

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. 0382

Кривенцова Л.С.

Преподаватель

Губкин А.Ф.

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.

Порядок выполнения работы.

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.

Используемые функции

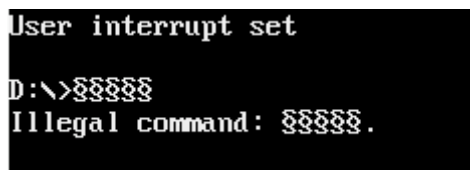
1. ROUT– Обработчик прерывания;
2. Control
3. INT_Setting – функция, выполняющая загрузку обработчика прерывания в память;
4. Delete_INT– функция, выполняющая выгрузку обработчика прерывания из памяти;
5. print– функция вывода в терминал;

Выполнение работы.

Исходный код модуля представлен в приложении А.

Шаг 1. В результате выполнения шага 1 был написан исполняемый модуль типа .exe, реализованный в котором обработчик прерывания получает управление по прерыванию int 9h при нажатии клавиши; сравнивая со скан-кодом – если код совпадает с данным, происходит замена символа.

Шаг 2. Была запущена отлаженная программа: резидентный обработчик прерывания 09h установлен. При нажатии на клавишу delete на экран выводится «§».



```
User interrupt set
D:\>§§§§§§
Illegal command: §§§§§§.
```

Рисунок 1 – Результат работы модуля lb5.exe

Шаг 3. Было продемонстрировано размещение прерывания в памяти: была запущена программа ЛР 3, которая отображает карту памяти в виде списка блоков МСВ. Из рисунка понятно, что обработчик прерывания находится в основной памяти.

```

D:\>lb3.com
available memory :          648912bytes:
extended memory  :    15360      ;
MCB      4D      address: 016F,   PCP: 0008h, Size:    16 SC/SD:
MCB      4D      address: 0171,   PCP: 0000h, Size:    64 SC/SD:
MCB      4D      address: 0176,   PCP: 0040h, Size:   256 SC/SD:
MCB      4D      address: 0187,   PCP: 0192h, Size:   144 SC/SD:
MCB      4D      address: 0191,   PCP: 0192h, Size:  1120 SC/SD:    LB5
MCB      4D      address: 01D8,   PCP: 01E3h, Size:   1144 SC/SD:
MCB      5A      address: 01E2,   PCP: 01E3h, Size: 647616 SC/SD:    LB3

```

Рисунок 2 – Результат работы модуля lb3.com - размещение прерывания в памяти.

Шаг 4. Отлаженная программа была запущена еще раз.

```

User interrupt set
D:\>§§§§§§§§
Illegal command: §§§§§§§.

```

Рисунок 3 – Результат вторичного запуска модуля lb5.exe

Шаг 5. Запущена отлаженная программа с ключом выгрузки: резидентный обработчик прерывания выгружен, то есть память, занятая резидентом освобождена.

```

D:\>lb5.exe /un
User interrupt discharged

```

Рисунок 4 – Демонстрация выгрузки.

```

D:\>lb3.com
available memory:    648912
extended memory:    245760
MCB      1      address: 016F , PCP :    0008 , size:    16, SC/SD:
MCB      2      address: 0171 , PCP :    0000 , size:    64, SC/SD:  DPMILOAD
MCB      3      address: 0176 , PCP :    0040 , size:   256, SC/SD:
MCB      4      address: 0187 , PCP :    0192 , size:   144, SC/SD:
MCB      5      address: 0191 , PCP :    0192 , size:648912, SC/SD:  LB3

```

Рисунок 5 – Демонстрация освобождения памяти.

Шаг 6. Контрольные вопросы.

Сегментный адрес недоступной памяти:

1. Какого типа прерывания использовались в работе?

Программные (21h), аппаратные (09h, 16h).

2. Чем отличается скан-код от кода ASCII?

Скан код – код «клавиши». Код ASCII – это код каждого символа в таблице ASCII. Таким образом, клавиши, которым не присущ никакой символ (нет в таблице ASCII), имеют свои скан-коды.

Выводы.

Был изучен механизм работы прерывания 09h и считывания введенных клавиш. Был реализован собственный обработчик прерывания, обрабатывающий считанные с клавиатуры данные.

