

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

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Операционные системы»
Тема: Обработка стандартных прерываний

Студент гр.8382

Синельников М.Р

Преподаватель

Ефремов М.А

Санкт-Петербург

2020

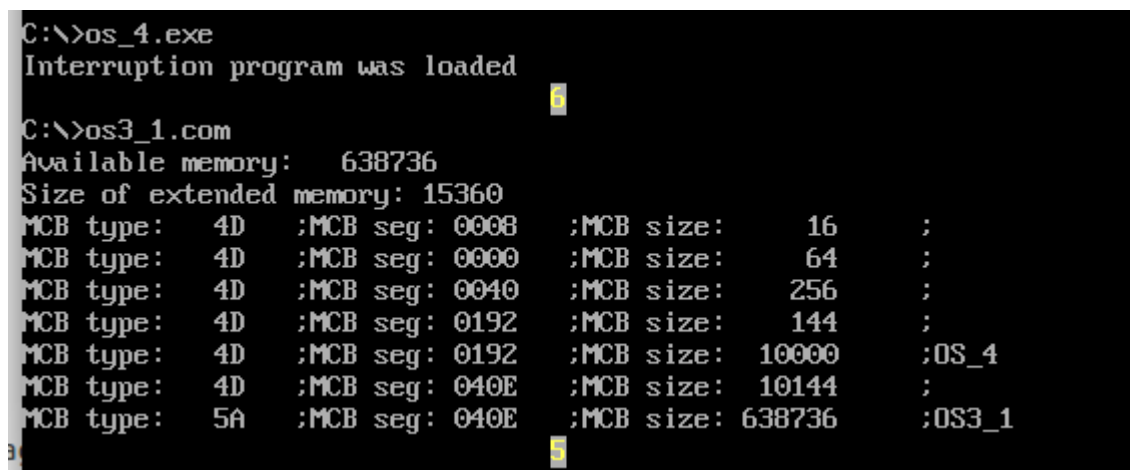
Цель работы.

В архитектуре компьютера существуют стандартные прерывания, за которыми закреплены определённые вектора прерываний. Вектор прерываний хранит адрес подпрограммы обработчика прерываний. При возникновении прерывания, аппаратура компьютера передаёт управление 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 определённые 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 работы.

1) Шаг 1



```
C:\>os_4.exe
Interruption program was loaded

C:\>os3_1.com
Available memory: 638736
Size of extended memory: 15360
MCB type: 4D ;MCB seg: 0008 ;MCB size: 16 ;
MCB type: 4D ;MCB seg: 0000 ;MCB size: 64 ;
MCB type: 4D ;MCB seg: 0040 ;MCB size: 256 ;
MCB type: 4D ;MCB seg: 0192 ;MCB size: 144 ;
MCB type: 4D ;MCB seg: 0192 ;MCB size: 10000 ;OS_4
MCB type: 4D ;MCB seg: 040E ;MCB size: 10144 ;
MCB type: 5A ;MCB seg: 040E ;MCB size: 638736 ;OS3_1
```

памяти после

рисунок 1 — счётчик и состояние загрузки обработчика прерывания

2) Шаг 2

```
Available memory: 638736
Size of extended memory: 15360
MCB type: 4D ;MCB seg: 0008 ;MCB size: 16 ;
MCB type: 4D ;MCB seg: 0000 ;MCB size: 64 ;
MCB type: 4D ;MCB seg: 0040 ;MCB size: 256 ;
MCB type: 4D ;MCB seg: 0192 ;MCB size: 144 ;
MCB type: 4D ;MCB seg: 0192 ;MCB size: 10000 ;OS_4
MCB type: 4D ;MCB seg: 040E ;MCB size: 10144 ;
MCB type: 5A ;MCB seg: 040E ;MCB size: 638736 ;OS3_1

C:\>os_4.exe
Interruption program is already loaded

C:\>_
```

рисунок 2 - повторная загрузка обработчика
прерывания

3) Шаг 3

```
C:\>os_4.exe/un
Interruption program unloaded

C:\>os3_2.com
Available memory: 648912
Size of extended memory: 15360
MCB type: 4D ;MCB seg: 0008 ;MCB size: 16 ;
MCB type: 4D ;MCB seg: 0000 ;MCB size: 64 ;
MCB type: 4D ;MCB seg: 0040 ;MCB size: 256 ;
MCB type: 4D ;MCB seg: 0192 ;MCB size: 144 ;
MCB type: 4D ;MCB seg: 0192 ;MCB size: 12576 ;OS3_
MCB type: 5A ;MCB seg: 0000 ;MCB size: 636320 ;
```

рисунок 3 — выгрузка обработчика и состояние памяти

Контрольные вопросы.

1) Как реализован механизм прерывания от часов?

Каждые 55 мс сначала сохраняется состояние регистров, затем определяется источник прерывания, который определяет в свою очередь адрес вектора прерывания в таблице векторов прерываний. Первые два байта помещаются в регистр IP, а вторые два байта – в CS. Затем управление передаётся по адресу CS:IP и происходит обработка соответствующего

прерывания. После завершения обработки управление возвращается прерванной программе.

2) Какого типа прерывания использовались в программе?

При выполнении работы использовались программные прерывания `int 21h`, `int 10h`, а также аппаратное прерывание `int 1Ch`, возникающее каждые 55 мс по системному таймеру

Вывод.

