

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МОЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №5**  
**по дисциплине «Операционные системы»**  
**Тема: Сопряжение стандартного и пользовательского обработчиков**  
**прерываний**

Студентка гр. 9383

\_\_\_\_\_

Чебесова И.Д.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2021

## ПОСТАНОВКА ЗАДАЧИ

### Цель работы.

Исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Пользовательский обработчик прерывания получает управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре. Он обрабатывает скан-код и осуществляет определенные действия, если скан-код совпадает с определенными кодами, которые он должен обрабатывать. Если скан-код не совпадает с этими кодами, то управление передается стандартному прерыванию.

### Задание.

**Шаг 1.** Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет такие же функции, как в программе ЛР 4, а именно:

1) Проверяет, установлено ли пользовательское прерывание с вектором 09h.

2) Если прерывание не установлено то, устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний. Адрес точки входа в стандартный обработчик прерывания находится в теле пользовательского обработчика. 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.

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) Сохранить значения регистров в стеке при входе и восстановить их при выходе.
- 2) При выполнении тела процедуры анализируется скан-код.
- 3) Если этот код совпадает с одним из заданных, то требуемый код записывается в буфер клавиатуры.
- 4) Если этот код не совпадает ни с одним из заданных, то осуществляется передача управления стандартному обработчику прерывания.

**Шаг 2.** Запустите отлаженную программу и убедитесь, что резидентный обработчик прерывания 09h установлен. Работа прерывания проверяется введением различных символов, обрабатываемых установленным обработчиком и стандартным обработчиком.

**Шаг 3.** 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 4.** Запустите отлаженную программу еще раз и убедитесь, что программа определяет установленный обработчик прерываний. Полученные результаты поместите в отчет.

**Шаг 5.** 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 6.** Ответьте на контрольные вопросы.

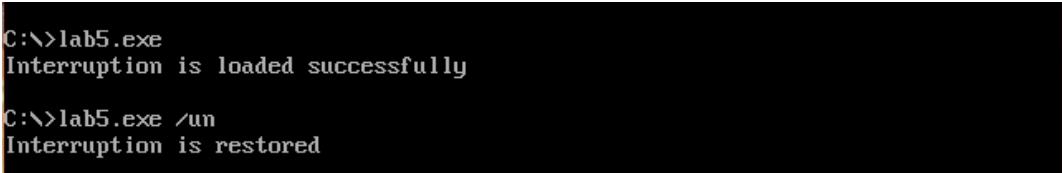
## **РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ ПРОБЛЕМ**

**Шаг 1.** Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет такие же функции, как в программе ЛР 4, а именно...

Был написан и отлажен программный модуль типа .EXE, который выполняет, прописанные в задании функции.

**Шаг 2.** Запустите отлаженную программу и убедитесь, что резидентный обработчик прерывания 09h установлен. Работа прерывания проверяется введением различных символов, обрабатываемых установленным обработчиком и стандартным обработчиком.

Была запущена программа. Резидентный обработчик прерываний был установлен и размещен в памяти.



```
C:\>lab5.exe
Interruption is loaded successfully

C:\>lab5.exe /un
Interruption is restored
```

Рисунок 1 – Демонстрация корректной работы резидентного обработчика прерываний

**Шаг 3.** 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.

Проверено размещение прерывания в памяти.

```

C:\>lab5.exe
Interruption is loaded successfully

C:\>lab3_1.com
Available Memory <Bytes>:644288
Extended Memory <KBytes>:15360

MCB List:
MCB 01 Address: 016F PSP TYPE:  Belong MS DOS      Size: 0001 SC/SD:
MCB 02 Address: 0171 PSP TYPE:  Free PSP           Size: 0004 SC/SD:  DPMI0A
MCB 03 Address: 0176 PSP TYPE:  0040              Size: 0010 SC/SD:
MCB 04 Address: 0187 PSP TYPE:  0192              Size: 0009 SC/SD:
MCB 05 Address: 0191 PSP TYPE:  0192              Size: 0116 SC/SD:  LAB5
MCB 06 Address: 02A8 PSP TYPE:  02B3              Size: 0009 SC/SD:
MCB 07 Address: 02B2 PSP TYPE:  02B3              Size: 9D4C SC/SD:  LAB3_1

```

Рисунок 2 – Демонстрация корректного отображения обработчика прерываний в памяти

**Шаг 4.** Запустите отлаженную программу еще раз и убедитесь, что программа определяет установленный обработчик прерываний. Полученные результаты поместите в отчет.

