**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. 9381 |  | Прибылов Н.А. |
| Преподаватель |  | Ефремов М.А. |

Санкт-Петербург

2021

## 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 и выполняет соответствующие действия.

В данной лабораторной работе 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. Был написан и отлажен программный модуль типа .EXE, который выполняет следующие функции:

• Проверяет, установлено ли пользовательское прерывание с вектором 1Ch.

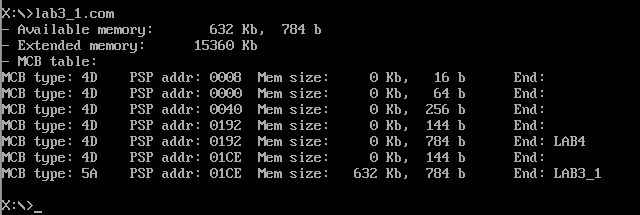
• Устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено, и осуществляется выход о функции 4Ch прерывания int 21h.

• Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h.

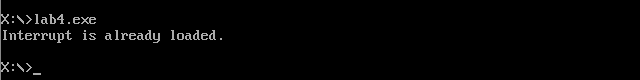
• Выгрузка прерывания о соответствующему значению параметра в командной строке /un. Выгрузка прерывания состоит в восстановлении стандартного вектора прерываний и освобождении памяти, занимаемой резидентом. Затем осуществляется выход по функции 4Ch прерывания int 21h.

2. Далее была запущена отлаженная программа, проверено, что резидентный обработчик прерывания 1Ch установлен. Сделано так, чтобы работа прерывания отображалась на экране, а также проверено размещение прерывания в памяти. Для этого была запущена программа лабораторной работы №3, которая отображает карту памяти в виде списка блоков MCB.

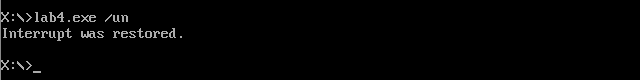
Программа запущена

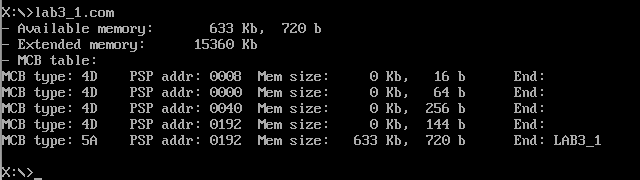
Программа из лаб. работы №3 запущена

3. Была запущена отлаженная программа еще раз и проверено, что она определяет установленный обработчик прерываний.

Программа запущена повторно

4. Была запущена отлаженная программа с ключом выгрузки и проверено, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а занятая резидентом память освобождена. Для этого также была запущена программа лабораторной работы №3.

Программа запущена с ключом /un

Программа из лаб. работы №3 запущена

**Описание функций и переменных.**

**Переменные:**

|  |  |
| --- | --- |
| Название | Описание |
| PSP\_ADDRESS\_1 | Хранит старое значение ES до того, как программа была оставлена резидентной в памяти |
| KEEP\_CS | Хранит сегмент прерывания |
| KEEP\_IP | Хранит смещение прерывания |
| NEW\_INTERRUPT\_SET | Хранит количество вызванных прерываний |
| INT\_COUNT | Строка "Interrupts call count: 0000" |
| STRING\_INT\_NOT\_LOADED | Строка "Interrupt was not loaded." |
| STRING\_INT\_RESTORED | Строка "Interrupt was restored." |
| STRING\_INT\_ALREADY\_LOADED | Строка "Interrupt is already loaded." |
| STRING\_INT\_IS\_LOADING | Строка "Interrupt is loading now." |

**Функции:**

|  |  |
| --- | --- |
| NEW\_INTERRUPT | Собственный обработчик прерывания. Выводит количество прерываний, которые были вызваны. |
| IS\_INTERRUPT\_SET | Проверяет, установлен ли вектор прерывания |
| CHECK\_COMMAND\_PROMT | Проверяет параметр un |
| LOAD\_INTERRUPT | Устанавливает новые обработчики прерывания |
| UNLOAD\_INTERRUPT | Восстанавливает сохранённые заранее обработчики прерываний и выгружает резидентную программу |
| PRINT\_STRING | Печатает строку на экран |

**Контрольные вопросы.**

**1. Как реализован механизм прерывания от часов?**

Принимается сигнал прерывания (примерно каждые 54 мс), запоминаются значения регистров, по номеру источника прерывания в таблице векторов определяется смещение, сохраняются адреса 2 байта в IP и 2 байта в CS. Далее выполняется прерывание по сохранённому адресу и восстанавливается информация прерванного процесса, управление возвращается прерванной программе.

**2. Какого типа прерывания использовались в работе?**

Аппаратные прерывания, прерывания DOS (21h), прерывания BIOS (10h).

## Выводы.

Был построен обработчик прерывания от сигналов таймера. Изучены дополнительные функции работы с памятью: установка программы-резидента и его выгрузка из памяти.

# Приложение А Исходный код программы

Название файла: lab4.asm

CODE SEGMENT

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

NEW\_INTERRUPT PROC FAR

jmp START\_FUNC

PSP\_ADDRESS\_0 dw 0

PSP\_ADDRESS\_1 dw 0

KEEP\_CS dw 0

KEEP\_IP dw 0

NEW\_INTERRUPT\_SET dw 0FEDCh

INT\_COUNT db 'Interrupts call count: 0000 $'

KEEP\_SS dw ?

KEEP\_SP dw ?

KEEP\_AX dw ?

INT\_STACK dw 64 dup (?)

END\_INT\_STACK dw ?

