

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. 9382

Кузьмин Д. И.

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 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.

1) Первым шагом было создание пользовательского обработчика прерываний, который срабатывает на каждый сигнал таймера, суммирует количество срабатываний и выводит на экран.

2) Далее была создана функция, проверяющая, установлено ли пользовательское прерывание и если нет, то устанавливает его, загружая при этом резидентную программу.

3) Проверка прерываний осуществляется с помощью функции 31h прерывания int 21h, а установка – с помощью функции 25h.

4) Далее была релизована проверка параметра /up, отвечающего за выгрузку прерывания. В случае, если параметр указан, при помощи функции 49h прерывания int 21h освобождается память, занимаемая резидентной программой и восстанавливается исходный вектор прерываний.

Результаты работы программы см. в приложении А

Исходный код см. в приложении Б

Контрольные вопросы.

1) Как реализован механизм прерывания от часов?

Перед выполнением прерывания, процессор кладет в стек основной программы текущее содержимое трех регистров процессора: регистра флагов, CS и IP. CS и IP формируют полный адрес возврата в прерванную программу. Далее процессор загружает CS и IP из соответствующего вектора прерываний, осуществляя тем самым переход к обработчику прерываний. Обработчик прерываний после выполнения заканчивается командой возврата из прерывания iret (interrupt return, возврат из прерывания), выполняющей загрузку IP, CS и регистра флагов из стека, тем самым возвращаясь в ту самую точку основной программы, где она была прервана.

Системный таймер вырабатывает сигналы с частотой 18,206 Гц, вызывающие аппаратные прерывания 08h. Их обработчик вызывает int 1ch, который передает управление программе, содержащей единственную команду – iret, то есть бездействующую.

2) Какого типа прерывания использовались в работе?

Программные:

Int 21h – функции DOS

Int 10h – функции стандартного видеосервиса ROM-BIOS.

Int 29h – вывод символа

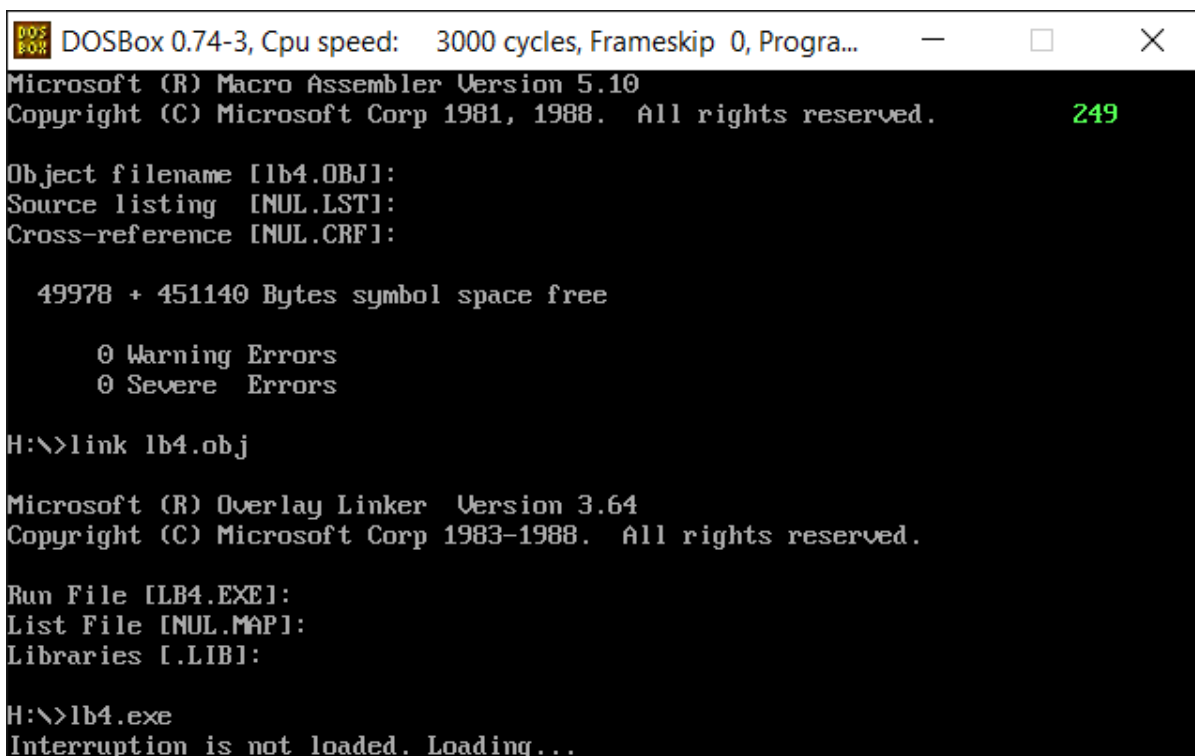
Аппаратные:

