

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

Крейсманн К.В.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

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 управления 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 прерывания с определенным значением вектора. Таким образом, управление будет передано функции, чья точка входа записана в соответствующий вектор прерывания.

Задание

Шаг1. 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 прерывания 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.

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) Сохранять значения регистров в стеке при входе и восстановить их при выходе.
- 2) При выполнении тела процедуры накапливать общее суммарное число прерываний и выводить на экран. Для вывода на экран следует использовать прерывания `int 10h`, которое позволяет непосредственно выводить информацию на экран.

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. Полученный 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

- Был написан и отлажен программный модуль типа .EXE, который выполняет следующие функции:

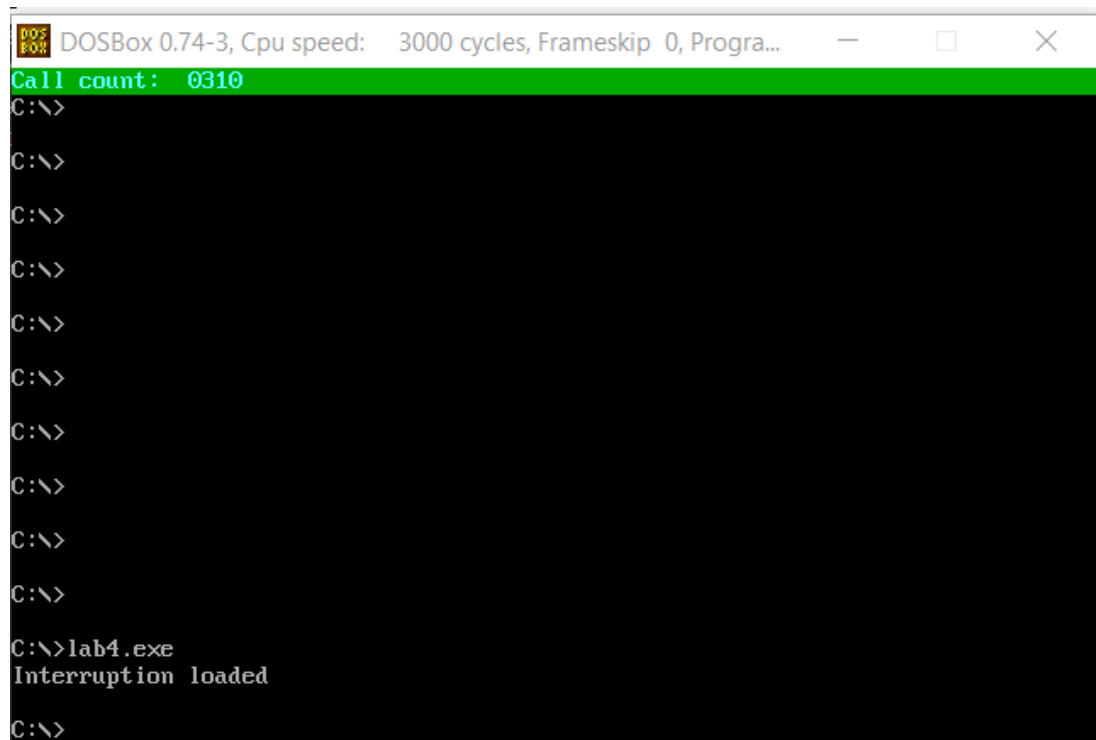
- 1) Проверяет, установлено ли пользовательское прерывание с вектором 1Ch.

- 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 прерывания 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.

