

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

Деткова А.С.

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.

Функции, используемые в программе:

1. *_print* — напечатать строку, адрес смещения до которой лежит в регистре DX.
2. *my_interruption* - собственный обработчик прерывания.
3. *load_int* — загрузка прерывания в память.
4. *unload_int* — выгрузка прерывания из памяти.
5. *find_cmd_key* — считать ключ командной строки, если есть — флаг = 1.
6. *is_loaded_int* — проверка, загружено ли прерывание в память.
7. *MAIN* — вызывающая функция.

Собственный обработчик обрабатывает следующие символы: left Ctrl, Q, F, заменяя их на *, N, а соответственно.

Шаг 1.

На первом шаге был написан .EXE модуль, который выполняет все требования. Была переписана lab4, где был изменен обработчик прерываний, а также сам вектор прерывания, 1CH на 09H.

Шаг 2.

На втором шаге был запущен модуль .EXE и нажаты следующие клавиши: q, p, x, f, left Ctrl, q. Результаты работы программ см. Рис. 1.

```
C:\>lab5
Interruption was loaded.
C:\>Npxa*N
```

Рисунок 1: Результат второго шага работы.

Как видно по рисунку 1, прерывание было успешно загружено и как следствие при вводе с клавиатуры заданной последовательности q был заменен на N, left Ctrl — на *, f — на a.

Шаг 3.

Была запущена программа lab3_1.com, которая показывает состояние памяти по блокам. Как видно по рисунку 2, обработчик прерываний действительно загружен в память (строка 5).

```
C:\>lab3_1.com
Size of available memory = 647872.
Size of extended memory = 15728640.
MCB: 1; adress: 016FH; PSP: 0008H; size in bytes: 16; SC\SD:
MCB: 2; adress: 0171H; PSP: 0000H; size in bytes: 64; SC\SD: DPMILOAD
MCB: 3; adress: 0176H; PSP: 0040H; size in bytes: 256; SC\SD:
MCB: 4; adress: 0187H; PSP: 0192H; size in bytes: 144; SC\SD:
MCB: 5; adress: 0191H; PSP: 0192H; size in bytes: 864; SC\SD: LAB5
MCB: 6; adress: 01C8H; PSP: 01D3H; size in bytes: 144; SC\SD:
MCB: 7; adress: 01D2H; PSP: 01D3H; size in bytes: 647872; SC\SD: LAB3_1
```

Рисунок 2: Результат третьего шага.

Шаг 4.

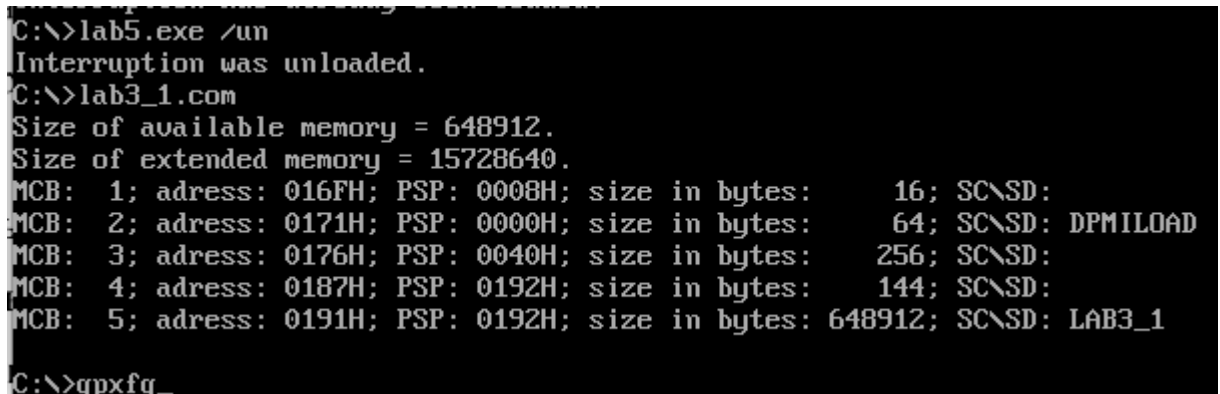
Повторно запустим программу, по рисунку 3 видно, что обработчик прерываний уже загружен в память.

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

Рисунок 3: Результат четвертого шага.

Шаг 5.

Программа была выгружена из памяти с помощью ключа выгрузки `/un`, введенного после имени запускаемого модуля. На рисунке 4 видно, что обработчик выгружен, что его действительно нет в памяти. Также рис. 4 показывает результат ввода той же последовательности нажатых клавиш, что и в шаге 2 (q, p, x, f, left Ctrl, q).



```
C:\>lab5.exe /un
Interruption was unloaded.
C:\>lab3_1.com
Size of available memory = 648912.
Size of extended memory = 15728640.
MCB: 1; adress: 016FH; PSP: 0008H; size in bytes: 16; SC\SD:
MCB: 2; adress: 0171H; PSP: 0000H; size in bytes: 64; SC\SD: DPMILOAD
MCB: 3; adress: 0176H; PSP: 0040H; size in bytes: 256; SC\SD:
MCB: 4; adress: 0187H; PSP: 0192H; size in bytes: 144; SC\SD:
MCB: 5; adress: 0191H; PSP: 0192H; size in bytes: 648912; SC\SD: LAB3_1
C:\>qpxfq_
```

Рисунок 4: Результаты пятого шага.

