

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

ОТЧЕТ
по лабораторной работе №6
по дисциплине «Операционные системы»
Тема: СОПРЯЖЕНИЕ СТАНДАРТНОГО И ПОЛЬЗОВАТЕЛЬСКОГО
ОБРАБОТЧИКОВ ПРЕРЫВАНИЙ.

Студент гр. 9383

Хотяков Е.П.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Постановка задачи.

Цель работы.

Исследовать возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Написать пользовательский обработчик прерывания, который получает управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре и обрабатывает скан-код, осуществляя 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.

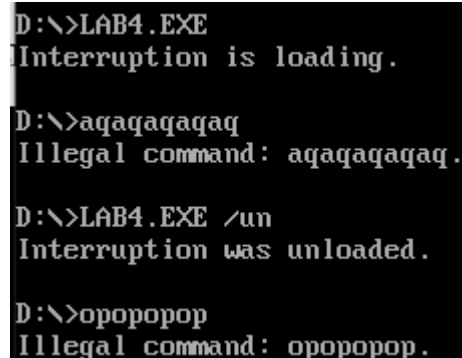
Исходный код.

Исходный код представлен в приложении А.

Результаты исследования проблем.

Шаг 1. Был написан и отлажен программный модуль типа .EXE, который выполняет, выполняет требуемый функционал. Пользовательский обработчик прерывания заменяет символы “o” на “a” и “p” на “q” .

Шаг 2. Программа была отлажена и запущена.



```
D:\>LAB4.EXE
Interruption is loading.

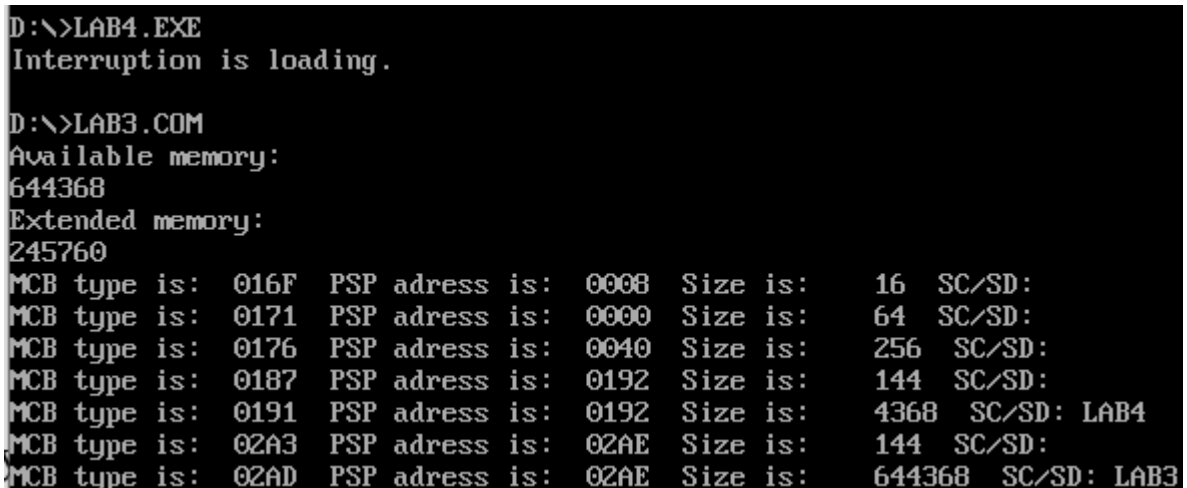
D:\>aqaaqaaqaaq
Illegal command: aqaaqaaqaaq.

D:\>LAB4.EXE /un
Interruption was unloaded.

D:\>opopopopop
Illegal command: opopopopop.
```

