

**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**

Студентка гр. 0382

\_\_\_\_\_

Бочаров Г.С.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2022

### **Цель работы.**

В архитектуре компьютера существуют стандартные прерывания, за которыми закреплены определенные вектора прерываний. Вектор прерываний хранит адрес подпрограммы обработчика прерываний. При возникновении прерывания, аппаратура компьютера передает 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 определенным значением вектора. Таким образом, управление будет передано функции, чья точка входа записана в соответствующий вектор прерывания.

### **Задание.**

**Шаг 1.** Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет следующие функции:

- 1) Проверяет, установлено ли пользовательское прерывание с вектором 1Ch.
- 2) Устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено, и 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.
- 4) Выгрузка прерывания по соответствующему значению параметра в командной строке /un. Выгрузка прерывания состоит в восстановлении стандартного вектора прерываний и освобождении памяти, занимаемой

резидентом. Затем осуществляется выход по функции 4Ch прерывания int 21h.

Для того, чтобы проверить установку прерывания, можно поступить следующим образом. Прочитать адрес, записанный в векторе прерывания. Предположим, что этот адрес указывает на точку входа в установленный резидент. На определенном, известном смещении в теле резидента располагается сигнатура, некоторый код, который идентифицирует резидент. Сравнив известное значение сигнатуры с реальным кодом, находящимся в резиденте, можно определить, установлен ли резидент. Если значения совпадают, то резидент установлен. Длину кода сигнатуры должна быть достаточной, чтобы сделать случайное совпадение маловероятным.

Программа должна содержать код устанавливаемого прерывания в виде удаленной процедуры. Этот код будет работать после установки при возникновении прерывания. Он должен выполнять следующие функции:

- 1) Сохраняет стек прерванной программы (регистры SS и SP) в рабочих переменных и восстановить при выходе.
- 2) Организовать свой стек.
- 3) Сохранить значения регистров в стеке при входе и восстановить их при выходе.
- 4) При выполнении тела процедуры накапливать общее суммарное число прерываний и выводить на экран. Для вывода на экран следует использовать прерывание int 10h, которое позволяет непосредственно выводить информацию на экран.
- 5) Функция прерывания должна содержать только переменные, которые она использует.

**Шаг 2.** Запустите отлаженную программу и убедитесь, что резидентный обработчик прерывания 1Ch установлен. Работа прерывания должна отображаться на экране, а также необходимо проверить размещение прерывания в памяти. Для этого запустите программу ЛР 3, которая

отображает карту памяти в виде списка блоков МСВ. Полученные результаты поместите в отчет.

**Шаг 3.** Запустите отлаженную программу еще раз и убедитесь, что программа определяет установленный обработчик прерываний. Полученные результаты поместите в отчет.

**Шаг 4.** Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

**Шаг 5.** Ответьте на контрольные вопросы.

### **Выполнение работы.**

Функции, написанные в ходе выполнения:

1. *print\_word* — выводит строку, из регистра dx
2. *interrupt* - обработчик прерывания таймера.
3. *load\_interrupt* — загружает функцию обработчика прерывания в память и оставляет ее резидентной.
4. *unload\_interrupt*— восстанавливает исходный обработчик прерывания и освобождает память.
5. *check\_cmd\_tail* — проверяет хвост командной строки.
6. *check\_interrupt*— проверяет, загружено ли прерывание в память.

### **Шаг 1.**

На первом шаге был написан .EXE модуль, меняющий прерывание таймера и выводящий на экран кол-во вызовов прерывания.

Рисунок 1: результат работы .EXE модуля

## Шаг 2.

Рисунок 2: результат работы .COM модуля lb3

На втором шаге был запущен модуль .EXE, а также была запущена программа из ЛР3. Видно что прерывание действительно загружено в память. МСВ №10.

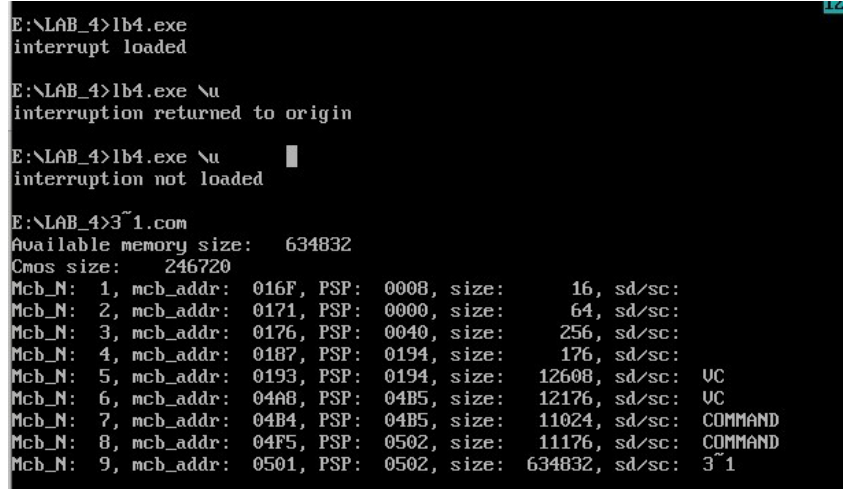
## Шаг 3.