Программа была повторно запущена, чтобы удостовериться, что программа определяет установленный обработчик прерываний.

```

C:\>lab3_1.com
Available Memory <Bytes>:644288
Extended Memory <KBytes>:15360

MCB List:
MCB 01 Address: 016F PSP TYPE:  Belong MS DOS      Size: 0001 SC/SD:
MCB 02 Address: 0171 PSP TYPE:  Free PSP           Size: 0004 SC/SD:  DPMI0A
MCB 03 Address: 0176 PSP TYPE:  0040              Size: 0010 SC/SD:
MCB 04 Address: 0187 PSP TYPE:  0192              Size: 0009 SC/SD:
MCB 05 Address: 0191 PSP TYPE:  0192              Size: 0116 SC/SD:  LAB5
MCB 06 Address: 02A8 PSP TYPE:  02B3              Size: 0009 SC/SD:
MCB 07 Address: 02B2 PSP TYPE:  02B3              Size: 9D4C SC/SD:  LAB3_1

C:\>lab5.exe
Interruption is already loaded

```

Рисунок 3 – Демонстрация корректного определения установленного обработчика прерывания при повторном запуске программы

**Шаг 5.** 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.

Программа была запущена с ключом выгрузки, чтобы убедиться, что резидентный обработчик прерывания выгружен и память, занятая резидентом освобождена.

```
C:\>lab3_1.com
Available Memory <Bytes>:648912
Extended Memory <KBytes>:15360

MCB List:
MCB @1 Address: 016F PSP TYPE:  Belong MS DOS      Size: 0001 SC/SD:
MCB @2 Address: 0171 PSP TYPE:  Free PSP           Size: 0004 SC/SD:  DPMILOA
MCB @3 Address: 0176 PSP TYPE:  0040              Size: 0010 SC/SD:
MCB @4 Address: 0187 PSP TYPE:  0192              Size: 0009 SC/SD:
MCB @5 Address: 0191 PSP TYPE:  0192              Size: 9E6D SC/SD:  LAB3_1
```

Рисунок 4 – Демонстрация корректной выгрузки резидентного обработчика прерываний.

## ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

**По итогам выполнения работы можно ответить на контрольные вопросы:**

1. *Какого типа прерывания использовались в работе?*

**Ответ:** в работе использовались следующие типы прерываний:

- 1). 09h и 16h – аппаратное прерывание,
- 2). 10h и 21h – программное прерывание.

2. *Чем отличается скан-код от кода ASCII?*

**Ответ:** ASCII код – это код символа в таблице ASCII.

А скан-код – это в своем роде уникальное число, которое однозначно определяет нажатую клавишу, но не ASCII-код.

### **Выводы.**

В ходе лабораторной работы были исследованы возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

**lab5.asm:**

ASTACK SEGMENT STACK

    DW 200 DUP(?)

ASTACK ENDS

DATA SEGMENT

    STR\_INT\_ALREADY\_LOADED DB 'INTERRUPTION IS ALREADY LOADED', 0DH,  
    0AH, '\$'

    STR\_INT\_LOADED\_SUCCESSFULLY DB 'INTERRUPTION IS LOADED  
SUCCESSFULLY', 0DH, 0AH, '\$'

    STR\_INT\_IS\_NOT\_LOADED DB 'INTERRUPTION IS NOT LOADED', 0DH, 0AH,  
    '\$'

    STR\_INT\_RESTORED DB 'INTERRUPTION IS RESTORED', 0DH, 0AH, '\$'  
DATA ENDS

CODE SEGMENT

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

;-----

    PRINT\_MESSAGE PROC NEAR

        PUSH AX

        MOV AH, 9

        INT 21H

        POP AX

        RET

    PRINT\_MESSAGE ENDP

;-----

;-----

    MY\_INT PROC FAR

        JMP INT\_START

        KEY DB 0H

        SHIFT DB 0

        INT\_SIGNATURE DW 7777H

        KEEP\_IP DW 0



```

KEEP_CS DW 0
KEEP_SS DW 0
KEEP_SP DW 0
KEEP_AX DW 0
    PSP_ADDRESS DW ?
INT_STACK DW 64 DUP(?)

```

INT\_START:

```

    MOV KEEP_SP, SP
    MOV KEEP_AX, AX
    MOV AX, SS
    MOV KEEP_SS, AX
    MOV SP, OFFSET INT_START
    MOV AX, SEG INT_STACK
    MOV SS, AX
    MOV AX, KEEP_AX
    PUSH AX
    PUSH CX
    PUSH DX
    MOV KEY, 0H
    MOV SHIFT, 0H
    MOV AX, 40H
    MOV ES, AX
    MOV AX, ES:[17H]
    AND AX, 11B
    CMP AX, 0H
    JE READ_NEW_SYMBOL
    MOV SHIFT, 1H

```

READ\_NEW\_SYMBOL:

```

    IN AL, 60H
    CMP AL, 10H
    JE SYMBOL_Q
    CMP AL, 11H
    JE SYMBOL_W
    MOV KEY, 1H
    JMP INT_END

```

SYMBOL\_Q:

```
MOV AL, 'A'  
JMP CHANGE
```

SYMBOL\_W:

```
MOV AL, 'Z'  
JMP CHANGE
```

CHANGE:

```
PUSH AX  
IN AL, 61H  
MOV AH, AL  
OR AL, 80H  
OUT 61H, AL  
XCHG AH, AL  
OUT 61H, AL  
MOV AL, 20H  
OUT 20H, AL  
POP AX  
CMP SHIFT, 0H  
JE WRITE_KEY  
SUB AL, 20H
```

