

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

ОТЧЕТ
по лабораторной работе №4
по дисциплине «Операционные системы»
Тема: Обработка стандартных прерываний

Студент гр. 9383

Мосин К.К.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

Построить обработчик прерываний сигналов таймера.

Задание.

Шаг 1. Написать и отладить модуль типа .EXE, который выполняет следующие функции:

- 1) Проверяет, установлено ли пользовательское прерывание с вектором 1Ch.
- 2) Устанавливает резидентную функцию и настраивает вектор прерываний.
- 3) Если прерывание установлено, то выводится соответствующее сообщение.
- 4) Выгрузка прерывания по значению параметра в командной строке “/un”.

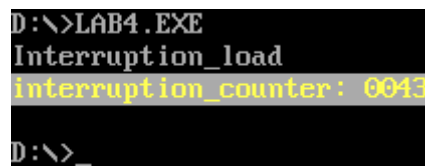
Шаг 2. Проверить установку обработчика, выведя MCB таблицу.

Шаг 3. Проверить, определяет ли программа установленный обработчик.

Шаг 4. Выгрузить обработчик.

Выполнение работы.

Был разработан модуль типа .EXE. Пример работы программы представлен на рисунке 1.



```
D:\>LAB4.EXE
Interruption_load
interruption_counter: 0013
D:\>_
```

Рис. 1 - Результат выполнения .COM модуля

При распечатке MCB таблицы наблюдается загруженный обработчик. Пример работы представлен на рисунке 2.

```
D:\>LAB3_1.COM
available_memory: 644416 bytes
extended_memory: 960 Kbytes
table:
    4D      0088      16
    4D      0000      64
    4D      0000     256
    4D      0122     144
    4D      0122    4320   LAB4
    4D      02BB    1440
    5A      02BB   644416  LAB3_1
interruption_counter: 1014
D:\>_
```

Рис. 2 - Проверка наличия обработчика

При попытке загрузить обработчик программа проверяет не был ли он уже загружен. Пример работы изображен на рисунке 3.

```
D:\>LAB4.EXE
Interruption_already_load
interruption_counter: 1679
D:\>
```

Рис. 3 - Программа определила, что обработчик уже загружен

Пример удаления обработчика представлен на рисунке 4.

```
D:\>LAB4.EXE /un
Interruption_was_delete
D:\>_
```

Рис. 4 - Выгрузка обработчика

Контрольные вопросы.

1) Как реализован механизм прерывания от часов?

Сохраняется содержимое регистров, определяется смещение и вызывается обработчик прерывания по сохраненному адресу. Затем управление передается прерванной программе.

2) Какого типа прерывания использовались в работе?

1Ch, 10h и 21h, где 1Ch аппаратное прерывание, а 10h и 21h - программное.

Выводы.

В ходе выполнения лабораторной работы был построен обработчик сигналов таймера.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab4.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_counter db 'interruption_counter: 0000$'
```

```
int_sig dw 9999h
```

```
int_seg dw 16 dup(?)
```

```
body:
```

```
mov keep_sp,sp
```

```
mov keep_ax,ax
```

```
mov ax,ss
```

mov keep_ss,ax

mov ax,keep_ax

mov sp,offset body

mov ax,seg int_seg

mov ss,ax

push ax

push cx

push dx

call getCurs

push dx

call setCurs

push si

push cx

push ds

push bp

mov ax,seg int_counter

mov ds,ax

mov si,offset int_counter

add si,21

mov cx,4

loop_:

mov bp,cx

mov ah,[si+bp]

inc ah

mov [si+bp],ah

cmp ah,3ah

jne update

mov ah,30h

mov [si+bp],ah

loop loop_

update:

pop bp

pop ds

pop cx

pop si

push es

push bp

mov ax,seg int_counter

mov es,ax

mov ax,offset int_counter

mov bp,ax

mov ah,13h

mov al,0

mov bh,0

mov cx,26

int 10h

pop bp

pop es

pop dx

mov ah,2

mov bh,0

int 10h

pop dx

pop cx

pop ax

mov keep_ax,ax

mov sp,keep_sp

mov ax,keep_ss

```
mov ss,ax
mov ax,keep_ax
```

```
mov al,20h
out 20h,al
iret
rout_end_ptr:
rout endp
```

```
outputAL proc
push ax
push bx
push cx
mov ah,09h
mov bh,0
mov cx,1
int 10h
pop cx
pop bx
pop ax
ret
outputAL endp
```

```
outputBP proc
push ax
push bx
push dx
push cx
mov ah,13h
mov al,1
mov bh,0
mov dh,22
mov dl,0
int 10h
pop cx
```



```
pop dx
pop bx
pop ax
ret
outputBP 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,1ch
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,1ch
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,1ch
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,1ch
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,1ch
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
```