Исходный код программы см. в приложении А.

Ответы на контрольные вопросы.

1. Какого типа прерывания использовались в работе?

09H, 16H — аппаратные прерывания. 121H — программное прерывание.

2. Чем отличается скан-код от кода ASCII?

Скан-код — код, присвоенный каждой клавише, с помощью которого драйвер клавиатуры распознает, какая клавиша была нажата. ASCII представляет собой кодировку для представления десятичных цифр, латинского и национального алфавитов, знаков препинания и управляющих символов. ASCII-код — численное представление (кодировка) символов из таблицы ASCII. Например, символы `q`, `r`, `1` и другие имеют и ASCII-код, и скан-код, так как это и символы, находящиеся в ASCII-таблице, и клавиши

клавиатуры, но клавиши `space`, `left Ctrl` и другие являются только клавишами, поэтому имеют только скан-коды.

Выводы.

В ходе работы был реализован обработчик прерываний клавиатуры. Проведена работа по созданию резидентного пользовательского обработчика прерывания, заменяющего действия, производимые при нажатии клавиш left Ctrl, q, f.

ПРИЛОЖЕНИЕ А

КОД МОДУЛЕЙ

Название файла: lab5.asm

```
AStack SEGMENT  STACK
                DB 100H DUP('!')
AStack ENDS
```

DATA SEGMENT

```
    flag DB 0
    msg_is_load DB 'Interruption was loaded.$'
    msg_is_in_memory DB 'Interruption has already been loaded.$'
    msg_is_unload DB 'Interruption was unloaded.$'
    msg_is_not_load DB 'Interruption was not loaded.$'
```

DATA ENDS

CODE SEGMENT

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

my_interruption 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
    int_indicator DW 0AAAAH
    LocalStack db 50 dup(" ")
```

```
handle:
    mov KEEP_AX, AX
    mov AX, SS
    mov KEEP_SS, AX
    mov KEEP_SP, SP
    mov AX, seg LocalStack
    mov SS, AX
    mov SP, offset handle
```

```
    push AX
    push BX
    push CX
    push DX
```



```

    in AL, 60h
    cmp AL, 1Dh
    je left_ctrl
    cmp AL, 21h
    je f_key
    cmp AL, 10h
    je q_key

    call dword ptr CS:KEEP_IP
    jmp exit_int

left_ctrl:
    mov KEY_SYM, '*'
    jmp next_key
f_key:
    mov KEY_SYM, 'a'
    jmp next_key
q_key:
    mov KEY_SYM, 'N'

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

    int_end:
my_interruption ENDP

load_int PROC NEAR

    push AX
    push BX
    push CX
    push DX
    push ES

    mov AH,35H
    mov AL,09H
    int 21H
    mov keep_IP,BX
    mov keep_CS,ES

    push DS
    mov DX,offset my_interruption
    mov AX,seg my_interruption
    mov DS,AX
    mov AH,25H
    mov AL,09H
    int 21H
    pop DS

    mov DX,offset int_end
    mov CL,4
    shr DX,CL
    inc DX
    mov AX,CS
    sub AX,PSP
    add DX,AX
    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

    push AX

```

```

push BX
push CX
push DX
push ES
push DS

cli
mov AH, 35H
mov AL, 09H
int 21H
mov DX, ES:[offset keep_IP]
mov AX, ES:[offset keep_CS]
mov DS, AX
mov AH, 25H
mov AL, 09H
int 21H

mov AX, ES:[offset PSP]
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 DS
pop ES
pop DX
pop CX
pop BX
pop AX
ret

```

```
unload_int ENDP
```

```
find_cmd_key PROC NEAR
```

```

push AX
push SI

mov SI, 82H
mov AL, ES:[SI]
cmp AL, '/'
jne end_check

inc SI
mov AL, ES:[SI]
cmp AL, 'u'
jne end_check

```

```

        inc SI
        mov AL,ES:[SI]
        cmp AL,'n'
        jne end_check
        mov flag,1

end_check:
        pop SI
        pop AX
        ret

find_cmd_key ENDP

is_loaded_int PROC NEAR

        push AX
        push DX
        push SI

        mov flag,1
        mov AH,35H
        mov AL,09H
        int 21H

        mov SI,offset int_indicator
        sub SI,offset my_interruption
        mov DX,ES:[BX+SI]
        cmp DX,0AAAAH
        je is_loaded
        mov flag,0

is_loaded:
        pop SI
        pop DX
        pop AX
        ret

is_loaded_int ENDP

_print PROC NEAR

        push AX

        mov AH,09H
        int 21H

        pop AX
        ret

_print ENDP

MAIN      PROC   FAR

```

```

    mov AX,DATA
    mov DS,AX
    mov PSP,ES
    mov flag,0

    call find_cmd_key
    cmp flag,1
    je unload

    call is_loaded_int
    cmp flag,0
    je not_loaded
    mov DX,offset msg_is_in_memory
    call _print
    jmp final

not_loaded:
    mov DX,offset msg_is_load
    call _print
    call load_int
    jmp final

unload:
    call is_loaded_int
    cmp flag,0
    jne already_loaded
    mov DX,offset msg_is_not_load
    call _print
    jmp final

already_loaded:
    call unload_int
    mov DX,offset msg_is_unload
    call _print

final:
    mov AH,4CH
    xor AL,AL
    int 21H

Main      ENDP
CODE      ENDS
          END MAIN

```