

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. 9382

Дерюгин Д.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

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:

Проверяет, установлено ли пользовательское прерывание с вектором 09h.

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.

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:

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.

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.

Прерывание заменяет все символы 'f' на 0. На рис. 1 видно, что программа работает корректно и заменяет все 'f' на 0, а также при выгрузки обработчика буквы не меняются

```
D:\>LAB5.EXE
loading complete

D:\>Orion0a
Illegal command: Orion0a.

D:\>LAB5.EXE /un
Interruption was already loaded
/un loaded
Reset interruption

D:\>frionfaS_
```

Рис. 1 Результат работы программы

Теперь проверим память после загрузки прерывания.

```
PSP address: 0000
Size of paragraph: 64
Last 8 bytes:

PSP address: 0040
Size of paragraph: 256
Last 8 bytes:

PSP address: 0192
Size of paragraph: 144
Last 8 bytes:

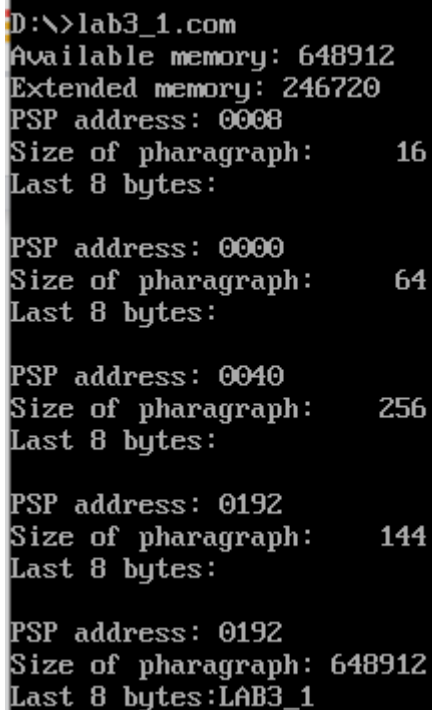
PSP address: 0192
Size of paragraph: 2368
Last 8 bytes: LAB5

PSP address: 0231
Size of paragraph: 2144
Last 8 bytes:

PSP address: 0231
Size of paragraph: 646368
Last 8 bytes: LAB3_1
```

Рис. 2 Память после загрузки прерывания

Выгрузим прерывание из памяти.



```
D:\>lab3_1.com
Available memory: 648912
Extended memory: 246720
PSP address: 0008
Size of paragraph:      16
Last 8 bytes:

PSP address: 0000
Size of paragraph:      64
Last 8 bytes:

PSP address: 0040
Size of paragraph:     256
Last 8 bytes:

PSP address: 0192
Size of paragraph:     144
Last 8 bytes:

PSP address: 0192
Size of paragraph: 648912
Last 8 bytes: LAB3_1
```

Рис 3. Память после выгрузки прерывания

Ответы на вопросы.

1. Какого типа прерывания использовались в работе?

Использовались аппаратные и программные прерывания (21h, 9h, 16h)

2. Чем отличается скан код от кода ASCII?

скан-код - код, который присвоен каждой клавише на клавиатуре, с помощью которого драйвер клавиатуры узнает, какая клавиша нажата.

ASCII - код символа в таблице ASCII.

Выводы.

В ходе данной лабораторной работы была сделана программа, которая встраивает пользовательский обработчик прерывания в обработчик клавиатуры.

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
AStack SEGMENT STACK
    DB 1000 DUP(?)
AStack ENDS

DATA SEGMENT
;IS_INTERRUPT_LOAD dw, 0
;IS_UN_LOAD dw, 0
LOADED dw 0
INTERRUPT_LOADED db 'Interrupt was already loaded' , 0dh, 0ah, '$'
UN_LOADED db '/un loaded', 0dh, 0ah, '$'
RESET_INTERRUPT db 'Reset interruption', 0dh, 0ah, '$'
COMPLITE_LOADING db 'loading complete', 0dh, 0ah, '$'
NOT_INTERRUPT db 'Interruption is not loaded', 0dh, 0ah, '$'
DATA ENDS

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

ROUT PROC FAR
    jmp start

    KEEP_CS DW 0; для хранения сегмента
    KEEP_IP DW 0; и мещения прерывания
    KEEP_SP DW 0
    KEEP_SS DW 0
    KEEP_PSP DW 0
    SIGNATURE DW 1234h
    int9_vect dd 0
    REQ_KEY db 21h; f
    new_stack dw 100h dup (?)

