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

Студент гр. 9383 Звега А.Р.

Преподаватель Ефремов М.А.

Санкт-Петербург
2021

Цель работы.

Исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Пользовательский обработчик прерывания получает управления 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

Шаг1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет такие же функции, как в программе ЛР4, а именно:

1) Проверяет, установлено ли пользовательское прерывание с вектором 1Ch.

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.

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) Сохранять 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.

Выполнение работы

Был написан и отлажен программный модуль типа .EXE, который проверяет, установлено ли пользовательское прерывание с вектором 09h, устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено, и осуществляется выход по функции 4Ch прерывания int 21h. Если прерывания 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.

Была запущена программа, работа прерывания отображается на экране. Выполнена проверка работы обработчика прерывания, при сочетании ctrl+a выводится ‘^’.

Рисунок 1 - Демонстрация работы

Выполнена проверка размещения прерывания в памяти, для этого запущена программа lab3.com.

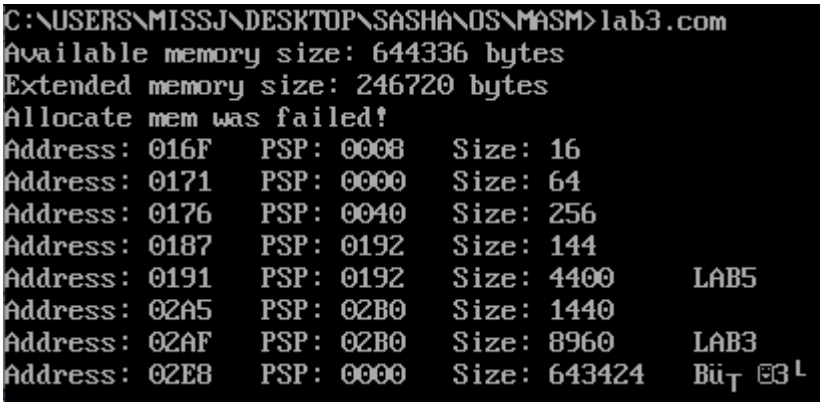

Рисунок 2 - Проверка размещения в памяти

Выполнена проверка, что программа определяет установленный обработчик.

Рисунок 3 - Проверка установленного обработчика

Программа запущена с ключом выгрузки ‘/un’. Запущена программа lab3.com, для проверки, что память освобождена.

Рисунок 4 - Проверка выгрузки

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ ПО ЛАБОРАТОРНОЙ №5

1) Какого типа прерывания использовались в работе?

Аппаратные – обеспечивают реакцию процессора на события, происходящие асинхронно по отношению к исполняемому программному коду. (прерывания от контроллера клавиатуры)

Программные – вызываются с помощью команды int, для обращение к специальным функциям операционной системы. (21h, 16h).

2) Чем отличается скан-код от кода ASCII?

С помощью скан-кода определяется какая клавиша нажата на клавиатуре. ASCII-код – это кодировка символов.

Выводы.

Была исследована возможность встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

Приложение А.

Исходный код программы lab5.asm:

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

DATA  SEGMENT
   LOADED db "Custom interruption is already loaded.$"
   LOAD db "Load custom interruption.$"
   NOT_LOADED db "Default interruption can't be unloaded.$"
   UNLOAD db "Custom interruption was unloaded.$"
DATA  ENDS

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

PRINT_BUF proc near
   push ax
   mov ah, 9h
   int 21h
   pop ax
   ret
PRINT_BUF endp


start_castom:
CUSTOM proc far
   jmp start_proc
   key_value db 0
   KEEP_PSP dw 0
```

```
    KEEP_IP dw 0
        KEEP_CS dw 0
    KEEP_SS DW 0
        KEEP_SP DW 0
        KEEP_AX DW 0
    CUSTOM_INDEX dw 1000h
    TIMER_COUNTER db 'Timer: 0000$'
    BStack DW 64 DUP(?)
start_proc:
    mov KEEP_SP, sp
    mov KEEP_AX, ax
    mov KEEP_SS, ss

    mov ax, seg BStack
    mov ss, ax
    mov ax, offset start_proc
    mov sp, ax

    mov ax, KEEP_AX


    push ax
    push bx
    push cx
    push dx
    push si
    push es
    push ds

        mov cx, 040h
```

```
        mov es, cx
        mov cx, es:[0017h]

    and cx, 0100b
    jz standart_interruption

    in al, 60h
    cmp al, 1Eh
    je do_req

standart_interruption:
    call dword ptr cs:[KEEP_IP]
    jmp restore_registers
do_req:
    in al, 61h
    mov ah, al
    or al, 80h
    out 61h, al
    xchg ah, al
    out 61h, al
    mov al, 20H
    out 20h, al

write_symbol:
    mov ah, 05h
    mov cl, '^'
    mov ch, 00h
    int 16h
    or al, al
    jnz skip
```

```
    jmp restore_registers
skip:
    mov al, es:[001Ah]
    mov es:[001Ch], al
    jmp write_symbol

restore_registers:
    pop ds
    pop es
    popsi
    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_custom:
CUSTOM 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_CUSTOM PROC near
        push ax
        push dx

    mov KEEP_PSP, es
```

```
        mov ah,35h
        mov al,09h
        int 21h
    mov KEEP_IP, bx
    mov KEEP_CS, es

        push ds
        lea dx, CUSTOM
        mov ax, SEG CUSTOM
        mov ds,ax
        mov ah,25h
        mov al,09h
        int 21h
        pop ds

        lea dx, end_custom
        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_CUSTOM endp

UNLOAD_CUSTOM PROC near
```

```
        push ax
        push si

cli
        push ds
        mov ah,35h
        mov al,09h
int 21h

mov si,offset KEEP_IP
sub si,offset CUSTOM
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_CUSTOM endp

IF_LOADED proc near
        push ax
        push si

    push es
    push dx

        mov ah,35h
        mov al,09h
        int 21h

        mov si, offset CUSTOM_INDEX
        sub si, offset CUSTOM
        mov dx,es:[bx+si]
        cmp dx, CUSTOM_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 is_already_set
    mov dx, offset LOAD
    call PRINT_BUF
    call LOAD_CUSTOM
    jmp exit

need_unload:
    call IF_LOADED
    cmp ch, 1h
    jne cant_be_unloaded
    call UNLOAD_CUSTOM
    mov dx, offset UNLOAD
    call PRINT_BUF
    jmp exit
```

```
cant_be_unloaded:
    mov dx, offset NOT_LOADED
    call PRINT_BUF
    jmp exit
is_already_set:
    mov dx, offset LOADED
    call PRINT_BUF
    jmp exit

exit:
    mov ah, 4ch
    int 21h
MAIN endp
CODE ends
END Main
```