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 прерывани

Студент гр. 0382 _____ Кондратов Ю.А.

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.

Постановка задачи.

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.

Используемые функции

1. custom_int – созданный обработчик прерывания;
2. load_int – функция, выполняющая загрузку обработчика прерывания в память;
3. unload_int – функция, выполняющая выгрузку обработчика прерывания из памяти;
4. find_cmd_flag – функция для определения флага командной строки;
5. is_loaded – функция, определяющая загружен ли созданный обработчик прерывания в память

Выполнение работы.

Исходный код модуля представлен в приложении А.

Шаг 1. В результате выполнения шага 1 был написан исполняемый модуль типа .exe, который совершает все необходимые по условию действия, при нажатии «x», «y», «z» выводятся «i», «j», «k» соответственно, а при нажатии правого shift выводится «_».

Шаг 2. Результат корректной работы модуля представлен на рисунке 1 (при нажатии три раза на правый shift и x,y,z выводится «___ijk»).

Рисунок 1 – Результат работы модуля main.exe

Шаг 3. Для выполнения шага 2 был взят модуль main1 из лабораторной работы 3. Результат выполнения модуля main1.com представлен на рисунке 2. Из рисунка видно, что обработчик прерывания действительно находится в основной памяти.

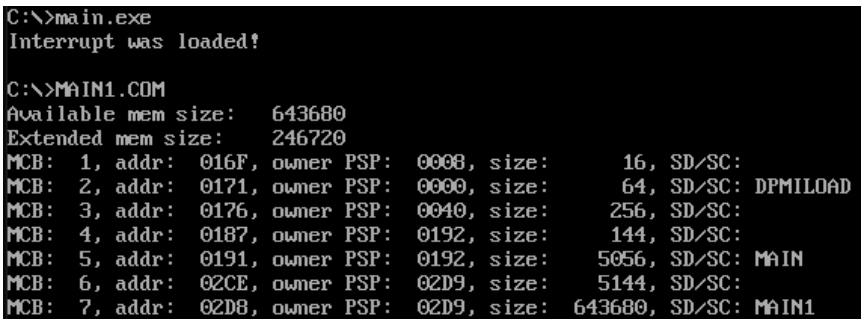

Рисунок 2 – Результат работы модуля main1.com

Шаг 4. Отлаженный модуль main.exe был запущен ещё раз, в результате было выведено сообщение о том, что обработчик уже находится в памяти (см. рисунок 3).

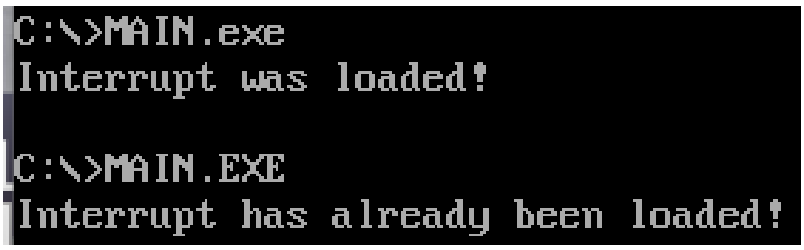

Рисунок 3 – Результат вторичного запуска модуля main.exe

Шаг 5. При запуске модуля с ключом выгрузки было выведено сообщение о том, что пользовательский обработчик успешно выгружен из памяти. Как можно видеть из рисунка 4, после ещё одного запуска модуля main.exe обработчик снова был успешно загружен в память.

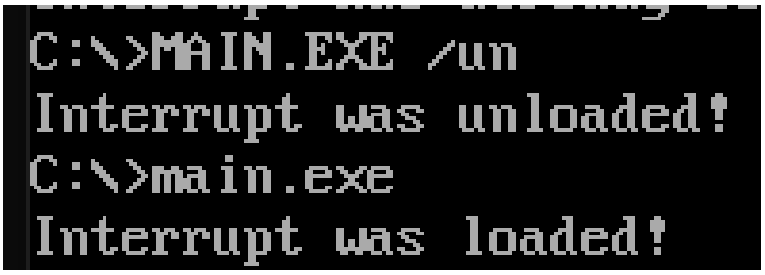

Рисунок 4 – Демонстрация успешной выгрузки пользовательского обработчика

Шаг 6. Контрольные вопросы.

Сегментный адрес недоступной памяти:

1. Какого типа прерывания использовались в работе?

Программные - 21h и аппаратные 09h и 16h.

2. Чем отличается скан-код от кода ASCII?

Скан код – это код, с помощью которого драйвер клавиатуры опознаёт нажатую клавишу. Код ASCII – это код каждого символа в таблице ASCII. Так, например, клавиша shift не является символом и, естественно, не имеет никакого ASCII кода, однако имеет скан код.

Выводы.

В процессе выполнения данной лабораторной работы был изучен механизм работы прерывания 09h и считывания введённых клавиш. Проведена работа по созданию резидентного пользовательского обработчика прерывания, заменяющего действия, производимые при нажатии клавиш rshift, x, y, z.

ПРИЛОЖЕНИЕ А.

