**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. 9382 |  | Рыжих Р.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 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 .ЕХЕ, который выполняет следующие функции:

1) Проверяет, установлено ли пользовательское прерывание с вектором 1Сh.

2) Устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено, и осуществляется выход по функции 4Сh прерывания int 21h.

3) Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Сh 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 4Сh 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 1Сh 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.

**Ход работы**

|  |  |
| --- | --- |
| Процедура | Описание |
| ROUT | Резидентное прерывание, которое загружается в память и выполняет накопление и вывод числа накопленных прерываний на экран. |
| WRITESTRING | Вывод строки на экран |
| IF\_NEED\_UNLOAD | Проверка на наличия флага “/un” |
| IF\_LOADED | Проверка на загрузку пользовательского прерывания в память |
| LOAD\_ROUT | Сохранение первоначального прерывания и загрузка пользовательского прерывания в память |
| UNLOAD\_ROUT | Выгрузка пользовательского прерывания из памяти, а также освобождение памяти и восстановление первоначальных прерываний |
| MAIN | Главная функция |

1. Был написан программный модуль типа .EXE, который требовался на первом шаге задания. Код см. в приложении А
2. Запуск программы и проверка установки резидентного обработчика прерывания 1Сh. Так же по заданию была запущена программа из 3 лабораторной работы для проверки размещения в памяти. (рис.1)

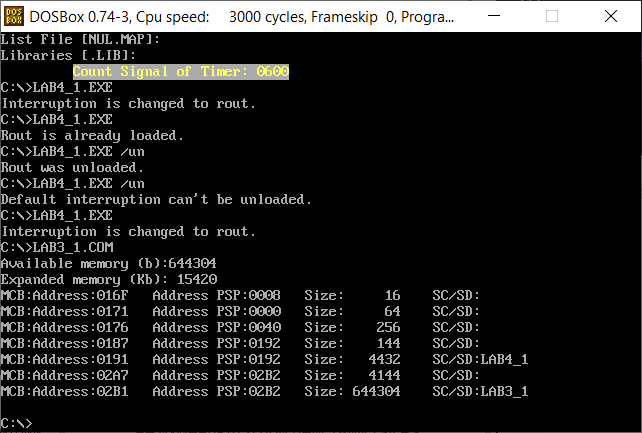


Рис.1 – работа и размещение написанной программы

1. Запустим программу с ключом выгрузки “/un”, чтобы увидеть что обработчик прерывания выгружен и память освободилась. (рис.2)

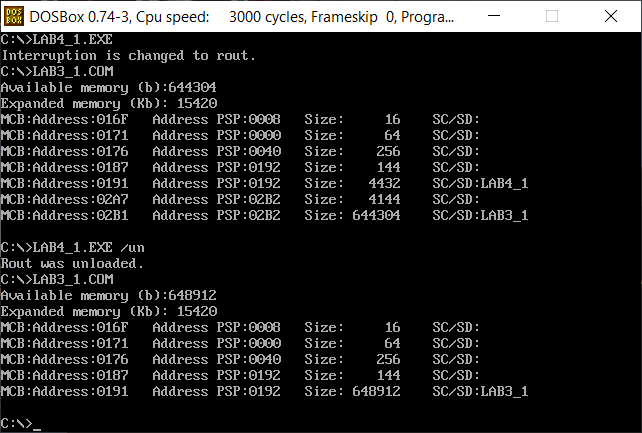


Рис.2 – Запуск с ключом ‘/un’

**Ответы на контрольные вопросы**

1. Как реализован механизм прерывания от часов?

Принимается сигнал прерывания (приходит примерно каждые 54 мс), запоминаются содержимые регистров, по номеру источника прерывания в таблице векторов определяется смещение, запоминается адрес 2 байта в IP и 2 байта в CS. Дальше выполняется прерывание по сохранённому адресу и далее восстанавливается информация прерванного процесса и управление возвращается прерванной программе.

1. Какого типа прерывания использовались в работе?

Int 10h – видео сервис BIOS

Int 21h – сервисы DOS

Пользовательское прерывание с вектором 1ch int 21h

**Выводы.**

В ходе лабораторной работы была исследована обработка стандартных прерываний, а также построен обработчик прерываний сигналов таймера, которые генерируются аппаратурой через определённые интервалы времени. Программа загружает и выгружает резидент, а также производится проверка флагов и загрузки прерывание в память. С помощью rout выполняет накопление и вывод числа накопленных прерываний на экран.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

**Lab4.asm:**

AStack SEGMENT STACK

DW 64 DUP(?)

AStack ENDS

DATA SEGMENT

ROUT\_LOADED db "Rout is already loaded.$"

ROUT\_IS\_LOADING db "Interruption is changed to rout.$"

ROUT\_IS\_NOT\_LOADED db "Default interruption can't be unloaded.$"

ROUT\_IS\_UNLOADED db "Rout was unloaded.$"

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA, SS:AStack

;----------------------------------------

WRITESTRING proc near

push ax

mov ah, 9h

int 21h

pop ax

ret

WRITESTRING endp

;----------------------------------------

start\_rout:

ROUT proc far

jmp START\_PROC

KEEP\_PSP dw 0

KEEP\_IP dw 0

KEEP\_CS dw 0

KEEP\_SS DW 0

KEEP\_SP DW 0

KEEP\_AX DW 0

ROUT\_INDEX dw 00AAh

TIMER\_COUNTER db 'Count Signal of Timer: 0000$'

BStack DW 64 DUP(?)