ПРИЛОЖЕНИЕ А.

Исходный код модуля

Laba5.asm:

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

DATA SEGMENT
SET db 'User interrupt set' , 0DH, 0AH, '$'
ASSIGN db 'User interrupt assigned', 0DH, 0AH, '$'
DIS db 'User interrupt discharged' , 0DH, 0AH, '$'
DATA ENDS

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

print PROC
    push AX
    mov AH, 09h
    int 21h
    pop AX
    ret
print ENDP

ROUT PROC FAR
    jmp _ROUT
    _STACK dw 100 dup (0)
    SIGN db '0000'
    KEEP_IP dw 0
    KEEP_CS dw 0
    KEEP_PSP dw 0
    KEEP_SS dw 0
    KEEP_AX dw 0
    KEEP_SP dw 0
    FIRST db 53h
    RES db 15h
```

```

_ROUT:
    mov KEEP_SS, SS
    mov KEEP_AX, AX
    mov KEEP_SP, SP
    mov AX, seg _STACK
    mov SS, AX
    mov SP, 0
    mov AX, KEEP_AX
    in AL, 60h
    cmp AL, FIRST
    je DO_REQ
    pushf
    call dword ptr KEEP_IP
    jmp The_end

```

```

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

```

```

Update:
    mov AL, 0
    mov AH, 05h
    mov CL, RES
    mov CH, 00h
    int 16h
    or AL, AL
    jz The_end
    jmp Update

```

```

The_end:
    pop ES

```



```

    pop DS
    pop DX
    pop AX
    mov AX, KEEP_SS
    mov SS, AX
    mov SP, KEEP_SP
    mov AX, KEEP_AX
    mov AL, 20H
    out 20H,AL
    iret
ROUT ENDP

```

Control PROC

```

    mov AH, 35h
    mov AL, 09h
    int 21h
    mov SI, offset SIGN
    sub SI, offset ROUT
    mov AX, '00'
    cmp AX, ES:[BX+SI]
    jne Upload
    cmp AX, ES:[BX+SI+2]
    je Download

```

Upload:

```

    call Int_Setting
    mov DX, offset Size_in_bytes
    mov CL, 4
    shr DX, CL
    inc DX
    add DX, CODE
    sub DX, KEEP_PSP
    xor AL, AL
    mov AH, 31h
    int 21h

```

Download:

```

    push ES
    push AX
    mov AX, KEEP_PSP
    mov ES, AX

```

```

        cmp byte ptr ES:[82h], '/'
        jne stay
        cmp byte ptr ES:[83h], 'u'
        jne stay
        cmp byte ptr ES:[84h], 'n'
        je _Upload
stay:
        pop AX
        pop ES
        mov DX, offset ASSIGN
        call print
        ret
 Upload:
        pop AX
        pop ES
        call Delete_INT
        mov DX, offset DIS
        call print
        ret
Control ENDP

```

```

Int_Setting PROC
        push DX
        push DS
        mov AH, 35h
        mov AL, 1Ch
        int 21h
        mov KEEP_IP, BX
        mov KEEP_CS, ES
        mov dx, offset ROUT
        mov ax, seg ROUT
        mov DS, AX
        mov AH, 25h
        mov AL, 1Ch
        int 21h
        pop DS
        mov DX, offset SET
        call PRINT
        pop DX

```

```

        ret
Int_Setting ENDP

Delete_INT PROC
    CLI
    push DS
    mov DX, ES:[BX+SI+4]
    mov AX, ES:[BX+SI+6]
    mov DS, AX
    mov AX, 2509h
    int 21h
    push ES
    mov AX, ES:[BX+SI+8]
    mov ES, AX
    mov ES, ES:[2Ch]
    mov AH, 49h
    int 21h
    pop ES
    mov ES, ES:[BX+SI+8]
    mov AH, 49h
    int 21h
    pop DS
    STI
    ret
Delete_INT ENDP

Main PROC FAR
    mov AX, DATA
    mov DS, AX
    mov KEEP_PSP, ES
    call Control
    xor AL, AL
    mov AH, 4Ch
    int 21h
Main ENDP

    Size_in_bytes:
CODE ENDS

    END Main

```