit

bad_cfg:
mov dx,offset is_interruption_load
call print

exit:
xor al,al
mov ah,4ch
int 21h
main endp

code_ ends
end main
```