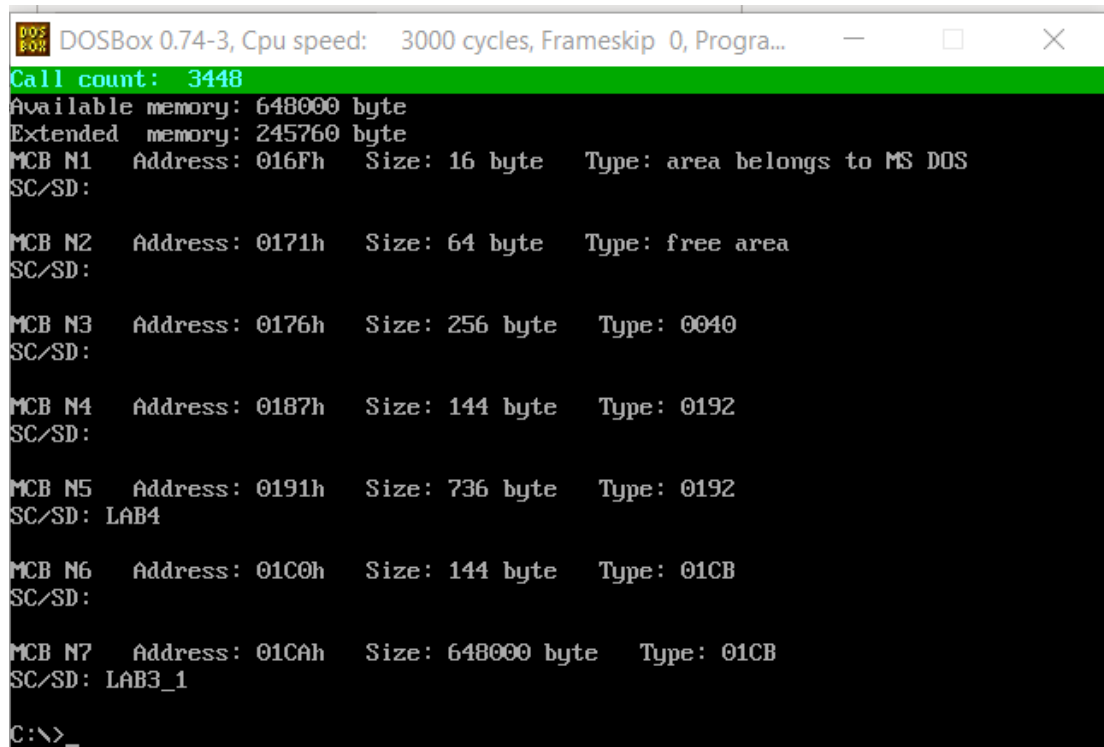
- Была запущена программа, работа прерывания отображается на экране. (Рисунок 1)



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
Call count: 0310
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>lab4.exe
Interruption loaded
C:\>
```

Рисунок 1 - Демонстрация работы модуля

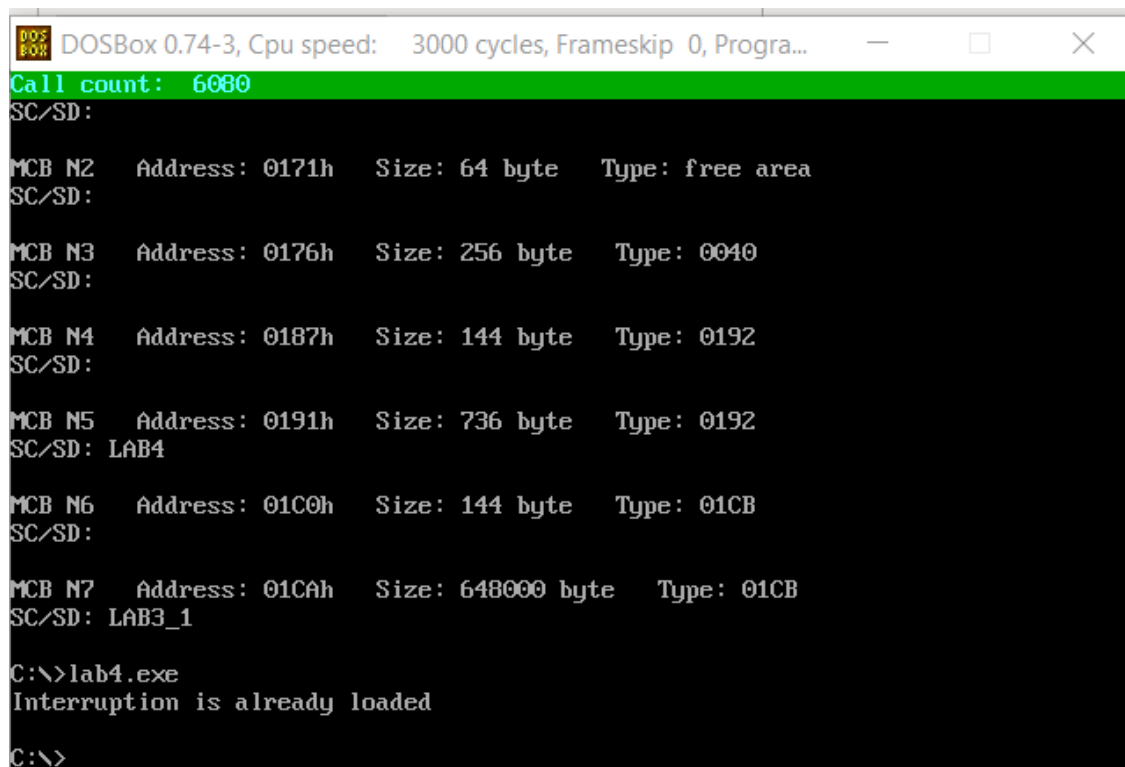
- Выполнена проверка размещения прерывания в памяти, для этого запущена программа lab3_1.com. (Рисунок 2)



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
Call count: 3448
Available memory: 648000 byte
Extended memory: 245760 byte
MCB N1 Address: 016Fh Size: 16 byte Type: area belongs to MS DOS
SC/SD:
MCB N2 Address: 0171h Size: 64 byte Type: free area
SC/SD:
MCB N3 Address: 0176h Size: 256 byte Type: 0040
SC/SD:
MCB N4 Address: 0187h Size: 144 byte Type: 0192
SC/SD:
MCB N5 Address: 0191h Size: 736 byte Type: 0192
SC/SD: LAB4
MCB N6 Address: 01C0h Size: 144 byte Type: 01CB
SC/SD:
MCB N7 Address: 01CAh Size: 648000 byte Type: 01CB
SC/SD: LAB3_1
C:\>_
```

