

**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

\_\_\_\_\_

Дерюгин Д.А.

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.

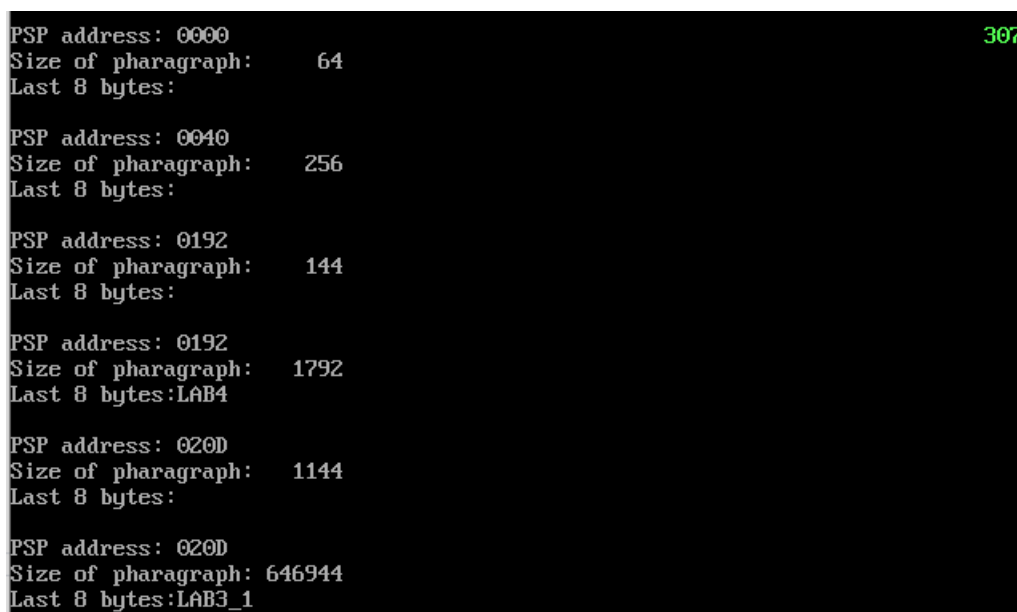
**Шаг 3.** Запустите отлаженную программу еще раз и убедитесь, что программа определяет установленный обработчик прерываний. Полученные результаты поместите в отчет.

**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.

### Ход работы

- 1) Написана программа lab4.exe, которая создает новое прерывание
- 2) Запущена программа из лр 3 для проверки выделения памяти для программы из лр 4.



```
PSP address: 0000
Size of paragraph: 64
Last 8 bytes:

PSP address: 0040
Size of paragraph: 256
Last 8 bytes:

PSP address: 0192
Size of paragraph: 144
Last 8 bytes:

PSP address: 0192
Size of paragraph: 1792
Last 8 bytes: LAB4

PSP address: 020D
Size of paragraph: 1144
Last 8 bytes:

PSP address: 020D
Size of paragraph: 646944
Last 8 bytes: LAB3_1
```

Рис.1 – Результат программы.

- 3) Программа из ЛР 3 запущена еще раз, дополнительной памяти она не заняла.

```

PSP address: 0000
Size of paragraph: 64
Last 8 bytes:

PSP address: 0040
Size of paragraph: 256
Last 8 bytes:

PSP address: 0192
Size of paragraph: 144
Last 8 bytes:

PSP address: 0192
Size of paragraph: 1792
Last 8 bytes: LAB4

PSP address: 0200
Size of paragraph: 1144
Last 8 bytes:

PSP address: 0200
Size of paragraph: 646944
Last 8 bytes: LAB3_1
D:\>1

```

Рис.2 – Результат повторного запуска программы

- 4) Программа запущена с ключом выгрузки. Резидентный обработчик оказался выгружен

```

Reset interruption

D:\>lab3_1.com
Available memory: 648912
Extended memory: 246720
PSP address: 0008
Size of paragraph: 16
Last 8 bytes:

PSP address: 0000
Size of paragraph: 64
Last 8 bytes:

PSP address: 0040
Size of paragraph: 256
Last 8 bytes:

PSP address: 0192
Size of paragraph: 144
Last 8 bytes:

PSP address: 0192
Size of paragraph: 648912
Last 8 bytes: LAB3_1
D:\>1

```

Рис.3 – Результат запуска программы с ключом выгрузки

## **Ответы на контрольные вопросы**

1) Как реализован механизм прерывания от часов?

Сохраняется содержимое регистров CS и IP, чтобы можно было вернуться обратно после прерывания. Берется адрес прерывания и записывается в эти регистры. Выполняется прерывания по этому адресу, после чего возвращается управление прерванной программе.

2) Какого типа прерывания использовались в работе?