Программа была запущена несколько раз. В первый раз устанавливается обработчик прерывания, в остальные разы на экран выводится сообщение, что обработчик уже выставлен.

Рисунок 3: результат нескольких запусков программы.

#### Шаг 4.

На четвертом шаге программа запущена с ключом выгрузки (\u). Также была запущена программа, из ЛР3. Память выделенная под обработчик была освобождена.



```
E:\LAB_4>lb4.exe
interrupt loaded

E:\LAB_4>lb4.exe \u
interruption returned to origin

E:\LAB_4>lb4.exe \u
interruption not loaded

E:\LAB_4>3~1.com
Available memory size: 634832
Cmos size: 246720
Mcb_N: 1, mcb_addr: 016F, PSP: 0008, size: 16, sd/sc:
Mcb_N: 2, mcb_addr: 0171, PSP: 0000, size: 64, sd/sc:
Mcb_N: 3, mcb_addr: 0176, PSP: 0040, size: 256, sd/sc:
Mcb_N: 4, mcb_addr: 0187, PSP: 0194, size: 176, sd/sc:
Mcb_N: 5, mcb_addr: 0193, PSP: 0194, size: 12608, sd/sc: UC
Mcb_N: 6, mcb_addr: 04A8, PSP: 04B5, size: 12176, sd/sc: UC
Mcb_N: 7, mcb_addr: 04B4, PSP: 04B5, size: 11024, sd/sc: COMMAND
Mcb_N: 8, mcb_addr: 04F5, PSP: 0502, size: 11176, sd/sc: COMMAND
Mcb_N: 9, mcb_addr: 0501, PSP: 0502, size: 634832, sd/sc: 3 1
```

Рисунок 4: результат работы программы с флагом выгрузки.

#### Шаг 5. Ответы на контрольные вопросы.

1. Как реализован механизм прерывания от часов?

Центральный процессор приостанавливает выполнение приоритетной программы для обработки прерывания, которое поступило от таймера. Прерывание таймера генерируется примерно 18 раз в секунду, вызывается с помощью 1CH — системного таймера. Обработчик прерывания можно менять установив его в векторе прерывания. Таким образом вместо встроеной функции обработки будет вызвана пользовательская.

2. Какого типа прерывания использовались в работе?

1CH — аппаратное прерывание. 10H и 21H — программные прерывания.

#### Выводы.

В ходе работы были изучены структуры обработчиков стандартных прерываний, был реализован обработчик прерываний сигналов таймера.

## ПРИЛОЖЕНИЕ А

### КОД МОДУЛЕЙ

Название файла: lb4.asm

```
assume cs:code, ds:data, ss:stack

stack segment stack
    dw 128 dup(?)
stack ends

code segment

word_to_dec proc near    ; input ax !, output ds:si
    push cx
    push dx
    push ax
    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 ax
    pop dx
    pop cx
    ret
word_to_dec endp

get_curs proc near

    push ax
    push bx

    mov ah, 03h
    mov bh, 0
    int 10h

    pop bx
    pop ax

    ret
get_curs endp
```

```
set_curs proc near
```

```
    push ax
    push bx
    mov ah, 02h
    mov bh, 0
    int 10h
```

```
    pop bx
    pop ax
```

```
    ret
set_curs endp
```

```
outputbp proc    ;es:bp
```

```
    push ax
    push bx
    push dx
    push cx
    mov ah,13h ; ôóíêöèÿ
    mov al,1 ; sub function code
    mov bh,0 ; âèääî ñòðàíèöà
    ;mov dh,22 ; dh,dl = ñòðíêà, êíêííêà (ñ÷èòàÿ îò 0)
    ;mov dl,0
    int 10h
    pop cx
    pop dx
    pop bx
    pop ax
```

```
    ret
outputbp endp
```

```
interrupt proc far
    jmp start_int
```

```
    interrupt_id    dw 0c204h
    keep_cs          dw 0
    keep_ip          dw 0
```

```
    keep_sp          dw 0
    keep_ss          dw 0
    keep_ax          dw 0
    keep_psp         dw 0
```

```
    count            dw 0
    count_message    db 'interrupt call count:    $'
    new_stack        dw 128 dup(?)
```

```
start_int:
    ; save
    mov keep_sp, sp
    mov keep_ax, ax
    mov keep_ss, ss
```



```

; stack
mov sp, offset start_int
mov ax, seg new_stack
mov ss, ax

;get_curs
push ax
push bx
push cx
push dx

call get_curs
push dx ; save curent curs

;inc count
push si
push ds

mov ax, seg count

mov ds, ax

mov si, offset count

mov ax, [si]

inc ax;

mov [si], ax

mov si, offset count_message ; same seg
add si, 25

call word_to_dec ; print count in dec to str

print_and_ret:

pop ds
pop si

; print es:bp
push es
push bp

mov ax, seg count_message

mov es, ax
mov ax, offset count_message

mov bp, ax

; print
mov ah, 13h
mov al, 1
mov bh, 0