START\_FUNC:

mov KEEP\_SS, ss

mov KEEP\_SP, sp

mov KEEP\_AX, ax

mov ax, cs

mov ss, ax

mov sp, offset END\_INT\_STACK

push ax

push bx

push cx

push dx

mov ah, 03h

mov bh, 00h

int 10h

push dx

mov ah, 02h

mov bh, 00h

mov dx, 0220h

int 10h

push si

push cx

push ds

mov ax, SEG INT\_COUNT

mov ds, ax

mov si, offset INT\_COUNT

add si, 1Ah

mov ah,[si]

inc ah

mov [si], ah

cmp ah, 3Ah

jne END\_CALC

mov ah, 30h

mov [si], ah

mov bh, [si - 1]

inc bh

mov [si - 1], bh

cmp bh, 3Ah

jne END\_CALC

mov bh, 30h

mov [si - 1], bh

mov ch, [si - 2]

inc ch

mov [si - 2], ch

cmp ch, 3Ah

jne END\_CALC

mov ch, 30h

mov [si - 2], ch

mov dh, [si - 3]

inc dh

mov [si - 3], dh

cmp dh, 3Ah

jne END\_CALC

mov dh, 30h

mov [si - 3],dh

END\_CALC:

pop ds

pop cx

pop si

push es

push bp

mov ax, SEG INT\_COUNT

mov es, ax

mov ax, offset INT\_COUNT

mov bp, ax

mov ah, 13h

mov al, 00h

mov cx, 1Dh

mov bh, 0

int 10h

pop bp

pop es

pop dx

mov ah, 02h

mov bh, 0h

int 10h

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

NEW\_INTERRUPT ENDP

NEED\_MEM\_AREA PROC

NEED\_MEM\_AREA ENDP

IS\_INTERRUPT\_SET PROC NEAR

push bx

push dx

push es

mov ah, 35h

mov al, 1Ch

int 21h

mov dx, es:[bx + 11]

cmp dx, 0FEDCh

je INT\_IS\_SET

mov al, 00h

jmp POP\_REG

INT\_IS\_SET:

mov al, 01h

jmp POP\_REG

POP\_REG:

pop es

pop dx

pop bx

ret

IS\_INTERRUPT\_SET ENDP

CHECK\_COMMAND\_PROMT PROC NEAR

push es

mov ax, PSP\_ADDRESS\_0

mov es, ax

mov bx, 0082h

mov al, es:[bx]

inc bx

cmp al, '/'

jne NULL\_CMD

mov al, es:[bx]

inc bx

cmp al, 'u'

jne NULL\_CMD

mov al, es:[bx]

inc bx

cmp al, 'n'

jne NULL\_CMD

mov al, 0001h

NULL\_CMD:

pop es

ret

CHECK\_COMMAND\_PROMT 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

push ds

mov dx, offset NEW\_INTERRUPT

mov ax, seg NEW\_INTERRUPT

mov ds, ax

mov ah, 25h

mov al, 1Ch

int 21h

pop ds

mov dx, offset STRING\_INT\_IS\_LOADING

call PRINT\_STRING

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

cli

push ds

mov dx, es:[bx + 9]

mov ax, es:[bx + 7]

mov ds, ax

mov ah, 25h

mov al, 1Ch

int 21h

pop ds

sti

mov dx, offset STRING\_INT\_RESTORED

call PRINT\_STRING

push es

mov cx, es:[bx + 3]

mov es, cx

mov ah, 49h

int 21h

pop es

mov cx, es:[bx + 5]

mov es, cx

int 21h

pop es

pop dx

pop bx

pop ax

ret

UNLOAD\_INTERRUPT ENDP

PRINT\_STRING PROC NEAR

push ax

mov ah, 09h

int 21h

pop ax

ret

PRINT\_STRING ENDP

MAIN\_PROGRAM PROC FAR

mov bx, 02Ch

mov ax, [bx]

mov PSP\_ADDRESS\_1, ax

mov PSP\_ADDRESS\_0, ds

sub ax, ax

xor bx, bx

mov ax, DATA

mov ds, ax

call CHECK\_COMMAND\_PROMT

cmp al, 01h

je UNLOAD\_START

call IS\_INTERRUPT\_SET

cmp al, 01h

jne INTERRUPT\_IS\_NOT\_LOADED

mov dx, offset STRING\_INT\_ALREADY\_LOADED

call PRINT\_STRING

jmp EXIT\_PROGRAM

mov ah,4Ch

int 21h

INTERRUPT\_IS\_NOT\_LOADED:

call LOAD\_INTERRUPT

mov dx, offset NEED\_MEM\_AREA

mov cl, 04h

shr dx, cl

add dx, 1Bh

mov ax, 3100h

int 21h

UNLOAD\_START:

call IS\_INTERRUPT\_SET

cmp al, 00h

je INT\_IS\_NOT\_SET

call UNLOAD\_INTERRUPT

jmp EXIT\_PROGRAM

INT\_IS\_NOT\_SET:

mov dx, offset STRING\_INT\_NOT\_LOADED

call PRINT\_STRING

jmp EXIT\_PROGRAM

EXIT\_PROGRAM:

mov ah, 4Ch

int 21h

MAIN\_PROGRAM ENDP

CODE ENDS

STACK SEGMENT STACK

db 64 DUP(?)

STACK ENDS

DATA SEGMENT

STRING\_INT\_NOT\_LOADED db "Interrupt was not loaded.", 0dh, 0ah, '$'

STRING\_INT\_RESTORED db "Interrupt was restored.", 0dh, 0ah, '$'

STRING\_INT\_ALREADY\_LOADED db "Interrupt is already loaded.", 0dh, 0ah, '$'

STRING\_INT\_IS\_LOADING db "Interrupt is loading now.", 0dh, 0ah, '$'

DATA ENDS

END MAIN\_PROGRAM