Int 1ch - пользовательское

Int 10h - функция видеосервиса BIOS

Int 21h - Функция DOS

### **Выводы.**

В данной лабораторной работе научились создавать собственный обработчик прерыванияю

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

```
ASTACK SEGMENT STACK
    DB 1000 DUP(?)
ASTACK ENDS

DATA SEGMENT

LOADED DW 0
INTERRUPT_LOADED DB 'INTERRUPTION WAS ALREADY LOADED' , 0DH, 0AH, '$'
UN_LOADED DB '/UN LOADED', 0DH, 0AH, '$'
RESET_INTERRUPT DB 'RESET INTERRUPTION', 0DH, 0AH, '$'
COMPLITE_LOADING DB 'LOADING COMPLETE', 0DH, 0AH, '$'
NOT_INTERRUPT DB 'INTERRUPTION IS NOT LOADED', 0DH, 0AH, '$'
DATA ENDS

CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:ASTACK

ROUT PROC FAR
    JMP START

    KEEP_CS DW 0; для хранения сегмента
    KEEP_IP DW 0; и мещения прерывания
    KEEP_PSP DW 0
    SIGNATURE DW 1234H

    SUMMARY DW 0H
    START:

    PUSH AX
    PUSH CX
    PUSH BX
    PUSH DX

    ;SAVE CURSOR
    MOV AH, 03H
    MOV BH, 0
    INT 10H
    PUSH DX

    ;SET CURSOR
```

```

MOV DH, 1
MOV DL, 75
MOV AH, 02H
MOV BH, 0
INT 10H

MOV AX, SUMMARY
INC AX
MOV SUMMARY, AX

SUB CX, CX
SUB DX, DX

MOV BX, 10
STARTING:

DIV BX
PUSH DX
INC CX
XOR DX, DX
TEST AX, AX
JNE STARTING

LOOPING:
POP DX
ADD DL, '0'
MOV AL, DL

PUSH CX

MOV AH, 03H
MOV BH, 0
INT 10H
INC DL
MOV AH, 02H
MOV BH, 0
INT 10H

MOV AH, 09H
MOV BH, 0
MOV CX, 1

```



```

        INT 10H
    POP CX

    LOOP LOOPING

    POP DX

    MOV AH, 02H
    MOV BH, 0
    INT 10H

    POP DX
    POP BX
    POP CX
    POP AX
    MOV AL, 20H
    OUT 20H, AL
    IRET

ROUT ENDP

PRINT PROC NEAR
    MOV     AH,     09H
    INT     21H

    RET
PRINT ENDP

Is_INTERRUPTION_SET PROC NEAR
    PUSH BX
    PUSH ES
    PUSH SI

    MOV AH, 35H; ФУНКЦИЯ ПОЛУЧЕНИЯ ВЕКТОРА
    MOV AL, 1CH; НОМЕР ВЕКТОРА
    INT 21H

    MOV SI, OFFSET SIGNATURE
    SUB SI, OFFSET ROUT
    CMP ES:[BX+SI], 1234H
    JNE RETURN

```

```

MOV DX, OFFSET INTERRUPT_LOADED
CALL PRINT

MOV LOADED, 1
JMP END_RET

RETURN:
MOV KEEP_IP, BX; ЗАПОМИНАНИЕ СМЕЩЕНИЯ
MOV KEEP_CS, ES; И СЕРМЕНТА
END_RET:
POP SI
POP ES
POP BX
RET

IS_INTERRUPTION_SET ENDP

IS_UN_SET PROC NEAR

MOV AL, ES:[81H + 1]
CMP AL, '/'
JNE ENDING
MOV AL, ES:[81H + 2]
CMP AL, 'U'
JNE ENDING
MOV AL, ES:[81H + 3]
CMP AL, 'N'
JNE ENDING

CMP LOADED, 1
JE INT_AND_UN

MOV DX, OFFSET NOT_INTERRUPTION
CALL PRINT
MOV LOADED, 2
JMP ENDING

INT_AND_UN:
MOV LOADED, 10
MOV DX, OFFSET UN_LOADED

```

```

CALL PRINT

ENDING:

RET

IS_UN_SET ENDP

FREE_MEMORY PROC NEAR

MOV AH, 35H
MOV AL, 1CH
INT 21H
XOR AX, AX

CLI
PUSH DS
MOV DX, ES:KEEP_IP
MOV AX, ES:KEEP_CS
MOV DS, AX
MOV AH, 25H
MOV AL, 1CH
INT 21H; ВОССТАНАВЛИВАЕМ ВЕКТОР
POP DS
STI

MOV AX, ES:KEEP_PSP
MOV ES, AX
PUSH ES
MOV AX, ES:[2CH]; АДРЕС СРЕДЫ
MOV ES, AX

MOV AH, 49H
INT 21H; ОВОБОЖДЕНИЕ СРЕДЫ

POP ES
MOV AH, 49H
INT 21H

MOV DX, OFFSET RESET_INTERRUPT
CALL PRINT

```

RET

FREE\_MEMORY ENDP

SET\_INTERRUPTION PROC NEAR

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\_INTERRUPTION ENDP

SAVE\_MEMORY PROC NEAR

PUSH AX

PUSH BX

PUSH DX

PUSH CX

MOV DX, OFFSET COMPLITE\_LOADING

CALL PRINT

MOV DX, OFFSET LAST

MOV CL, 4H

SHR DX, CL

INC DX

MOV AX, CS

SUB AX, KEEP\_PSP

```

    ADD DX, AX
    XOR AX, AX
    MOV AH, 31H
    INT 21H

    POP CX
    POP DX
    POP BX
    POP AX

RET
SAVE_MEMORY ENDP

MAIN PROC FAR

    MOV AX, DATA
    MOV DS, AX
    MOV KEEP_PSP, ES

    CALL IS_INTERRUPTION_SET
    CALL IS_UN_SET

    CMP LOADED, 10
    JE FREE_MEM
    CMP LOADED, 2
    JE FINISH
    CMP LOADED, 1
    JE FINISH
    CALL SET_INTERRUPTION
    CALL SAVE_MEMORY
    JMP FINISH

FREE_MEM:
    CALL FREE_MEMORY

FINISH:

```

```
        MOV AH, 4CH
        INT 21H
        LAST:

MAIN ENDP

CODE ENDS

END MAIN
```