Исходный код модуля

main.asm:

```
AStack  SEGMENT STACK
 DW  256 dup(?)
AStack  ENDS


DATA SEGMENT
    load_flag       DB                                          0
    unload_flag     DB                                          0
    load_msg        DB      "Interrupt was loaded!", 0DH, 0AH, "$"
    in_mem_msg       DB       "Interrupt has already been loaded!",
0DH, 0AH, "$"
    unload_msg      DB      "Interrupt was unloaded!", 0DH, 0AH, "$"
    not_loaded_msg  DB      "Interrupt wasnt loaded!", 0DH, 0AH, "$"
DATA ENDS

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


custom_int PROC FAR
    jmp  int_start

    key_value DB 0
    key_code DB 36h
    int_sign DW 6666h
    old_ip DW 0
    old_cs DW 0
    old_psp DW 0
    old_ax DW 0
    old_ss DW 0
    old_sp DW 0

    new_stack DW 256 dup(?)
```

```
int_start:
    mov old_ax, AX
    mov old_sp, SP
    mov old_ss, SS
    mov AX, SEG new_stack
    mov SS, AX
    mov AX, OFFSET new_stack
    add AX, 256
    mov SP, AX

    push AX
    push BX
    push CX
    push DX
    push SI
    push ES
    push DS
    mov AX, SEG key_value
    mov DS, AX

    in AL, 60h
    cmp AL, key_code
    je rshift_k
    cmp AL, 1Eh
    je x_k
    cmp AL, 30h
    je y_k
    cmp AL, 2Eh
    je z_k

    pushf
    call dword ptr CS:old_ip
    jmp end_int

rshift_k:
    mov key_value, '_'
    jmp next_key
x_k:
    mov key_value, 'i'
```

```
    jmp next_key
y_k:
    mov key_value, 'j'
    jmp next_key
z_k:
    mov key_value, 'k'

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_value
    mov CH, 00h
    int 16h
    or AL, AL
    jz end_int
    mov AX, 40h
    mov ES, AX
    mov AX, ES:[1Ah]
    mov ES:[1Ch], AX
    jmp print_key


end_int:
    pop  DS
    pop  ES
    pop  SI
    pop  DX
    pop  CX
    pop  BX
    pop  AX
```

```
    mov SP, old_sp
    mov AX, old_ss
    mov SS, AX
    mov AX, old_ax

    mov  AL, 20h
    out  20h, AL
    iret
custom_int ENDP
 _end:


is_loaded PROC near
    push AX
    push BX
    push SI

    mov  AH, 35h
    mov  AL, 09h
    int  21h
    mov  SI, OFFSET int_sign
    sub  SI, OFFSET custom_int
    mov  AX, ES:[BX + SI]
    cmp  AX, int_sign
    jne  end_proc
    mov  load_flag, 1

end_proc:
    pop  SI
    pop  BX
    pop  AX
    ret
is_loaded ENDP

load_int PROC near
    push AX
    push BX
    push CX
```

```
    push DX
    push ES
    push DS

    mov AH, 35h
    mov AL, 09h
    int 21h
    mov old_cs, ES
    mov old_ip, BX
    mov AX, SEG custom_int
    mov DX, OFFSET custom_int
    mov DS, AX
    mov AH, 25h
    mov AL, 09h
    int 21h
    pop DS

    mov DX, OFFSET _end
    mov CL, 4h
    shr DX, CL
    add DX, 10Fh
    inc DX
    xor AX, AX
    mov AH, 31h
    int 21h

    pop ES
    pop DX
    pop CX
    pop BX
    pop AX
ret
load_int ENDP


unload_int PROC near
    cli
    push AX
    push BX
```

```
push DX
push DS
push ES
push SI

mov AH, 35h
mov AL, 09h
int 21h
mov SI, OFFSET old_ip
sub SI, OFFSET custom_int
mov DX, ES:[BX + SI]
mov AX, ES:[BX + SI + 2]

push DS
mov DS, AX
mov AH, 25h
mov AL, 09h
int 21h
pop DS

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

sti

pop SI
pop ES
pop DS
pop DX
pop BX
pop AX
```

```
ret
unload_int ENDP


find_cmd_flag  PROC near
    push AX
    push ES

    mov AX, old_psp
    mov ES, AX
    cmp byte ptr ES:[82h], '/'
    jne end_unload
    cmp byte ptr ES:[83h], 'u'
    jne end_unload
    cmp byte ptr ES:[84h], 'n'
    jne end_unload
    mov unload_flag, 1

end_unload:
    pop ES
    pop AX
 ret
find_cmd_flag ENDP


PRINT PROC near
    push AX
    mov AH, 09h
    int 21h
    pop AX
ret
PRINT ENDP


BEGIN PROC far
    push DS
    xor AX, AX
    push AX
```

```
    mov AX, DATA
    mov DS, AX
    mov old_psp, ES

    call is_loaded
    call find_cmd_flag
    cmp unload_flag, 1
    je unload
    mov AL, load_flag
    cmp AL, 1
    jne load
    mov DX, OFFSET in_mem_msg
    call PRINT
    jmp end_begin

load:
    mov DX, OFFSET load_msg
    call PRINT
    call load_int
    jmp  end_begin

unload:
    cmp  load_flag, 1
    jne  not_loaded
    mov DX, OFFSET unload_msg
    call PRINT
    call unload_int
    jmp  end_begin
not_loaded:
    mov  DX, offset not_loaded_msg
    call PRINT
end_begin:
    xor AL, AL
    mov AH, 4ch
    int 21h
BEGIN ENDP
CODE ENDS

END BEGIN
```