В ходе выполнения работы был реализован обработчик прерываний сигналов системного таймера.

Приложение А

Исходный код файла OS_4.asm

```
CODE SEGMENT
keep_1c      dd 0
keep_2f      dd 0
keep_PSP     dw ?
ASSUME CS:CODE, DS:DATA, SS:AStack

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

getCurs PROC
    push ax
    push bx
    mov ah, 03h
    mov bh, 0
    int 10h
    pop bx
    pop ax
    ret
getCurs ENDP

SetCurs PROC
    push ax
    push bx
    mov ah, 02h
    mov bh, 0
```

```

        int 10h
        pop bx
        pop ax
        ret
SetCurs ENDP

2F      PROC
        cmp     ah, 080h
        jne not_loaded
        mov al, 0ffh
not_loaded:
        iret
2F      ENDP

1C      PROC
        push ax
        push bx
        push cx
        push dx
        push es
        inc count
        cmp count, 57
        jne show
        mov count, 48
show:

        call getCurs
        mov cx, dx
        mov dh, 23
        mov dl, 33
        call SetCurs
        push ax
        mov al, count
        call OutputAL
        pop ax
        mov dx, cx
        call SetCurs
        mov al, 20h
        out 20h, al
        pop es
        pop dx
        pop cx

```

```

        pop bx
        pop ax
        iret
LAST_BYTE:
1C      ENDP

Un_check PROC      FAR
        push ax

        mov     ax, Keep_PSP
        mov     es, ax
        sub     ax, ax
        cmp     byte ptr es:[82h], '/'
        jne     not_un
        cmp     byte ptr es:[83h], 'u'
        jne     not_un
        cmp     byte ptr es:[84h], 'n'
        jne     not_un
        mov     flag, 0

not_un:
        pop     ax
        ret

Un_check ENDP

Keep_interr PROC
        push ax
        push bx
        push es
        mov     ah, 35h
        mov     al, 1Ch
        int     21h
        mov     word ptr keep_1c, bx
        mov     word ptr keep_1c+2, es
        mov     ah, 35h
        mov     al, 2Fh
        int     21h
        mov     word ptr keep_2f, bx
        mov     word ptr keep_2f+2, es
        pop     es
        pop     bx
        pop     ax

```

```

ret
Keep_interr    ENDP

Load_interr    PROC
push ds
push dx
push ax
call Keep_interr
push ds
mov dx, offset 1C
mov ax, seg 1C
mov ds, ax
mov ah, 25h
mov al, 1Ch
int 21h
mov dx, offset 2F
mov ax, seg 2F
mov ds, ax
mov ah, 25h
mov al, 2Fh
int 21h
pop ds
pop ax
pop dx
pop ds
ret
Load_interr    ENDP

Unload_interr  PROC
push ds
mov ah, 35h
mov al, 1Ch
int 21h
mov dx, word ptr es:keep_1c
mov ax, word ptr es:keep_1c+2
mov word ptr keep_1c, dx
mov word ptr keep_1c+2, ax

mov ah, 35h
mov al, 2Fh
int 21h
mov dx, word ptr es:keep_2f

```



```

mov ax, word ptr es:keep_2f+2
mov word ptr keep_2f, dx
mov word ptr keep_2f+2, ax
cli
mov dx, word ptr keep_1c
mov ax, word ptr keep_1c+2
mov ds, ax
mov ah, 25h
mov al, 1Ch
int 21h
mov dx, word ptr keep_2f
mov ax, word ptr keep_2f+2
mov ds, ax
mov ah, 25h
mov al, 2Fh
int 21h
sti
pop ds
mov es, es:Keep_PSP
mov ax, 4900h
int 21h
mov flag, 1
mov dx, offset Message2
call Write_message
mov es, es:[2ch]
mov ax, 4900h
int 21h
mov ax, 4C00h
int 21h

```

Unload_interr ENDP

Make_resident PROC

```

mov ax, es
mov Keep_PSP, ax
mov dx, offset LAST_BYTE
add dx, 200h
mov ah, 31h
mov al, 0
int 21h

```

Make_resident ENDP

; функция вывода сообщения на экран

```

Write_message PROC
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
Write_message ENDP

```

; Главная функция

```

Main PROC
    push ds
    xor ax, ax
    push ax
    mov ax, data
    mov ds, ax
    mov Keep_PSP, es
    mov Count, 48
    mov ax, 8000h
    int 2Fh
    cmp sal, 0ffh
    jne loading
    call Un_check
    cmp flag, 0
    jne alr_loaded
    call Unload_interr

loading:
    call Load_interr
    lea DX, Message1
    call Write_message
    call Make_resident

alr_loaded:
    lea dx, Message3
    call Write_message
    mov ax, 4C00h
    int 21h

Main ENDP
CODE ENDS

```

```

AStack SEGMENT STACK
    DW 256 DUP(?)
AStack ENDS

```

```

DATA          SEGMENT
               Count          db ?
               flag           dw 1
               Message1       db 'Interruption program was loaded', 0dh, 0ah, '$'
               Message2       db 'Interruption program unloaded', 0dh, 0ah, '$'
               Message3       db 'Interruption program is already loaded', 0dh, 0ah, '$'
DATA          ENDS
               END Main

```