Рисунок 2 - Проверка размещения в памяти

- Выполнена проверка, что программа определяет установленный обработчик. (Рисунок 3)



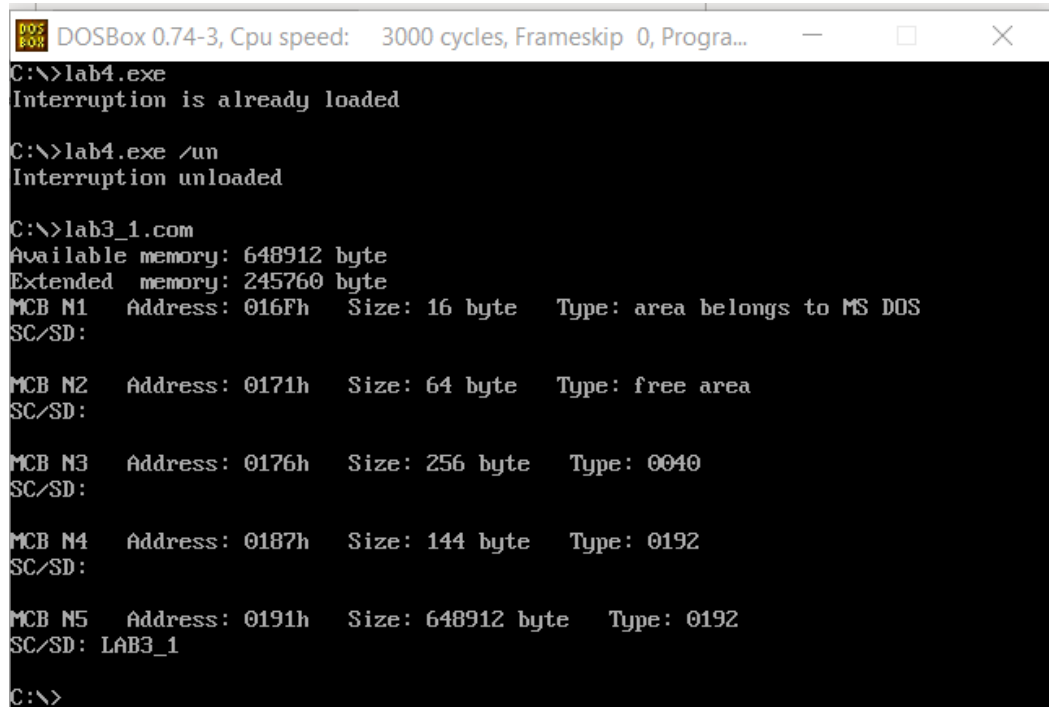
The screenshot shows a DOSBox window titled "DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...". The window contains a green status bar at the top with the text "Call count: 6080". Below this, the text "SC/SD:" is displayed. A memory dump follows, showing several Memory Control Blocks (MCB) with their addresses, sizes, and types. The dump is as follows:

