

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.

Студентка гр. 0382

Михайлова О.Д.

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.

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.


Выполнение работы.

Для выполнения задания были использованы шаблоны из методических указаний, а также были добавлены следующие процедуры:

- PRINT_STRING – процедура вывода строки на экран;
- MY_INTERRUPT – пользовательский обработчик прерывания;
- CHECK_COMMAND – проверка наличия ключа /un при запуске программы;
- IS_INTERRUPT_LOAD – проверка, загружено ли пользовательское прерывание;
- LOAD_INTERRUPT – загрузка обработчика прерывания в память;
- INTERRUPT_UNLOAD – выгрузка пользовательского прерывания из памяти.

Шаг 1. Был написан и отлажен программный модуль .EXE, который выполняет все заданные в условии функции и заменяет вводимые символы ‘х’ и ‘у’ на ‘р’ и ‘q’ соответственно.

Шаг 2. Была запущена отлаженная программа, а затем была введена последовательность символом ‘abcdxyz’ для проверки корректной работы программы.



```
C:\>lab5.exe
Interrupt was load successfully
C:\>abcdpqz
```

Рисунок 1 - Результат запуска модуля lab5.exe

Шаг 3. Был запущен модуль .COM из лабораторной работы №3 для проверки того, что прерывание находится в памяти.

```
C:\>lab3_1.com
Amount of available memory: 647408 b
Size of extended memory: 15360 Kb
MCB table:
MCB type: 4D, MCB address: 016F, PSP address: 0008, Size: 16, SC/CD:
MCB type: 4D, MCB address: 0171, PSP address: 0000, Size: 64, SC/CD:
MCB type: 4D, MCB address: 0176, PSP address: 0040, Size: 256, SC/CD:
MCB type: 4D, MCB address: 0187, PSP address: 0192, Size: 144, SC/CD:
MCB type: 4D, MCB address: 0191, PSP address: 0192, Size: 1328, SC/CD: LAB5
MCB type: 4D, MCB address: 01E5, PSP address: 01F0, Size: 1144, SC/CD:
MCB type: 5A, MCB address: 01EF, PSP address: 01F0, Size: 647408, SC/CD: LAB3_1
C:\>_
```

Рисунок 2 - Результат запуска модуля lab3_1.com

Шаг 4. Отлаженная программа была запущена еще раз, в результате чего на экран было выведено сообщение о том, что обработчик прерывания уже загружен в память.

```
C:\>lab5.exe
Interrupt has already been loaded
```

Рисунок 3 - Результат повторного запуска модуля lab5.exe

Шаг 5. Была запущена программа с ключом выгрузки, в результате чего на экран было выведено сообщение о том, что обработчик прерывания был выгружен из памяти. Для того, чтобы в этом убедиться, повторно был запущен модуль lab3_1.com.

```
C:\>lab5.exe /un
Interrupt was unload

C:\>lab3_1.com
Amount of available memory: 648912 b
Size of extended memory: 15360 Kb
MCB table:
MCB type: 4D, MCB adress: 016F, PSP adress: 0008, Size: 16, SC/CD:
MCB type: 4D, MCB adress: 0171, PSP adress: 0000, Size: 64, SC/CD:
MCB type: 4D, MCB adress: 0176, PSP adress: 0040, Size: 256, SC/CD:
MCB type: 4D, MCB adress: 0187, PSP adress: 0192, Size: 144, SC/CD:
MCB type: 5A, MCB adress: 0191, PSP adress: 0192, Size: 648912, SC/CD: LAB3_1
```

Рисунок 4 - Запуск модуля lab5.exe с ключом /un и результат повторного запуска модуля lab3_1.com

Исходный код программы см. в приложении А.

Шаг 6. Ответы на контрольные вопросы.

1. Какого типа прерывания использовались в работе?

Аппаратные прерывания 09h и 16h и программное прерывание 21h.

2. Чем отличается скан код от кода ASCII?

Скан код – это код клавиши, с помощью которого драйвер клавиатуры определяет, какая клавиша была нажата. ASCII код – это код символа в таблице кодировок.

Выводы.