Рисунок 1 – Демонстрация корректной работы резидентного обработчика прерываний

Шаг 3. Проверка корректного размещения прерывания в памяти.



```
D:\>LAB4.EXE
Interruption is loading.

D:\>LAB3.COM
Available memory:
644368
Extended memory:
245760
MCB type is: 016F PSP adress is: 0008 Size is: 16 SC/SD:
MCB type is: 0171 PSP adress is: 0000 Size is: 64 SC/SD:
MCB type is: 0176 PSP adress is: 0040 Size is: 256 SC/SD:
MCB type is: 0187 PSP adress is: 0192 Size is: 144 SC/SD:
MCB type is: 0191 PSP adress is: 0192 Size is: 4368 SC/SD: LAB4
MCB type is: 02A3 PSP adress is: 02AE Size is: 144 SC/SD:
MCB type is: 02AD PSP adress is: 02AE Size is: 644368 SC/SD: LAB3
```

Рисунок 2 – Демонстрация корректного определения установленного обработчика прерывания.

Шаг 4. Программа была повторно запущена с ключом, чтобы убедиться, что программа определяет резидентный обработчик прерывания.

```
D:\>LAB4.EXE
Interruption is loading.

D:\>LAB3.COM
Available memory:
644368
Extended memory:
245760
MCB type is: 016F PSP address is: 0008 Size is: 16 SC/SD:
MCB type is: 0171 PSP address is: 0000 Size is: 64 SC/SD:
MCB type is: 0176 PSP address is: 0040 Size is: 256 SC/SD:
MCB type is: 0187 PSP address is: 0192 Size is: 144 SC/SD:
MCB type is: 0191 PSP address is: 0192 Size is: 4368 SC/SD: LAB4
MCB type is: 02A3 PSP address is: 02AE Size is: 144 SC/SD:
MCB type is: 02AD PSP address is: 02AE Size is: 644368 SC/SD: LAB3

D:\>LAB4.EXE
Interruption has already loaded.
```

Рисунок 3 – Демонстрация корректного определения обработчика прерывания при повторном запуске.

Шаг 4. Была запущена отлаженная программа с ключом выгрузки. Резидентный обработчик был выгружен. Память была успешно освобождена.

```
D:\>LAB4.EXE /un
Interruption was unloaded.

D:\>LAB3.COM
Available memory:
648912
Extended memory:
245760
MCB type is: 016F PSP address is: 0008 Size is: 16 SC/SD:
MCB type is: 0171 PSP address is: 0000 Size is: 64 SC/SD:
MCB type is: 0176 PSP address is: 0040 Size is: 256 SC/SD:
MCB type is: 0187 PSP address is: 0192 Size is: 144 SC/SD:
MCB type is: 0191 PSP address is: 0192 Size is: 648912 SC/SD: LAB3
```

По итогам выполнения работы можно ответить на контрольные вопросы:

1. Какого типа прерывания использовались в работе?

Было использовано аппаратное прерывание 09h и 16h, а также пользовательское прерывание 21h и 10h.

2. Чем отличается скан-код от кода ASCII

Скан-код – это код клавиши клавиатуры, который преобразуется обработчиком в код символа. ASCII код – это код самого символа в таблице.

Выводы.

В результате работы была исследована возможность встраивать пользовательский обработчик прерывания в стандартный обработчик с использованием ввода с клавиатуры.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

