

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

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

Студент гр. 8382

Щемель Д.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

## **Цель работы**

В архитектуре компьютера существуют стандартные прерывания, за которыми закреплены определённые вектора прерываний. При возникновении прерывания, аппаратура компьютера передаёт управление по соответствующему адресу вектора прерывания. Обработчик прерываний получает управление и выполняет соответствующие действия.

В лабораторной работе №4 предлагается построить обработчик прерываний сигналов таймера. Эти сигналы генерируются аппаратурой через определенные интервалы времени и, при возникновении такого сигнала, возникает прерывание с определённым значением вектора. Таким образом, управление будет передано функции, чья точка входа записана в соответствующий вектор прерывания.

## **Ход выполнения работы**

Был написан исходный код для .EXE-модуля, выполняющий следующие действия:

1. Получает текущий обработчик прерывания вектора прерывания сигнала таймера
2. Если установлен стандартный обработчик - выводится соответствующее сообщение, устанавливается пользовательский, а программа переходит в резидентный режим.
3. Если пользовательский обработчик уже установлен - выводится сообщение и повторная загрузка не происходит
4. Если при загруженном пользовательском обработчике запустить программу с флагом “/un” - обработчик заменяется стандартным, а память, выделенная под пользовательский, освобождается

Пользовательский обработчик прерываний при возникновении прерывания увеличивает внутренний счётчик и выводит его значение на экран.

Результат работы модуля при первом запуске:

Результат работы программы при повторном запуске с указанием флага “/un”

```

C:\>1.EXE
Handler is not loaded
00850

C:\>3.COM
644640
15360
Type = 4D; Owner = 0008; Size = 000016; Last 8 bytes = ;
Type = 4D; Owner = 0000; Size = 000064; Last 8 bytes = ;
Type = 4D; Owner = 0040; Size = 000256; Last 8 bytes = ;
Type = 4D; Owner = 0192; Size = 000144; Last 8 bytes = ;
Type = 4D; Owner = 0192; Size = 004096; Last 8 bytes = 1 ;
Type = 4D; Owner = 029D; Size = 004144; Last 8 bytes = ;
Type = 4D; Owner = 029D; Size = 065536; Last 8 bytes = 3 ;
Type = 5A; Owner = 0000; Size = 579088; Last 8 bytes = 04140

C:\>_

```

Figure 1: Работа при первом запуске

```

C:\>1.EXE /un
Handler is loaded

C:\>3.COM
648912
15360
Type = 4D; Owner = 0008; Size = 000016; Last 8 bytes = ;
Type = 4D; Owner = 0000; Size = 000064; Last 8 bytes = ;
Type = 4D; Owner = 0040; Size = 000256; Last 8 bytes = ;
Type = 4D; Owner = 0192; Size = 000144; Last 8 bytes = ;
Type = 4D; Owner = 0192; Size = 065536; Last 8 bytes = 3 ;
Type = 5A; Owner = 0000; Size = 583360; Last 8 bytes = ;

C:\>

```

Figure 2: Результат работы

## **Контрольные вопросы**

1. Как устроен механизм прерываний от часов?

С частотой 18.2Hz PIT(Programmable interval timer) через IRQ-0 вызывает системное прерывание 08, которое обновляет часы. После чего вызывается прерывание 1ch.

2. Какого типа прерывания использовались в работе?

В ручную вызывались только программные прерывания 1ch, 10h, 21h.

## **Вывод**

В ходе выполнения лабораторной работы был реализован пользовательский обработчик прерывания, на примере сигналов таймера. А так же были получены навыки написания резидентных программ (с последующей их выгрузкой из памяти).

## ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

```
_STACK SEGMENT STACK
```

```
    DW 100h DUP(0)
```

```
_STACK ENDS
```

```
DATA SEGMENT
```

```
HANLDER_IS_LOADED db 'Handler is loaded', 13, 10, '$'
```

```
HANLDER_IS_NOT_LOADED db 'Handler is not loaded', 13, 10, '$'
```

```
UNLOAD_FLAG db '/un', 13, 10, '$'
```

```
DATA ENDS
```

```
CODE SEGMENT
```

```
ASSUME CS:CODE, DS:DATA, SS:_STACK
```

```
HANDLER PROC far
```

```
    jmp HANDLER_CODE
```

```
HANDLER_UNIQUE_CODE dw 1111h
```

```
HANDLER_DATA:
```

```
    HANDLER_STACK dw 100h dup(0)
```

```
    KEEP_CS dw 0
```

```
    KEEP_IP dw 0
```

```
    KEEP_PSP dw 0
```

```
    KEEP_AX dw 0
```

```
    KEEP_SS dw 0
```

```
    KEEP_SP dw 0
```

```
    COUNTER dw 0
```

```
    COUNTER_STRING db '00000$'
```

```
HANDLER_CODE:
```

```
    mov KEEP_AX, ax
```

```
    mov KEEP_SS, ss
```

```
    mov KEEP_SP, sp
```

```
    mov ax, seg HANDLER_STACK
```

```
    mov ss, ax
```

```

mov sp, offset KEEP_CS
push es
push ds
push dx
push si

mov ax, seg HANDLER_DATA
mov ds, ax

inc COUNTER
mov dx, 0
mov ax, COUNTER
mov si, offset COUNTER
add si, 5
call WRD_TO_DEC

mov ax, seg COUNTER_STRING
mov es, ax
mov bp, offset COUNTER_STRING
call OUTPUT_BP

mov al, 20h
out 20h, al

pop si
pop dx
pop ds
pop es
mov ss, KEEP_SS
mov sp, KEEP_SP
mov ax, KEEP_AX
iret

