

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 ПРЕРЫВАНИЯ

Студент гр. 0382

Сергеев Д.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.

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) при.

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. 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 2 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.

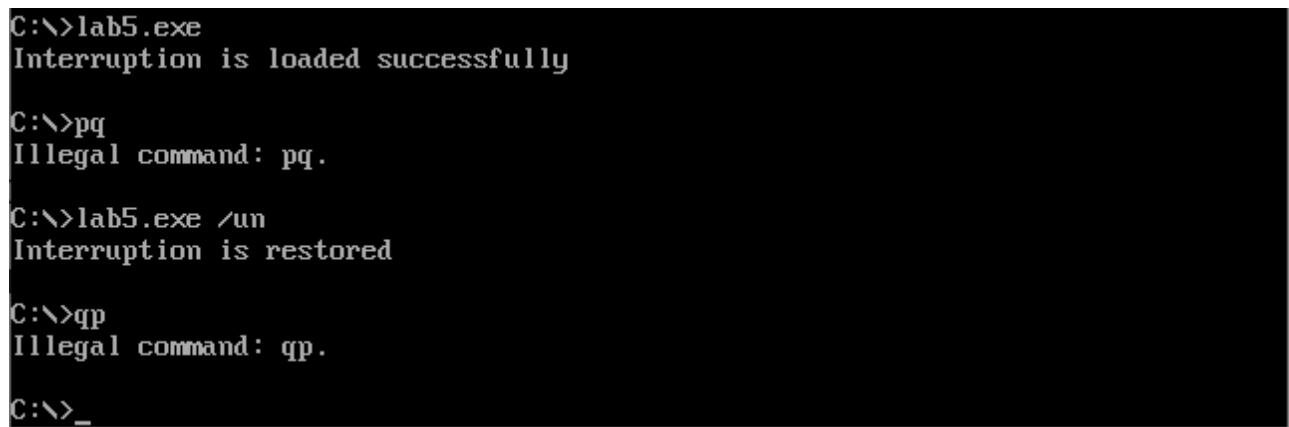
Выполнение работы.

В ходе работы был взят шаблон из методических материалов, также были написаны процедуры:

- CHECK_UPLOAD_KEY – проверяет указан ли параметр /un при запуске программы
- LOAD_INT – загружает прерывание в память
- UNLOAD – выгружает из памяти пользовательское прерывание, освобождает память и восстанавливает исходный вектор прерываний
- MY_INT – пользовательское прерывание, которое меняет символы “q” и “p” местами
- IF_ALREADY_LOAD – проверяет, загружено ли прерывание

Шаг 1. Был написан и отлажен программный модуль типа .EXE, который при вводе в консоль буквы “q” печатает “p” и наоборот.

Шаг 2. Была запущена программа, далее были нажаты клавиши “q” и “p”, затем прерывание было выгружено, после данного действия были заново нажаты упомянутые клавиши.



```
C:\>lab5.exe
Interruption is loaded successfully

C:\>pq
Illegal command: pq.

C:\>lab5.exe /un
Interruption is restored

C:\>qp
Illegal command: qp.

C:\>_
```

Рисунок 1 – Результат работы написанной программы

Шаг 3. С помощью программы из лабораторной работы №3 было проверено размещение прерывания в памяти.

```
C:\>lab5.exe
Interruption is loaded successfully

C:\>prog1.com
Available Memory in Bytes: 644288
Extended Memory in KiloBytes: 15360

MCB List:
MCB #1 Address: 016F PSP TYPE:  Belong MS DOS      Size: 0001 SC/SD:
MCB #2 Address: 0171 PSP TYPE:  Free PSP           Size: 0004 SC/SD:
MCB #3 Address: 0176 PSP TYPE:  0040              Size: 0010 SC/SD:
MCB #4 Address: 0187 PSP TYPE:  0192              Size: 0009 SC/SD:
MCB #5 Address: 0191 PSP TYPE:  0192              Size: 0116 SC/SD:  LAB5
MCB #6 Address: 02A8 PSP TYPE:  02B3              Size: 0009 SC/SD:
MCB #7 Address: 02B2 PSP TYPE:  02B3              Size: 9D4C SC/SD:  PROG1

C:\>
```

Рисунок 2 – Демонстрация размещения прерывания в памяти

Как видно на рисунке, прерывание действительно находится в памяти.