Int 08h – сигнал таймера

Выводы.

Были получены навыки разработки обработчика прерываний сигнала таймера.

ПРИЛОЖЕНИЕ А. ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988. All rights reserved. 249

Object filename [lb4.OBJ]:
Source listing [NUL.LST]:
Cross-reference [NUL.CRF]:

49978 + 451140 Bytes symbol space free

0 Warning Errors
0 Severe Errors

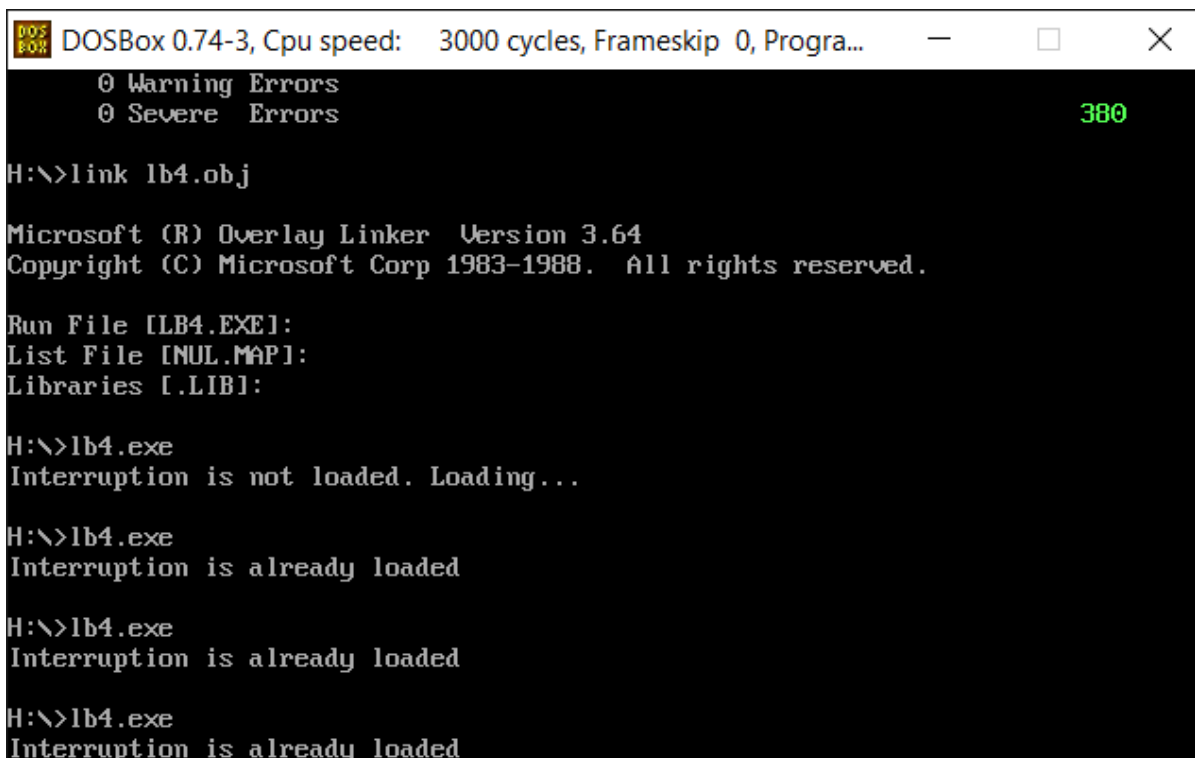
H:\>link lb4.obj

Microsoft (R) Overlay Linker Version 3.64
Copyright (C) Microsoft Corp 1983-1988. All rights reserved.