| MCB N | Address | Size | Type |
|-------|---------|-------------|-----------|
| N2 | 0171h | 64 byte | free area |
| N3 | 0176h | 256 byte | 0040 |
| N4 | 0187h | 144 byte | 0192 |
| N5 | 0191h | 736 byte | 0192 |
| N6 | 01C0h | 144 byte | 01CB |
| N7 | 01CAh | 648000 byte | 01CB |

Below the memory dump, the text "SC/SD: LAB4" is displayed. The command prompt shows "C:\>lab4.exe" and the message "Interruption is already loaded". The prompt "C:\>" is shown at the bottom.

Рисунок 3 - Проверка установленного обработчика

- Программа запущена с ключом выгрузки '/un'. Запущена программа lab3_1.com, чтобы убедиться, что память занятая резидентом освобождена. (Рисунок 4)



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
C:\>lab4.exe
Interruption is already loaded

C:\>lab4.exe /un
Interruption unloaded

C:\>lab3_1.com
Available memory: 648912 byte
Extended memory: 245760 byte
MCB N1 Address: 016Fh Size: 16 byte Type: area belongs to MS DOS
SC/SD:

MCB N2 Address: 0171h Size: 64 byte Type: free area
SC/SD:

MCB N3 Address: 0176h Size: 256 byte Type: 0040
SC/SD:

MCB N4 Address: 0187h Size: 144 byte Type: 0192
SC/SD:

MCB N5 Address: 0191h Size: 648912 byte Type: 0192
SC/SD: LAB3_1

C:\>
```

Рисунок 4 - Проверка выгрузки обработчика

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ ПО ЛАБОРАТОРНОЙ №4

1) Как реализован механизм прерывания от часов?

Аппаратное прерывание происходит примерно раз в 55 мс.

Сначала сохраняется содержимое регистров. После определяется смещение по номеру источника прерывания в таблице векторов. Первые 2 байта записываются в IP, вторые – в CS.

Затем передается управление по адресу CS:IP. Выполняется прерывание и восстанавливается информация прерванного процесса. После происходит возврат управления прерванной программе.

2) Какого типа прерывания использовались в работе?

1. Аппаратные – обеспечивают реакцию процессора на события, происходящие асинхронно по отношению к исполняемому программному коду. (1Ch)
2. Программные – вызываются с помощью команды `int`, для обращения к специальным функциям операционной системы. (21h, 10h).

Выводы.

Была исследована обработка стандартных прерывания, построен обработчик прерываний таймера. Реализована загрузка и выгрузка резидента.

Приложение А.

Исходный код программы lab4.asm:

```
dosseg
.model small
.stack 400h

.data
    mstack dw 100h dup(?)
    int_is_load dw ?
    cmd_line_flag dw 0
    str_is_not_load db "Interruption didn't load",0dh,0ah,'$'
    str_load db "Interruption loaded",0dh,0ah,'$'
    str_unload db "Interruption unloaded",0dh,0ah,'$'
    str_already_loaded db "Interruption is already loaded",0dh,0ah,'$'

.code
jmp m

WRITE_STR proc near
    push ax
    mov ah,9h
    int 21h
    pop ax
    ret
WRITE_STR ENDP

MY_INT PROC FAR
    jmp process
    _code dw 0abcdh
```

```

keep_cs dw 0
keep_ip dw 0
temp_ss dw 0
temp_sp dw 0
PSP_0 dw 0
PSP_1 dw 0
str_count      db      "Call      count:      0000
"

```

process:

