**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №5**

**по дисциплине «Операционные системы»**

Тема: **Сопряжение стандартного и пользовательского обработчиков прерываний**

|  |  |  |
| --- | --- | --- |
| Студент гр.0382 |  | Литягин С.М. |
| Преподаватель |  | Ефремов М.А. |

Санкт-Петербург

2022

## 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.

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

- 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 по функции 4Ch прерывания int 21h;

- 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. Длина кода сигнатуры должна быть достаточной, чтобы сделать случайное совпадение маловероятным.

Программа должна содержать код устанавливаемого прерывания в виде удаленной процедуры. Этот код и будет работать после установки при возникновении прерывания. Он должен выполнять следующие функции:

- сохранить значения регистров в стеке при входе и восстановить их при выходе;

- при выполнении тела процедуры анализируется скан-код;

- если этот код совпадает с одним из заданных, то требуемый код записывается в буфер клавиатуры;

- если этот код не совпадает ни с одним из заданных, то осуществляется передача управления стандартному обработчику прерывания.

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

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.

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

5. Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, т.е. сообщения на экран не выводятся, а память, занятая резидентом, освобождена. Для этого также следует запустить программу ЛР3. Полученные результаты поместите в отчет.

6. Ответьте на контрольные вопросы.

## Выполнение работы.

1. Был изменен .EXE модуль ЛР 4 для новых условий, а именно: изменена процедура прерывания MY\_INT, изменяется вектор прерывания 09h, а не 1Ch.

2. После запуска модуля были нажаты следующие клавиши: left ALT, e, f, z, space, k. Результат представлен на рисунке 1:

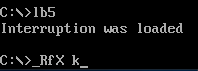


Рисунок 1 – результат первого шага

3. Для выполнения данного шага воспользуемся программой lb3\_1 из предыдущей лабораторной работы. Результат представлен на рисунке 2. Как можно заметить, обработчик прерывания действительно загружен в память.

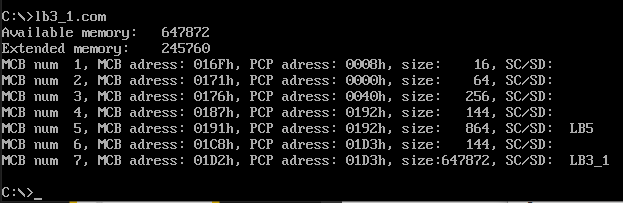


Рисунок 2 – Результаты второго шага

4. При повторном запуске программы действительно выводится сообщение о том, что обработчик уже установлен. Вывод сообщения представлен на рисунке 3.

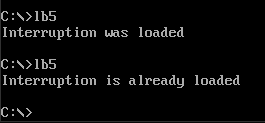


Рисунок 3 – Результаты третьего шага

5. Вызовем программу с параметром /un. Запустим программу прошлой лабораторной работы. Результат представлен на рисунке 4. Как можно заметить, при нажатии клавиш f, left ALT, e, r, z символы выводятся стандартно, а память освобождена.

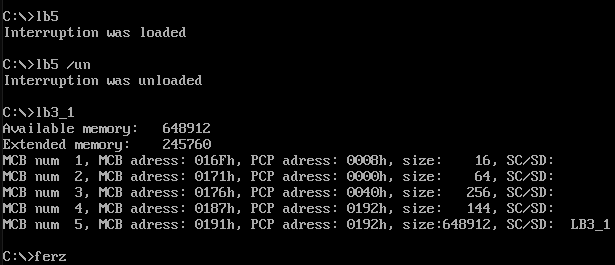


Рисунок 4 – Результаты четвертого шага

Исходный код программы см. в приложении А.

## Ответы на вопросы.

1. Какого типа прерывания использовались в работе?

В работе использовались аппаратные (09h, 16h) и программные (21h) прерывания.

2. Чем отличается скан-код от кода ASCII?

Скан-код – это специальный код, присвоенный каждой клавише, с помощью которого драйвер клавиатуры распознает нажатую клавишу. А ASCII код – это численное представление символов в таблице ASCII (например, ‘a’, ‘b’ и др. имеют и скан-коды, и ASCII-коды, т. к. это и символы, и клавиши, но ‘space’, ‘enter’ и др. – лишь клавиши, у них есть только скан-коды)

## Выводы.

В ходе работы был написан собственный обработчик прерываний клавиатуры, а также была реализована установка и выгрузка данного обработчика.

# Приложение А Исходный код программы

Название файла: lb5.asm

AStack SEGMENT STACK

DW 128 DUP(?)