Run File [LB4.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:

H:\>lb4.exe
Interruption is not loaded. Loading...
```

Рис. 1. - Установка резидентного обработчика прерывания 1ch.



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
0 Warning Errors
0 Severe Errors 380

H:\>link lb4.obj

Microsoft (R) Overlay Linker Version 3.64
Copyright (C) Microsoft Corp 1983-1988. All rights reserved.

Run File [LB4.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:

H:\>lb4.exe
Interruption is not loaded. Loading...

H:\>lb4.exe
Interruption is already loaded

H:\>lb4.exe
Interruption is already loaded

H:\>lb4.exe
Interruption is already loaded
```

Рис. 2 - Проверка уже установленного прерывания.

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
Available memory: 646160
Extended memory: 15360
-----MCB Type:004D
PSP Address/Extra data:0008 Memory size: 16
Data:
-----MCB Type:004D
PSP Address/Extra data:0000 Memory size: 64
Data:
-----MCB Type:004D
PSP Address/Extra data:0040 Memory size: 256
Data:
-----MCB Type:004D
PSP Address/Extra data:0192 Memory size: 144
Data:
-----MCB Type:004D
PSP Address/Extra data:0192 Memory size: 2576
Data:LB4
-----MCB Type:004D
PSP Address/Extra data:023E Memory size: 144
Data:
-----MCB Type:005A
PSP Address/Extra data:023E Memory size: 646160
Data:LB3
```

Рис. 3 - Состояние памяти после загрузки резидентной части

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
Data:LB3

H:\>lb4.exe /un
Interruption unloaded

H:\>lb3.com
Available memory: 648912
Extended memory: 15360
-----MCB Type:004D
PSP Address/Extra data:0008 Memory size: 16
Data:
-----MCB Type:004D
PSP Address/Extra data:0000 Memory size: 64
Data:
-----MCB Type:004D
PSP Address/Extra data:0040 Memory size: 256
Data:
-----MCB Type:004D
PSP Address/Extra data:0192 Memory size: 144
Data:
-----MCB Type:005A
PSP Address/Extra data:0192 Memory size: 648912
Data:LB3
```

Рис. 4 - Выгрузка прерывания и состояние памяти после выгрузки.

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД

Файл lb4.asm

```
ASTACK      SEGMENT      STACK
            DW 256 DUP (?)
ASTACK      ENDS

DATA SEGMENT
resident_set db 'Interruption is already loaded' , 13, 10, '$'
resident_not_set db 'Interruption is not loaded. Loading...', 13, 10, '$'
unload db 'Interruption unloaded', 13, 10, '$'
param db ' /un'
DATA ENDS
CODE SEGMENT
        .386
        ASSUME CS:CODE, DS:DATA, SS:ASTACK
; Процедуры
ROUT PROC FAR

        jmp start
        sum dw 0
        keep_ip dw 0
        keep_cs dw 0
        sign dw 0A35Fh
        psp dw 0
        keep_sp dw 0
        keep_ss dw 0
        keep_ax dw 0
        new_stack db 100 dup (?)
        start:
        mov keep_sp, sp
        mov keep_ss, ss
        mov keep_ax, ax
        mov sp, offset start
        mov ax, seg new_stack
        mov ss, ax

        push ax
        push bx
        push cx
        push dx

        call getCurs
        push dx

        mov dh, 1
        mov dl, 70
        call setCurs

        mov ax, sum
        inc ax
        mov sum, ax

        call EAX_TO_DEC
        pop dx
        call setCurs
```



```

        pop dx
        pop cx
        pop bx
        POP AX

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

ROUT ENDP
WRITE_MSG PROC near
        push ax
        mov AH, 09h
        int 21h
        pop ax
        ret
WRITE_MSG ENDP

EAX_TO_DEC PROC near

        push ebx
        push di
        push dx
        push cx
        push ax
        mov ebx, 0Ah
        mov di, 0
        mov cx, dx
        xor dx, dx

divide:
        xor dx, dx
        div ebx
        push dx
        inc di
        cmp ax, 0
        jne divide

print:
        dec di
        pop dx
        add dl, '0'
        mov al, dl

        push dx
        mov dx, cx
        inc dl
        call setCurs
        call outputAL
        pop dx
        inc cl

        cmp di, 0
        jg print
        pop ax
        pop cx