    start:

    mov KEEP_SP, sp
    mov KEEP_SS, ss
    mov sp, offset new_stack
    add sp, 100
    push ax
    mov ax, seg new_stack
```

```

mov ss, ax
pop ax

push ax
push cx
push ds
push es

in al, 60h
cmp al, REQ_KEY
je do_req

pop es
pop ds
pop cx
pop ax
mov ss, KEEP_SS
mov sp, KEEP_SP
jmp cs:[int9_vect]

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

print_0:
mov ah, 05h
mov cl, '0'
mov ch, 00h
int 16h
or al, al
jnz skip
jmp end_int

skip:

```

```

    mov ax, 0040h
    mov es, ax
    mov ax, es:[1ah]
    mov es:[1ch], ax
    jmp print_0

end_int:
    pop es
    pop ds
    pop cx
    pop ax
    mov ss, KEEP_SS
    mov sp, KEEP_SP
    iret

ROUT endp

print proc near
    mov     AH,     09h
    int     21h

    ret
print ENDP

IS_INERRUPTION_SET proc near
    push bx
    push es
    push si

    mov ah, 35h; функция получения вектора
    mov al, 09h; номер вектора
    int 21h

    mov si, offset SIGNATURE
    sub si, offset ROUT
    cmp es:[bx+si], 1234h
    jne return

    mov dx, offset INTERRUPT_LOADED
    call print

```



```

mov LOADED, 1
jmp end_ret

return:
mov KEEP_IP, bx; запоминание смещения
mov KEEP_CS, es; и сегмента

end_ret:
pop si
pop es
pop bx
ret
IS_INERRUPTION_SET endp

IS_UN_SET proc near

    mov al, es:[81h + 1]
    cmp al, '/'
    jne ending
    mov al, es:[81h + 2]
    cmp al, 'u'
    jne ending
    mov al, es:[81h + 3]
    cmp al, 'n'
    jne ending

    cmp LOADED, 1
    je int_and_un

    mov dx, offset NOT_INTERRUPTION
    call print
    mov LOADED, 2
    jmp ending

int_and_un:
mov LOADED, 10
mov dx, offset UN_LOADED
call print

ending:

```

```

ret
IS_UN_SET ENDP

FREE_MEMORY proc near

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

    cli
    push ds
    mov dx, es:KEEP_IP
    mov ax, es:KEEP_CS
    mov ds, ax
    mov ah, 25h
    mov al, 09h
    int 21h; восстанавливаем вектор
    pop ds
    sti

    mov ax, es:KEEP_PSP
    mov es, ax
    push es
    mov ax, es:[2ch]; адрес среды
    mov es, ax

    mov ah, 49h
    int 21h; освобождение среды

    pop es
    mov ah,49h
    int 21h

    mov dx, offset RESET_INTERRUPT
    call print

```

```

ret
FREE_MEMORY ENDP

SET_INTERRUPTION proc near

    mov ah, 35h
    mov al, 09h
    int 21h
    mov KEEP_CS, es
    mov KEEP_IP, bx
    mov word ptr int9_vect[02h], es
    mov word ptr int9_vect, bx

    push ds
    push ax
    push dx
    mov dx, offset ROUT
    mov ax, seg ROUT
    mov ds, ax
    mov ah, 25h
    mov al, 09h
    int 21h

    pop dx
    pop ax
    pop ds
ret
SET_INTERRUPTION ENDP

SAVE_MEMORY proc near

    push ax
    push bx
    push dx
    push cx

    mov dx, offset COMPLITE_LOADING
    call print

    mov DX, offset LAST
    mov cl, 4h
    shr dx, cl

```

```

    inc dx
    mov ax,cs
    sub ax, KEEP_PSP
    add dx,ax
    xor ax,ax
    mov ah,31h
    int 21h

    pop cx
    pop dx
    pop bx
    pop ax

ret
SAVE_MEMORY ENDP

Main proc far

    mov ax, DATA
    mov ds, ax
    mov KEEP_PSP, es

    call IS_INERRUPTION_SET
    call IS_UN_SET

    cmp LOADED, 10
    je free_mem
    cmp LOADED, 2
    je finish
    cmp LOADED, 1
    je finish
    call SET_INTERRUPTION
    call SAVE_MEMORY
    jmp finish

free_mem:
    call FREE_MEMORY

```

```
finish:
mov ah, 4ch
int 21h
LAST:
```

```
Main endp
```

```
CODE ENDS
```

```
END Main
```