```

```

mov     cx, 29
mov     dx, 0
int     10h

    pop bp
    pop es

    pop dx

    call set_curs

    pop dx
    pop cx
    pop bx
    pop ax

    mov ss, keep_ss
    mov ax, keep_ax
    mov sp, keep_sp

mov     al, 20h
out     20h, al

    iret
    my_int_end :
interrupt endp

load_interrupt proc near
    push ax
    push bx
    push dx
    push es

    mov ah, 35h
    mov al, 1ch
    int 21h
    mov keep_ip, bx
    mov keep_cs, es        ; old int

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

    mov dx, offset interrupt_successfully_loaded
    call print_word

    mov dx, offset my_int_end ; mk resident

```

```

        mov cl,4
        shr dx,cl
        inc dx
        mov ax, cs
        sub ax, keep_psp
        add dx, ax
        xor ax, ax
        mov ah,31h
        int 21h                                ; exit dos

        pop es
        pop dx
        pop bx
        pop ax
        ret
load_interrupt endp

unload_interrupt proc near
        push ax
        push bx
        push dx
        push es

        mov ah, 35h
        mov al, 1ch
        int 21h; es:bx - int adr

        ; restore old int
        cli
        push ds
        mov dx, es:[keep_ip]
        mov ax, es:[keep_cs]
        mov ds, ax
        mov ah, 25h
        mov al, 1ch
        int 21h

        pop ds
        sti

        mov dx, offset returned_original_interrupt
        call print_word

        ; mem free
        mov ax, es:[keep_psp]
        mov es, ax
        push es
        mov ax, es:[2ch]
        mov es, ax
        mov ah, 49h
        int 21h
        pop es
        int 21h

```

```

        pop es
        pop dx
        pop bx
        pop ax

        ret
unload_interrupt endp

check_interrupt proc near ; al  0 no, 1 - yes
    push bx
    push dx
    push es

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

    mov si, offset interrupt_id
    sub si, offset interrupt
    mov dx, es:[bx + si]
    mov al, 0
    cmp dx, 0c204h ; signature
    jne fin_

int_set_:
    mov al, 1

fin_:
    pop es
    pop dx
    pop bx

    ret
check_interrupt endp

check_cmd_tai proc near
    ; al  0 no, 1 - yes
    push bx

    mov al, 0
    mov bh, es:[82h]    ; es:[81h] cmd tail
    cmp bh, '\'
    jne end_
    mov bh, es:[83h]
    cmp bh, 'u'
    jne end_
    mov al, 1

end_:

    pop bx

```

```

        ret

check_cmd_tai endp

print_word proc near
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
print_word endp

main proc far

    mov ax, data
    mov ds, ax
    mov keep_psp, es

    call check_cmd_tai
    cmp al, 1
    je start_unload_int

    call check_interrupt
    cmp al, 1
    jne start_load

    mov dx, offset interrupt_already_loaded
    call print_word
    jmp endl

start_load:
    call load_interrupt

start_unload_int:

    call check_interrupt
    cmp al, 0
    je interrupt_not_loaded_
    call unload_interrupt
    jmp endl

interrupt_not_loaded_:
    mov dx, offset interrupt_not_loaded
    call print_word
    jmp endl

endl:
    mov ah, 4ch
    int 21h

main endp
code ends

data segment

```

interrupt_successfully_loaded db	'interrupt loaded',
0dh, 0ah, '\$'	
interrupt_already_loaded db	'interruption already
loaded', 0dh, 0ah, '\$'	
returned_original_interrupt db	'interruption returned to
origin', 0dh, 0ah, '\$'	
interrupt_not_loaded db	'interruption not loaded',
0dh, 0ah, '\$'	
data ends	
end main	