WRITE\_KEY:

```
MOV AH, 05H  
MOV CL, AL  
MOV CH, 00H  
INT 16H  
OR AL, AL  
JZ INT_END  
MOV AX, 0040H  
MOV ES, AX  
MOV AX, ES:[1AH]  
MOV ES:[1CH], AX  
JMP WRITE_KEY
```

INT\_END:

```
POP DX
```

```

POP CX
POP AX
MOV SP, KEEP_SP
MOV AX, KEEP_SS
MOV SS, AX
MOV AX, KEEP_AX
MOV AL, 20H
OUT 20H, AL
CMP KEY, 1H
JNE INT_IRET
JMP DWORD PTR CS:[KEEP_IP]

```

```

INT_IRET:
    IRET

```

```

MY_INT ENDP

```

```

INT_LAST:

```

```

;-----

```

```

;-----

```

```

CHECK_UN PROC NEAR
    PUSH AX
    PUSH BP
    MOV CL, 0H
    MOV BP, 81H
    MOV AL,ES:[BP + 1]
    CMP AL, '/'
    JNE FINAL
    MOV AL,ES:[BP + 2]
    CMP AL, 'U'
    JNE FINAL
    MOV AL,ES:[BP + 3]
    CMP AL, 'N'
    JNE FINAL
    MOV CL, 1H

```

```

FINAL:
    POP BP

```

```

        POP AX
        RET
CHECK_UN ENDP

```

```

IS_LOAD PROC NEAR
    PUSH AX
    PUSH DX
    PUSH ES
    PUSH SI
    MOV CL, 0H
    MOV AH, 35H
    MOV AL, 09H
    INT 21H
    MOV SI, OFFSET INT_SIGNATURE
    SUB SI, OFFSET MY_INT
    MOV DX, ES:[BX + SI]
    CMP DX, INT_SIGNATURE
    JNE FINISH_CHECK_LOAD
    MOV CL, 1H

```

```

FINISH_CHECK_LOAD:
    POP SI
    POP ES
    POP DX
    POP AX
    RET
IS_LOAD ENDP

```

```

;-----

```

```

;-----

```

```

LOAD_INT PROC NEAR
    PUSH AX
    PUSH CX
    PUSH DX
    CALL IS_LOAD
    CMP CL, 1H
    JE ALREADY_LOADED
    MOV PSP_ADDRESS, ES

```

```

        MOV AH, 35H
MOV AL, 09H
INT 21H
        MOV KEEP_CS, ES
        MOV KEEP_IP, BX
PUSH ES
PUSH BX
        PUSH DS
        LEA DX, MY_INT
        MOV AX, SEG MY_INT
        MOV DS, AX
        MOV AH, 25H
        MOV AL, 09H
        INT 21H
        POP DS
POP BX
POP ES
MOV DX, OFFSET STR_INT_LOADED_SUCCESSFULLY
        CALL PRINT_MESSAGE
        LEA DX, INT_LAST
        MOV CL, 4H
        SHR DX, CL
        INC DX
        ADD DX, 100H
        XOR AX, AX
        MOV AH, 31H
        INT 21H
JMP FINISH_LOAD

```

ALREADY\_LOADED:

```

        MOV DX, OFFSET STR_INT_ALREADY_LOADED
        CALL PRINT_MESSAGE

```

FINISH\_LOAD:

```

        POP DX
POP CX
        POP AX
RET

```

LOAD\_INT ENDP

UNLOAD\_INT PROC NEAR

```
    PUSH AX
    PUSH SI
    CALL IS_LOAD
    CMP CL, 1H
    JNE NOT_LOADED
CLI
PUSH DS
PUSH ES
MOV AH, 35H
MOV AL, 09H
INT 21H
MOV SI, OFFSET KEEP_IP
    SUB SI, OFFSET MY_INT
    MOV DX, ES:[BX + SI]
    MOV AX, ES:[BX + SI + 2]
    MOV DS, AX
MOV AH, 25H
MOV AL, 09H
INT 21H
MOV AX, ES:[BX + SI + 4]
    MOV ES, AX
    PUSH ES
        MOV AX, ES:[2CH]
        MOV ES, AX
        MOV AH, 49H
        INT 21H
    POP ES
    MOV AH, 49H
    INT 21H
POP ES
POP DS
STI
MOV DX, OFFSET STR_INT_RESTORED
    CALL PRINT_MESSAGE
JMP FINISH_UNLOAD
```

```

NOT_LOADED:
    MOV DX, OFFSET STR_INT_IS_NOT_LOADED
    CALL PRINT_MESSAGE

FINISH_UNLOAD:
    POP SI
    POP AX
    RET
UNLOAD_INT ENDP
;-----

;-----

MAIN PROC FAR
    MOV AX, DATA
    MOV DS, AX
    CALL CHECK_UN
    CMP CL, 0H
    JNE UN_UNLOAD
    CALL LOAD_INT
    JMP FINISH_MAIN

UN_UNLOAD:
    CALL UNLOAD_INT

FINISH_MAIN:
    XOR AL, AL
    MOV AH, 4CH
    INT 21H
MAIN ENDP

CODE ENDS
END MAIN

```