Шаг 4. Программа была повторно запущена, чтобы удостовериться, что программа определяет установленный обработчик прерываний.

```
C:\>lab5.exe
Interruption is loaded successfully

C:\>prog1.com
Available Memory in Bytes: 644288
Extended Memory in KiloBytes: 15360

MCB List:
MCB #1 Address: 016F PSP TYPE:  Belong MS DOS      Size: 0001 SC/SD:
MCB #2 Address: 0171 PSP TYPE:  Free PSP           Size: 0004 SC/SD:
MCB #3 Address: 0176 PSP TYPE:  0040              Size: 0010 SC/SD:
MCB #4 Address: 0187 PSP TYPE:  0192              Size: 0009 SC/SD:
MCB #5 Address: 0191 PSP TYPE:  0192              Size: 0116 SC/SD:  LAB5
MCB #6 Address: 02A8 PSP TYPE:  02B3              Size: 0009 SC/SD:
MCB #7 Address: 02B2 PSP TYPE:  02B3              Size: 9D4C SC/SD:  PROG1

C:\>lab5.exe
Interruption is already loaded

C:\>_
```

Рисунок 3 – Результат выполнения шага 4

Как видно на рисунке 3, было выведено сообщение о том, что обработчик уже находится в памяти.

Шаг 5. Программа была запущена с ключом выгрузки, в результате чего, был восстановлен стандартный обработчик прерываний, также была освобождена память, занятая резидентом.

```
C:\>lab5.exe
Interruption is loaded successfully

C:\>lab5.exe /un
Interruption is restored

C:\>prog1.com
Available Memory in Bytes: 648912
Extended Memory in KiloBytes: 15360

MCB List:
MCB #1 Address: 016F PSP TYPE:  Belong MS DOS      Size: 0001 SC/SD:
MCB #2 Address: 0171 PSP TYPE:  Free PSP           Size: 0004 SC/SD:
MCB #3 Address: 0176 PSP TYPE:  0040              Size: 0010 SC/SD:
MCB #4 Address: 0187 PSP TYPE:  0192              Size: 0009 SC/SD:
MCB #5 Address: 0191 PSP TYPE:  0192              Size: 9E6D SC/SD:  PROG1

C:\>
```

Рисунок 4 – Демонстрация корректной выгрузки резидентного обработчика прерываний

Как видно на рисунке 4, память действительно освобождена, а обработчик прерываний восстановлен.

Исходный программный код смотрите в приложении А.

Контрольные вопросы.

1. Какого типа прерывания использовались в работе?

Ответ:

- 1) 09h и 16h - аппаратные прерывания
- 2) 10h и 21h – программные прерывания
2. Чем отличается скан-код от кода ASCII?

Ответ: ASCII код – код символа в таблице ASCII, а скан-код – уникальное число, которое позволяет однозначно идентифицировать, нажатую клавишу.

Выводы.

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

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab5.asm

```
STACK SEGMENT STACK
    DW 200 DUP(?)
STACK ENDS

DATA SEGMENT
    str_int_already_loaded db 'Interruption is already loaded', 0DH, 0AH, '$'
    str_int_loaded_successfully db 'Interruption is loaded successfully', 0DH,
0AH, '$'
    str_int_is_not_loaded db 'Interruption is not loaded', 0DH, 0AH, '$'
    str_int_restored db 'Interruption is restored', 0DH, 0AH, '$'
DATA ENDS

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

;-----
    PRINT_MESSAGE PROC NEAR
        push AX
        mov AH, 9
        int 21h
        pop AX
        ret
    PRINT_MESSAGE ENDP
;-----

;-----
    MY_INT PROC FAR
        jmp int_start
        KEY db 0h
        SHIFT db 0
        int_signature dw 7777h
        keep_ip dw 0
        keep_cs dw 0
        psp_address dw ?
        keep_ss dw 0
        keep_sp dw 0
        keep_ax dw 0
        int_stack dw 64 dup(?)

    int_start:
        mov keep_sp, SP
        mov keep_ax, AX
        mov AX, SS
        mov keep_ss, AX
        mov SP, OFFSET int_start
        mov AX, seg int_stack
        mov SS, AX
        mov AX, keep_ax
        push AX
        push CX
        push DX
        push ES
```



```

    mov KEY, 0h
    mov SHIFT, 0h
    mov AX, 40h
    mov ES, AX
    mov AX, ES:[17h]
    and AX, 11b
    cmp AX, 0h
    je read_new_symbol
    mov SHIFT, 1h

read_new_symbol:
    in AL, 60h
    cmp AL, 10h
    je symbol_q
    cmp AL, 19h
    je symbol_p
    mov KEY, 1h
    jmp int_end

symbol_q:
    mov AL, 'p'
    jmp change

symbol_p:
    mov AL, 'q'
    jmp change

change:
    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
    cmp SHIFT, 0h
    je write_key
    sub AL, 20h

write_key:
    mov AH, 05h
    mov CL, AL
    mov CH, 00h
    int 16h
    or AL, AL
    jz int_end
    mov AX, 0040h
    mov ES, AX
    mov AX, ES:[1ah]
    mov ES:[1ch], AX
    jmp write_key

int_end:
    pop ES
    pop DX