```

HANDLER ENDP

WRD\_TO\_DEC PROC near

```
; IN si - end of string. ax - word
```

```
push bx
```

```
push ax
```

```
mov bx, 10
```

```
DIVISION:
```

```
div bx
```

```
add dl, 30h
```

```
mov [si], dl
```

```
xor dx, dx
```

```
dec si
```

```
cmp ax, 0
```

```
jne DIVISION
```

```
pop ax
```

```
pop bx
```

```
ret
```

```
WRD_TO_DEC ENDP
```

```
OUTPUT_BP PROC near
```

```
; IN ES:BP
```

```
push ax
```

```
push bx
```

```
push dx
```

```
push cx
```

```
mov ah, 13h
```

```
mov al, 0
```

```
mov bh, 0
```

```
mov dh, 22
```

```
mov dl, 60
```

```
mov cx, 5
```

```
int 10h
```

```
pop cx
```

```

    pop dx
    pop bx
    pop ax
    ret
OUTPUT_BP ENDP

CHECK_IS_HANDLER_ALREADY_LOADED PROC near
; OUT al. 1 - true
    push es
    push bx
    push cx
    push si

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

    mov al, 0
    mov si, offset HANDLER_UNIQUE_CODE
    sub si, offset HANDLER
    mov cx, es:[bx+si]
    cmp cx, HANDLER_UNIQUE_CODE
    jne FINISH_CHECK_IS_HANDLER_ALREADY_LOADED

    mov al, 1

FINISH_CHECK_IS_HANDLER_ALREADY_LOADED:
    pop si
    pop cx
    pop bx
    pop es
    ret
CHECK_IS_HANDLER_ALREADY_LOADED ENDP

LOAD_HANDLER PROC near

```



```

    mov KEEP_PSP, es
    mov ah, 35h
    mov al, 1ch
    int 21h
    mov KEEP_CS, es
    mov KEEP_IP, bx

    push ds
    mov dx, offset HANDLER
    mov ax, seg HANDLER
    mov ds, ax
    mov ah, 25h
    mov al, 1ch
    int 21h
    pop ds

    mov dx, 100h
    mov ax, 3100h
    int 21h
    ret
LOAD_HANDLER ENDP

CHECK_FLAG PROC near
; OUT al. 1 - true
    push cx
    push si
    push bx

    mov al, 1
    mov cx, 3
    mov si, offset UNLOAD_FLAG
    mov bx, 82h
CHECK_FLAG_INNER_LOOP:
    mov ah, [si]
    cmp byte ptr es:[bx], ah

```

```

        jne SET_FALSE_AND_FINISH
        inc si
        inc bx
        mov al, 1
        loop CHECK_FLAG_INNER_LOOP
        jmp FINISH_CHECK_FLAG
SET_FALSE_AND_FINISH:
        mov al, 0
        jmp FINISH_CHECK_FLAG

FINISH_CHECK_FLAG:
        pop bx
        pop si
        pop cx
        ret
CHECK_FLAG ENDP

```

```

UNLOAD_HANDLER PROC near

```

```

        cli
        push ax
        push bx
        push ds
        push si
        push es

```

```

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

```

```

        mov si, offset KEEP_CS
        sub si, offset HANDLER
        mov ax, es:[bx+si]
        mov si, offset KEEP_IP
        sub si, offset HANDLER
        mov dx, es:[bx+si]

```

```

push ds
mov ds, ax
mov ah, 25h
mov al, 1ch
int 21h
pop ds

mov si, offset KEEP_PSP
sub si, offset HANDLER

mov es, es:[bx+si]
mov ah, 49h
int 21h

mov es, es:[2ch]
mov ah, 49h
int 21h

pop es
pop si
pop ds
pop bx
pop ax
sti
ret
UNLOAD_HANLDER ENDP

UNLOAD_HANDLER_IF_FLAG PROC near
; IN /un in cmd
push ax

call CHECK_FLAG
cmp al, 1
jne FINISH_UNLOAD_HANDLER_IF_FLAG

```

```

    call UNLOAD_HANDLER

    FINISH_UNLOAD_HANDLER_IF_FLAG:

    pop ax
    ret

UNLOAD_HANDLER_IF_FLAG ENDP

PRINT_IS_HANDLER_LOADED PROC near
; IN al

    push dx
    cmp al, 1
    je LOADED
    jmp NOT_LOADED

LOADED:
    mov dx, offset HANDLER_IS_LOADED
    jmp DO_PRINT

NOT_LOADED:
    mov dx, offset HANDLER_IS_NOT_LOADED

DO_PRINT:
    call PRINT

    pop dx
    ret

PRINT_IS_HANDLER_LOADED ENDP

EXIT PROC near
    xor AL, AL
    mov AH, 4ch
    int 21h
    ret

EXIT ENDP

PRINT PROC near
    push ax
    mov ah, 09h

```

```

    int 21h
    pop ax
    ret
PRINT ENDP

MAIN:
    mov ax, DATA
    mov ds, ax
    call CHECK_IS_HANDLER_ALREADY_LOADED
    call PRINT_IS_HANDLER_LOADED
    cmp al, 0
    je LOAD
    jmp UNLOAD_IF_FLAG
LOAD:
    call LOAD_HANDLER
    jmp FINISH_MAIN
UNLOAD_IF_FLAG:
    call UNLOAD_HANDLER_IF_FLAG

FINISH_MAIN:
    call EXIT
LAST_HANDLER_BYTE:
CODE ENDS
END MAIN

```