В ходе работы были исследованы возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab5.asm

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

DATA SEGMENT
    INT_LOAD db "Interrupt was load successfully", 0Dh, 0Ah, '$'
    INT_NOT_LOAD db "Interrupt is not load", 0Dh, 0Ah, '$'
    INT_UNLOAD db "Interrupt was unload", 0Dh, 0Ah, '$'
    INT_ALREADY_LOAD db "Interrupt has already been loaded", 0Dh,
0Ah, '$'

    flag_cmd db 0
    flag_load db 0
DATA ENDS

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

PRINT_STRING PROC near
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
PRINT_STRING ENDP

MY_INTERRUPT PROC far
    jmp start_func

    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
    INT_ID dw 5555h

    INTERRUPT_STACK dw 128 dup (?)
    END_INT_STACK dw ?

start_func:
    mov KEEP_SS, ss
    mov KEEP_SP, sp
    mov KEEP_AX, ax
    mov ax, cs
    mov ss, ax
    mov sp, offset END_INT_STACK
```

```

    push ax
    push bx
    push cx
    push dx
    push si
    push es
    push ds
    mov ax, seg value
    mov ds, ax

    in al, 60h
    cmp al, 2Dh
    je key_x
    cmp al, 15h
    je key_y

    pushf
    call dword ptr cs:KEEP_IP
    jmp int_final

key_x:
    mov value, 'p'
    jmp do_req
key_y:
    mov value, 'q'

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

print_key:
    mov ah, 05h
    mov cl, value
    mov ch, 00h
    int 16h
    or al, al
    jz int_final
    mov ax, 40h
    mov es, ax
    mov ax, es:[1ah]
    mov es:[1ch], ax
    jmp print_key

int_final:
    pop ds
    pop es
    pop si
    pop dx
    pop cx
    pop bx

```



```

    pop ax

    mov ax, KEEP_SS
    mov ss, ax
    mov ax, KEEP_AX
    mov sp, KEEP_SP

    mov al, 20h
    out 20h, al
    iret
int_end:
MY_INTERRUPT ENDP

```

```

CHECK_COMMAND PROC NEAR
    push ax
    push es
    mov ax, KEEP_PSP
    mov es, ax
    mov bx, 82h
    mov al, es:[bx]
    inc bx
    cmp al, '/'
    jne check_end

    mov al, es:[bx]
    inc bx
    cmp al, 'u'
    jne check_end

    mov al, es:[bx]
    inc bx
    cmp al, 'n'
    jne check_end
    mov flag_cmd, 1h

check_end:
    pop es
    pop ax
    ret
CHECK_COMMAND ENDP

```

```

IS_INTERRUPT_LOAD PROC NEAR
    push ax
    push bx
    push si

    mov ah, 35h
    mov al, 09h
    int 21h
    mov si, offset INT_ID
    sub si, offset MY_INTERRUPT
    mov dx, es:[bx + si]
    cmp dx, 5555h

```

```

        jne is_load_end
        mov flag_load, 1h

is_load_end:
        pop si
        pop bx
        pop ax

        ret
IS_INTERRUPT_LOAD ENDP


LOAD_INTERRUPT PROC NEAR
        push ax
        push cx
        push dx
        push es
        push ds

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

        mov KEEP_IP, bx
        mov KEEP_CS, es

        mov dx, offset MY_INTERRUPT
        mov ax, seg MY_INTERRUPT
        mov ds, ax

        mov ah, 25h
        mov al, 09h
        int 21h
        pop ds

        mov dx, offset INT_LOAD
        call PRINT_STRING

        mov dx, offset int_end
        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 es
        pop dx
        pop cx
        pop ax

```

```

        ret
LOAD_INTERRUPT ENDP

INTERRUPT_UNLOAD PROC NEAR
    cli
    push ax
    push bx
    push dx
    push si
    push es
    push ds

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

    mov si, offset KEEP_IP
    sub si, offset MY_INTERRUPT
    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

    mov dx, offset INT_UNLOAD
    call PRINT_STRING

    pop es
    pop si
    pop dx
    pop bx
    pop ax

    ret
INTERRUPT_UNLOAD ENDP

```

```

Main PROC FAR

```

```

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

    call CHECK_COMMAND
    cmp flag_cmd, 1
    je unload_int

    call IS_INTERRUPT_LOAD
    cmp flag_load, 0
    je not_load
    mov DX, OFFSET INT_ALREADY_LOAD
    call PRINT_STRING
    jmp final

not_load:
    call LOAD_INTERRUPT
    jmp final

unload_int:
    call IS_INTERRUPT_LOAD
    cmp flag_load, 0
    jne already_load
    mov DX, OFFSET INT_NOT_LOAD
    call PRINT_STRING
    jmp final

already_load:
    call INTERRUPT_UNLOAD

final:
    xor al, al
    mov ah, 4Ch
    int 21h

Main ENDP
CODE ENDS
END Main

```