```

cli
mov temp_ss,ss
mov temp_sp,sp
mov ax,seg mstack
mov ss,ax
mov ax,offset mstack
add ax,100H
mov sp,ax
sti

push ax
push bx
push cx
push dx

mov ah,3
mov bh,0
int 10h ; в dh-текущая строка, dl-текущая колонка курсора

push dx

```

```

push ds
mov ax,seg MY_INT
mov ds,ax
mov di,offset str_count
add di,16

mov cx,4
for:
    mov bh,[di]
    inc bh
    mov [di],bh
    cmp bh,3ah; если цифра в числе превзошла 9
    jne output
    mov bh,30h;ставим цифру 0
    mov [di],bh
    dec di
    loop for

output:
    push es

    mov ax,ds
    mov es,ax
    mov bx,offset str_count
    mov bp,bx

    mov ah,13h
    mov al,0 ;оставить курсор
    mov bh,0;видео страница

```

```
mov dx,0 ; dh,dl-строка,колонка  
mov cx,80;конец строки  
int 10h;вывод строки по адресу es:bp
```

```
pop es  
pop ds
```

```
pop dx  
mov ah,2  
mov bh,0  
int 10h ;установить позицию dh-текущая строка,dl-колонка
```

```
pop dx  
pop cx  
pop bx  
pop ax
```

```
cli  
mov ax,temp_ss  
mov ss,ax  
mov sp,temp_sp  
sti  
iret
```

```
MY_INT ENDP
```

```
empty_func proc  
empty_func endp
```

```
is_LOAD PROC NEAR
```

mov ah,35h

mov al,1ch

int 21H

mov dx,es:[bx+3]

cmp dx,0abcdh

je isLoad

mov int_is_load,0

jmp endOfIsLoad

isLoad:

mov int_is_load,1

endofisload:

ret

is_LOAD endp

UNLOAD PROC NEAR

call is_load

cmp int_is_load,1

jne metka1

mov ah,35h

mov al,1ch

int 21h;получаем вектор

cli

push ds

mov ax,es:[bx+5]

mov ds,ax

mov dx,es:[bx+7]

```
mov ah,25h
mov al,1ch
int 21h ;восстанавливаем вектор
pop ds
sti
```

```
mov dx,offset str_unload
call write_str
```

```
push es
mov cx,es:[bx+13]
mov es,cx
mov ah,49h
int 21h
pop es
mov cx,es:[bx+15]
mov es,cx
int 21h
```

```
jmp metka2
```

```
metka1:
```

```
mov dx,offset str_is_not_load
call write_str
```

```
metka2:
```

```
ret
```

```
UNLOAD ENDP
```

```
LOAD PROC NEAR
```

```
mov ah,35h
mov al,1ch
```



```
int 21h
mov keep_cs,es
mov keep_ip,bx
```

```
push ds
mov dx,offset MY_INT
mov ax,seg MY_INT
mov DS,AX
mov ah,25h
mov al,1ch
int 21h
pop ds
```

```
ret
LOAD endp
```

```
CHECK_CMD_LINE PROC NEAR
```

```
mov di,82h
mov al,es:[di]
cmp al,'/'
jnz end_of_check
inc di
```

```
mov al,es:[di]
cmp al,'u'
jnz end_of_check
inc di
```

```

    mov al,es:[di]
    cmp al,'n'
    jnz end_of_check

    call UNLOAD
    mov cmd_line_flag,1
    jmp ret_check
end_of_check:
    mov cmd_line_flag,0
ret_check:
    ret
CHECK_CMD_LINE ENDP

```

MAIN PROC FAR

```

m:
    mov bx,02ch
    mov ax,[bx]
    mov psp_0,ds
    mov psp_1,ax

    mov ax,@data
    mov ds,ax

    call CHECK_CMD_LINE
    cmp cmd_line_flag,0
    jne exit

```

```
call is_load
cmp int_is_load,1
je alreadyLoaded
```

```
call load
mov dx,offset str_load
call write_str
```

```
;оставим резидентной в памяти
mov dx,offset empty_func
mov cl,04h
shr dx,cl ;в параграфы
add dx,1bh
mov ah,31h
mov al,00h
int 21h
jmp exit
```

```
alreadyLoaded:
    mov dx,offset str_already_loaded
    call write_str
```

```
exit:
    mov ah,4ch
    int 21h
```

```
MAIN ENDP
```

```
end
```