lab4.asm:

```
MY_STACK SEGMENT STACK
    DW 64 DUP(?)
MY_STACK ENDS

DATA SEGMENT
    INT_NOT_LOAD DB 'INTERRUPTION DID NOT LOAD.', 0DH, 0AH, '$'
    INT_IS_UNLOADED DB 'INTERRUPTION WAS UNLOADED.', 0DH, 0AH, '$'
    INT_LOADED DB 'INTERRUPTION HAS ALREADY LOADED.', 0DH, 0AH, '$'
    INT_IS_LOADING DB 'INTERRUPTION IS LOADING.', 0DH, 0AH, '$'
DATA ENDS

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

WRITE_STRING PROC NEAR
    PUSH AX
    MOV AH, 9H
    INT 21H
    POP AX
    RET
WRITE_STRING ENDP

START:
ROUT PROC FAR
    JMP START_PROC
    SAVED_PSP DW 0
    SAVED_IP DW 0
    SAVED_CS DW 0
    SAVED_SS DW 0
    SAVED_SP DW 0
    SAVED_AX DW 0
    INDEX DW 1337H
    INTERRUPTION_STACK DW 64 DUP(?)
START_PROC:
```

```

MOV SAVED_SP, SP
MOV SAVED_AX, AX
MOV SAVED_SS, SS

MOV AX, SEG INTERRUPTION_STACK
MOV SS, AX
MOV AX, OFFSET START_PROC
MOV SP, AX

MOV AX, SAVED_AX

PUSH BX
    PUSH CX
    PUSH DX
PUSH SI
PUSH CX
PUSH DS
PUSH AX

IN AL, 60H
CMP AL, 18H
JE KEY_O
CMP AL, 19H
JE KEY_P
STANDART:
    CALL DWORD PTR CS:[SAVED_IP] ;ИДЕМ К СТАНДАРТНОМУ ОБРАБОТЧИКУ
    JMP END_ROUT
KEY_O:
    MOV AL, 'A'
    JMP DO_REQ
KEY_P:
    MOV AL, 'Q'

DO_REQ:
    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

```

READ_SYMBOL:

```

MOV AH, 05H
MOV CL, AL
MOV CH, 00H
INT 16H
OR AL, AL
JZ END_ROUT
MOV AX, 40H
MOV ES, AX
MOV AX, ES:[1AH]
MOV ES:[1CH], AX
JMP READ_SYMBOL

```

END_ROUT:

```

POP DS
POP ES
POP SI
POP DX
POP CX
POP BX
POP AX

MOV SP, SAVED_SP
MOV AX, SAVED_SS
MOV SS, AX
MOV AX, SAVED_AX
MOV AL, 20H
OUT 20H, AL
IRET

```

ROUT 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

LOAD_ROUT PROC NEAR
    PUSH AX
    PUSH DX

    MOV SAVED_PSP, ES

    MOV AH,35H
    MOV AL, 09H
    INT 21H
    MOV SAVED_IP, BX
    MOV SAVED_CS, ES

    PUSH DS
    LEA DX, ROUT

```

```

MOV AX, SEG ROUT
MOV DS,AX
MOV AH,25H
MOV AL,09H
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

UNLOAD_ROUT PROC NEAR
PUSH AX
PUSH SI

CLI
PUSH DS
MOV AH,35H
MOV AL,09H
INT 21H

MOV SI,OFFSET SAVED_IP
SUB SI,OFFSET ROUT
MOV DX,ES:[BX+SI]
MOV AX,ES:[BX+SI+2]
MOV DS,AX
MOV AH,25H
MOV AL,09H

```

```

    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

IF_LOADED PROC NEAR
    PUSH AX
    PUSH SI

    PUSH ES
    PUSH DX

    MOV AH,35H
    MOV AL,09H
    INT 21H

    MOV SI, OFFSET INDEX
    SUB SI, OFFSET ROUT
    MOV DX,ES:[BX+SI]
    CMP DX, INDEX
    JNE END_IF_LOADED

```

```

        MOV CH,1H

END_IF_LOADED:
        POP DX
        POP ES
        POP SI
        POP AX
        RET
IF_LOADED 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 INT_IS_LOADING
        CALL WRITE_STRING
        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 INT_IS_UNLOADED
        CALL WRITE_STRING
        JMP EXIT

PRINT_ROUT_CANT_BE_UNLOADED:

```

```
    MOV DX, OFFSET INT_NOT_LOAD
    CALL WRITE_STRING
    JMP EXIT
PRINT_ROUT_IS_ALREADY_SET:
    MOV DX, OFFSET INT_LOADED
    CALL WRITE_STRING
    JMP EXIT

EXIT:
    MOV AH, 4CH
    INT 21H
MAIN ENDP
CODE ENDS
END MAIN
```