START\_PROC:

mov KEEP\_SP, sp

mov KEEP\_AX, ax

mov ax, ss

mov KEEP\_SS, ss

mov ax, KEEP\_AX

mov sp, offset START\_PROC

mov ax, seg BStack

mov ss, ax

push bx

push cx

push dx

mov ah,3h

mov bh,0h

int 10h

push dx

push si

push cx

push ds

push ax

push bp

mov ax, SEG TIMER\_COUNTER

mov ds,ax

mov si, offset TIMER\_COUNTER

add si, 22

mov cx, 4

TIMER\_INC:

mov bp, cx

mov ah, [si+bp]

inc ah

cmp ah, 3ah

jl TIMER\_INC\_END

mov ah, 30h

mov [si+bp], ah

loop TIMER\_INC

TIMER\_INC\_END:

mov [si+bp], ah

pop bp

pop ax

pop ds

pop cx

pop si

push es

push bp

mov ax, SEG TIMER\_COUNTER

mov es,ax

mov ax, offset TIMER\_COUNTER

mov bp,ax

mov ah,13h

mov al,00h

mov dh,02h

mov dl,09h

mov cx,27

mov bh,0

int 10h

pop bp

pop es

;return cursor

pop dx

mov ah,02h

mov bh,0h

int 10h

pop dx

pop cx

pop bx

mov KEEP\_AX, ax

mov sp, KEEP\_SP

mov ax, KEEP\_SS

mov ss, ax

mov ax, KEEP\_AX

mov al, 20H

out 20H, al

iret

end\_rout:

ROUT endp

;----------------------------------------

IF\_LOADED proc near

push ax

push si

push es

push dx

mov ah,35h

mov al,1ch

int 21h

mov si, offset ROUT\_INDEX

sub si, offset ROUT

mov dx,es:[bx+si]

cmp dx, ROUT\_INDEX

jne end\_if\_loaded

mov ch,1h

end\_if\_loaded:

pop dx

pop es

pop si

pop ax

ret

IF\_LOADED ENDP

;----------------------------------------

IF\_NEED\_UNLOAD proc near

push ax

push es

mov al,es:[81h+1]

cmp al,'/'

jne end\_if\_need\_unload

mov al,es:[81h+2]

cmp al,'u'

jne end\_if\_need\_unload

mov al,es:[81h+3]

cmp al,'n'

jne end\_if\_need\_unload

mov cl,1h

end\_if\_need\_unload:

pop es

pop ax

ret

IF\_NEED\_UNLOAD endp

;----------------------------------------

UNLOAD\_ROUT PROC near

push ax

push si

cli

push ds

mov ah,35h

mov al,1ch

int 21h

mov si,offset KEEP\_IP

sub si,offset ROUT

mov dx,es:[bx+si]

mov ax,es:[bx+si+2]

mov ds,ax

mov ah,25h

mov al,1ch

int 21h

pop ds

mov ax,es:[bx+si-2]

mov es,ax

push es

mov ax,es:[2ch]

mov es,ax

mov ah,49h

int 21h

pop es

mov ah,49h

int 21h

sti

pop si

pop ax

ret

UNLOAD\_ROUT endp

;----------------------------------------

LOAD\_ROUT PROC near

push ax

push dx

mov KEEP\_PSP, es

mov ah,35h

mov al,1ch

int 21h

mov KEEP\_IP, bx

mov KEEP\_CS, es

push ds

lea dx, ROUT

mov ax, SEG ROUT

mov ds,ax

mov ah,25h

mov al,1ch

int 21h

pop ds

lea dx, end\_rout

mov cl,4h

shr dx,cl

inc dx

add dx,100h

xor ax, ax

mov ah,31h

int 21h

pop dx

pop ax

ret

LOAD\_ROUT endp

;----------------------------------------

MAIN proc far

push DS

push AX

mov AX,DATA

mov DS,AX

call IF\_NEED\_UNLOAD

cmp cl, 1h

je need\_unload

call IF\_LOADED

cmp ch, 1h

je print\_rout\_is\_already\_set

mov dx, offset ROUT\_IS\_LOADING

call WRITESTRING

call LOAD\_ROUT

jmp exit

need\_unload:

call IF\_LOADED

cmp ch, 1h

jne print\_rout\_cant\_be\_unloaded

call UNLOAD\_ROUT

mov dx, offset ROUT\_IS\_UNLOADED

call WRITESTRING

jmp exit

print\_rout\_cant\_be\_unloaded:

mov dx, offset ROUT\_IS\_NOT\_LOADED

call WRITESTRING

jmp exit

print\_rout\_is\_already\_set:

mov dx, offset ROUT\_LOADED

call WRITESTRING

jmp exit

exit:

mov ah, 4ch

int 21h

MAIN endp

CODE ends

END Main