```

```

        pop CX
        pop AX
        mov SP, keep_sp
        mov AX, keep_ss
        mov SS, AX
        mov AX, keep_ax
        mov AL, 20h
        out 20h, AL
        cmp KEY, 1h
        jne int_iret
        jmp dword ptr CS:[keep_ip]

int_iret:
        iret

MY_INT ENDP
int_last:
;-----

;-----

CHECK_UN PROC near
        push AX
        push BP
        mov CL, 0h
        mov BP, 81h
        mov AL,ES:[BP + 1]
        cmp AL,'/'
        jne final
        mov AL,ES:[BP + 2]
        cmp AL,'u'
        jne final
        mov AL,ES:[BP + 3]
        cmp AL,'n'
        jne final
        mov CL, 1h

final:
        pop BP
        pop AX
        ret
CHECK_UN ENDP

IS_LOAD PROC NEAR
        push AX
        push DX
        push ES
        push SI
        mov CL, 0h
        mov AH, 35h
        mov AL, 09h
        int 21h
        mov SI, offset int_signature
        sub SI, offset MY_INT
        mov DX, ES:[BX + SI]
        cmp DX, int_signature
        jne finish_check_load
        mov CL, 1h

```

finish_check_load:

pop SI
pop ES
pop DX
pop AX
ret

IS_LOAD ENDP

LOAD_INT PROC near

push AX
push CX
push DX
call IS_LOAD
cmp CL, 1h
je already_loaded
mov psp_address, ES
mov AH, 35h
mov AL, 09h
int 21h
mov keep_cs, ES
mov keep_ip, BX
push ES
push BX
push DS
lea DX, MY_INT
mov AX, SEG MY_INT
mov DS, AX
mov AH, 25h
mov AL, 09h
int 21h
pop DS
pop BX
pop ES
mov DX, offset str_int_loaded_successfully
call PRINT_MESSAGE
lea DX, int_last
mov CL, 4h
shr DX, CL
inc DX
add DX, 100h
xor AX, AX
mov AH, 31h
int 21h
jmp finish_load

already_loaded:

mov DX, offset str_int_already_loaded
call PRINT_MESSAGE

finish_load:

pop DX
pop CX
pop AX
ret

LOAD_INT ENDP

```

UNLOAD_INT PROC near
    push AX
    push SI
    call IS_LOAD
    cmp CL, 1h
    jne not_loaded
    cli
    push DS
    push ES
    mov AH, 35h
    mov AL, 09h
    int 21h
    mov SI, offset keep_ip
    sub SI, offset MY_INT
    mov DX, ES:[BX + SI]
    mov AX, ES:[BX + SI + 2]
    mov DS, AX
    mov AH, 25h
    mov AL, 09h
    int 21h
    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
    pop ES
    pop DS
    sti
    mov DX, offset str_int_restored
    call PRINT_MESSAGE
    jmp finish_unload

not_loaded:
    mov DX, offset str_int_is_not_loaded
    call PRINT_MESSAGE

finish_unload:
    pop SI
    pop AX
    ret
UNLOAD_INT ENDP
;-----

;-----
MAIN PROC FAR
    mov AX, DATA
    mov DS, AX
    call CHECK_UN
    cmp CL, 0h
    jne un_unload
    call LOAD_INT
    jmp finish_main

```

```
un_unload:
    call UNLOAD_INT

finish_main:
    xor AL, AL
    mov AH, 4ch
    int 21h
MAIN ENDP
```

```
CODE ENDS
END MAIN
```