```

```

        pop dx
        pop di
        pop ebx
        ret
EAX_TO_DEC ENDP
;-----

outputAL proc
        push ax
        push bx
        push cx
        mov ah,09h
        mov bh, 0h
        mov cx,1
        int 10h
        pop cx
        pop bx
        pop ax
        ret
outputAL ENDP
;-----
setCurs proc
        push ax
        push bx
        mov ah,02h
        mov bh,0
        int 10h
        pop bx
        pop ax
        ret

setCurs ENDP
;-----
getCurs proc
        push ax
        push bx
        mov ah,03h
        mov bh,0
        int 10h
        pop bx
        pop ax
        ret

getCurs ENDP
;-----
RESTORE_VECTOR PROC NEAR

        CLI
        PUSH DS
        push dx
        push ax
        MOV DX, es:keep_ip
        MOV AX, es:keep_cs
        MOV DS, AX
        MOV AH, 25H
        MOV AL, 1CH
        INT 21H

```

```

        pop ax
        pop dx
        POP DS
        STI
        RET
RESTORE_VECTOR ENDP
end_rout:
;-----

SET_INTERRUPT PROC NEAR

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

        PUSH DS
        PUSH AX
        PUSH DX
        MOV DX, OFFSET ROUT
        MOV AX, SEG ROUT

        MOV DS, AX
        MOV AH, 25H

        MOV AL, 1CH
        INT 21H

        POP DX
        POP AX
        POP DS

        RET
SET_INTERRUPT ENDP

LOAD_TO_RESIDENT PROC NEAR

        push ax
        push bx
        push dx
        push cx

        mov DX, offset end_rout
        mov cl, 4h
        shr dx, cl
        inc dx
        mov ax, cs
        sub ax, psp
        add dx, ax
        xor ax, ax
        mov ah, 31h
        int 21h

        pop cx

```

```

        pop dx
        pop bx
        pop ax

        ret

LOAD_TO_RESIDENT ENDP
;-----
CHECK_VECTOR PROC NEAR

        PUSH AX
        PUSH BX
        PUSH ES
        PUSH SI

        MOV AH, 35H
        MOV AL, 1Ch
        INT 21H

        mov ax, 0A35Fh ;уникальное значение
        cmp ax,es:sign
        jne setres

        call CHECK_PARAM ;если установлено, переход к проверке
параметра /un
        jmp endthis

setres:      ;если прерывание не установлено
mov dx, offset resident_not_set
call WRITE_MSG
call SET_INTERRUPT
call LOAD_TO_RESIDENT

endthis:
POP SI
POP ES
POP BX
POP AX
RET

CHECK_VECTOR ENDP

UNLOAD_INTERRUPTION PROC NEAR

        call RESTORE_VECTOR

        mov ax,es:psp
        mov es,ax

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

```

```

        mov ah,49h ;резидентная часть
        int 21h
        ret
UNLOAD_INTERRUPTION ENDP

CHECK_PARAM PROC NEAR

        push es
        mov es, psp
        mov cx, 4
        mov di, 81h
        mov si, offset param
        cld
        repe cmpsb
        jne notequal
        pop es
        call UNLOAD_INTERRUPTION
        mov dx, offset unload
        call WRITE_MSG
        jmp ex
notequal:
        pop es
        mov dx, offset resident_set
        call WRITE_MSG
ex:
        ret
CHECK_PARAM ENDP

MAIN      PROC FAR

        mov ax,DATA
        mov ds,ax
        mov psp, es
        call CHECK_VECTOR
        mov ah, 4ch
        int 21h

MAIN ENDP
CODE ENDS
        END MAIN

```