AStack ENDS

DATA SEGMENT

NOT\_LOAD db 'Interruption did not load', 0DH, 0AH, '$'

LOAD db 'Interruption was loaded', 0DH, 0AH, '$'

UNLOAD db 'Interruption was unloaded', 0DH, 0AH, '$'

ALREADY\_LOAD db 'Interruption is already loaded', 0DH, 0AH, '$'

DATA ENDS

TESTPC SEGMENT

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

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

; ПРОЦЕДУРЫ

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

PRINT PROC near

push AX

mov AH, 09h

int 21h

pop AX

ret

PRINT ENDP

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

MY\_INT PROC far

jmp handle

PSP dw 0

KEEP\_IP dw 0

KEEP\_CS dw 0

KEEP\_SS dw 0

KEEP\_SP dw 0

KEEP\_AX dw 0

KEY\_SYM db 0

signature dw 9871h

IStack db 50 dup(" ")

handle:

mov KEEP\_AX, AX

mov AX, SS

mov KEEP\_SS, AX

mov KEEP\_SP, SP

mov AX, seg IStack

mov SS, AX

mov SP, offset handle

push AX

push BX

push CX

push DX

in AL, 60h

cmp AL, 38h

je space

cmp AL, 2Ch

je z\_key

cmp AL, 12h

je e\_key

pushf

call dword ptr CS:KEEP\_IP

jmp exit\_int

space:

mov KEY\_SYM, '\_'

jmp next\_key

z\_key:

mov KEY\_SYM, 'X'

jmp next\_key

e\_key:

mov KEY\_SYM, 'R'

next\_key:

in AL, 61h

mov AH, AL

or AL, 80h

out 61h, AL

xchg AL, AL

out 61h, AL

mov AL, 20h

out 20h, AL

print\_key:

mov AH, 05h

mov CL, KEY\_SYM

mov CH, 00h

int 16h

or AL, AL

jz exit\_int

mov AX, 40h

mov ES, AX

mov AX, ES:[1Ah]

mov ES:[1Ch], AX

jmp print\_key

exit\_int:

pop DX

pop CX

pop BX

pop AX

mov SP, KEEP\_SP

mov AX, KEEP\_SS

mov SS, AX

mov AX, KEEP\_AX

mov AL, 20h

out 20h, AL

iret

end\_int:

MY\_INT ENDP

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

MY\_INT\_LOAD PROC near

mov PSP, ES

mov AH, 35h

mov AL, 09h

int 21h

mov KEEP\_IP, BX

mov KEEP\_CS, ES

push DS

mov DX, offset MY\_INT

mov AX, seg MY\_INT

mov DS, AX

mov AH, 25h

mov AL, 09h

int 21h

pop DS

mov DX, offset end\_int

mov CL, 4

shr DX, CL

inc DX

mov AX, CS

sub AX, PSP

add DX, AX

mov AL, 0

mov AH, 31h

int 21h

ret

MY\_INT\_LOAD ENDP

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

MY\_INT\_UNLOAD PROC near

CLI

push DS

mov AX, ES:[KEEP\_CS]

mov DS, AX

mov DX, ES:[KEEP\_IP]

mov AH, 25h

mov AL, 09h

int 21h

pop DS

STI

mov AX, ES:[PSP]

mov ES, AX

push ES

mov AX, ES:[2Ch]

mov ES, AX

mov AH, 49h

int 21h

pop ES

int 21h

ret

MY\_INT\_UNLOAD ENDp

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

IS\_LOADED PROC near

push BX

push ES

mov AH, 35h

mov AL, 09h

int 21h

mov AX, ES:[signature]

cmp AX, 9871h

je loaded

mov AL, 0h

jmp end\_isloaded

loaded:

mov AL, 01h

end\_isloaded:

pop ES

pop BX

ret

IS\_LOADED ENDP

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

IS\_FLAG PROC near

push BP

mov BP, 0082h

mov AL, ES:[BP]

cmp AL, '/'

jne not\_good

mov AL, ES:[BP+1]

cmp AL, 'u'

jne not\_good

mov AL, ES:[BP+2]

cmp AL, 'n'

jne not\_good

mov AL, 01h

jmp good

not\_good:

mov AL, 0h

good:

pop BP

ret

IS\_FLAG endp

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

MAIN PROC far

mov ax, data

mov ds, ax

call IS\_FLAG

mov BX, AX

call IS\_LOADED

cmp AL, 0h

je not\_loaded

cmp BL, 0h

jne int\_unload

mov DX, offset ALREADY\_LOAD

call PRINT

jmp end\_main

not\_loaded:

cmp BL, 0h

je int\_load

mov DX, offset NOT\_LOAD

call PRINT

jmp end\_main

int\_load:

mov DX, offset LOAD

call PRINT

call MY\_INT\_LOAD

jmp end\_main

int\_unload:

mov AH, 35h

mov AL, 09h

int 21h

mov DX, offset UNLOAD

call PRINT

call MY\_INT\_UNLOAD

end\_main:

xor AL, AL

mov AH, 4Ch

int 21h

MAIN ENDP

TESTPC ENDS

END MAIN