

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ
по лабораторной работе №5
по дисциплине «Операционные системы»
Тема: Сопряжение стандартного и пользовательского обработчиков
прерываний

Студент гр. 9382

Рыжих Р.В.

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:

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.

Ход работы

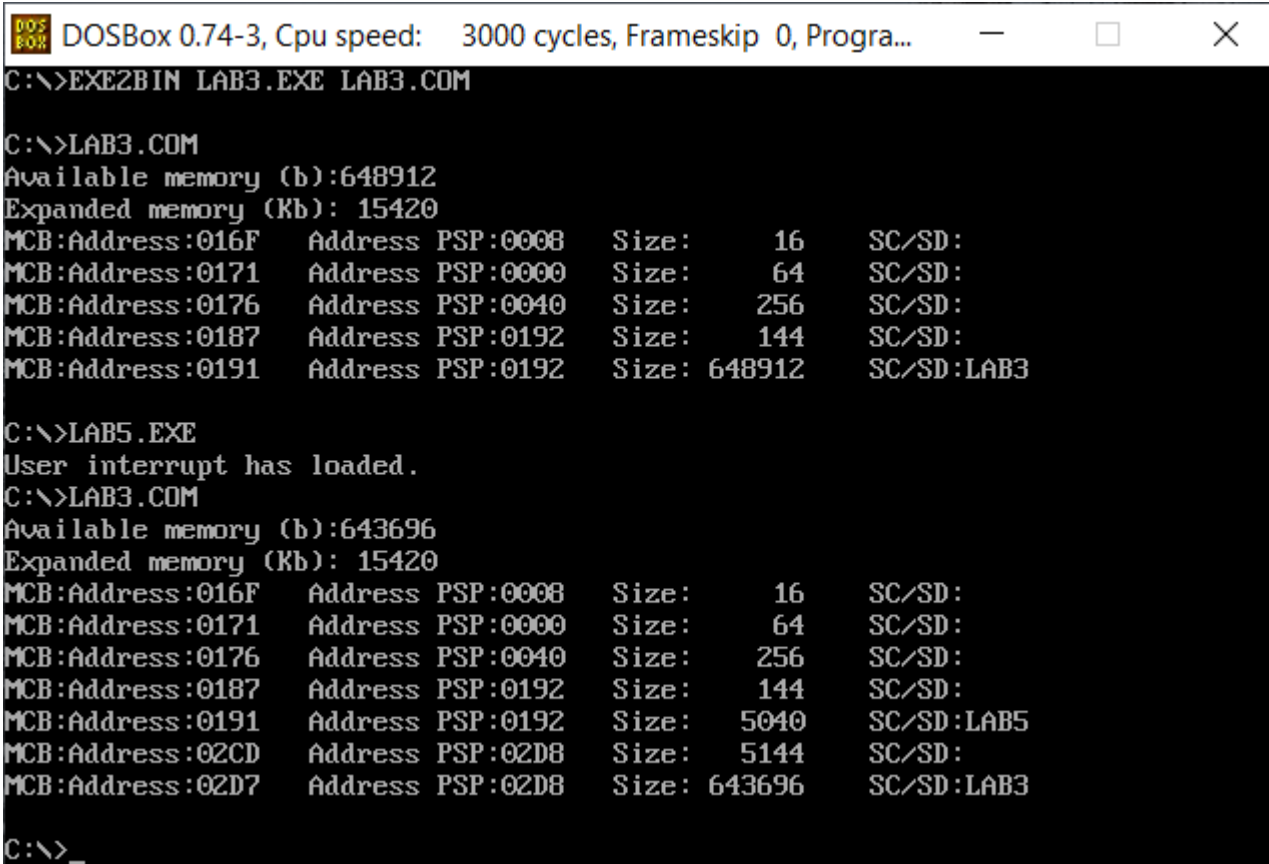
Шаг 1. Был написан и отлажен программный модуль типа .EXE, который выполняет, данные в задании функции.

Шаг 2. Была запущена программа Lab5.exe. Видно, что обработчик прерываний работает успешно. Прерывание меня символы 'D', 'W', 'H' на „!“ , „\$“ , „#“ соответственно.

```
C:\>LAB5.EXE
User interrupt has loaded.
C:\>!$g_
```

Рис. 1: Демонстрация работы резидентного обработчика прерываний.

Шаг 3. Было проверено размещение прерывания в памяти.



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
C:\>EXE2BIN LAB3.EXE LAB3.COM
C:\>LAB3.COM
Available memory (b):648912
Expanded memory (Kb): 15420
MCB:Address:016F Address PSP:0008 Size: 16 SC/SD:
MCB:Address:0171 Address PSP:0000 Size: 64 SC/SD:
MCB:Address:0176 Address PSP:0040 Size: 256 SC/SD:
MCB:Address:0187 Address PSP:0192 Size: 144 SC/SD:
MCB:Address:0191 Address PSP:0192 Size: 648912 SC/SD:LAB3
C:\>LAB5.EXE
User interrupt has loaded.
C:\>LAB3.COM
Available memory (b):643696
Expanded memory (Kb): 15420
MCB:Address:016F Address PSP:0008 Size: 16 SC/SD:
MCB:Address:0171 Address PSP:0000 Size: 64 SC/SD:
MCB:Address:0176 Address PSP:0040 Size: 256 SC/SD:
MCB:Address:0187 Address PSP:0192 Size: 144 SC/SD:
MCB:Address:0191 Address PSP:0192 Size: 5040 SC/SD:LAB5
MCB:Address:02CD Address PSP:02D8 Size: 5144 SC/SD:
MCB:Address:02D7 Address PSP:02D8 Size: 643696 SC/SD:LAB3
C:\>_
```

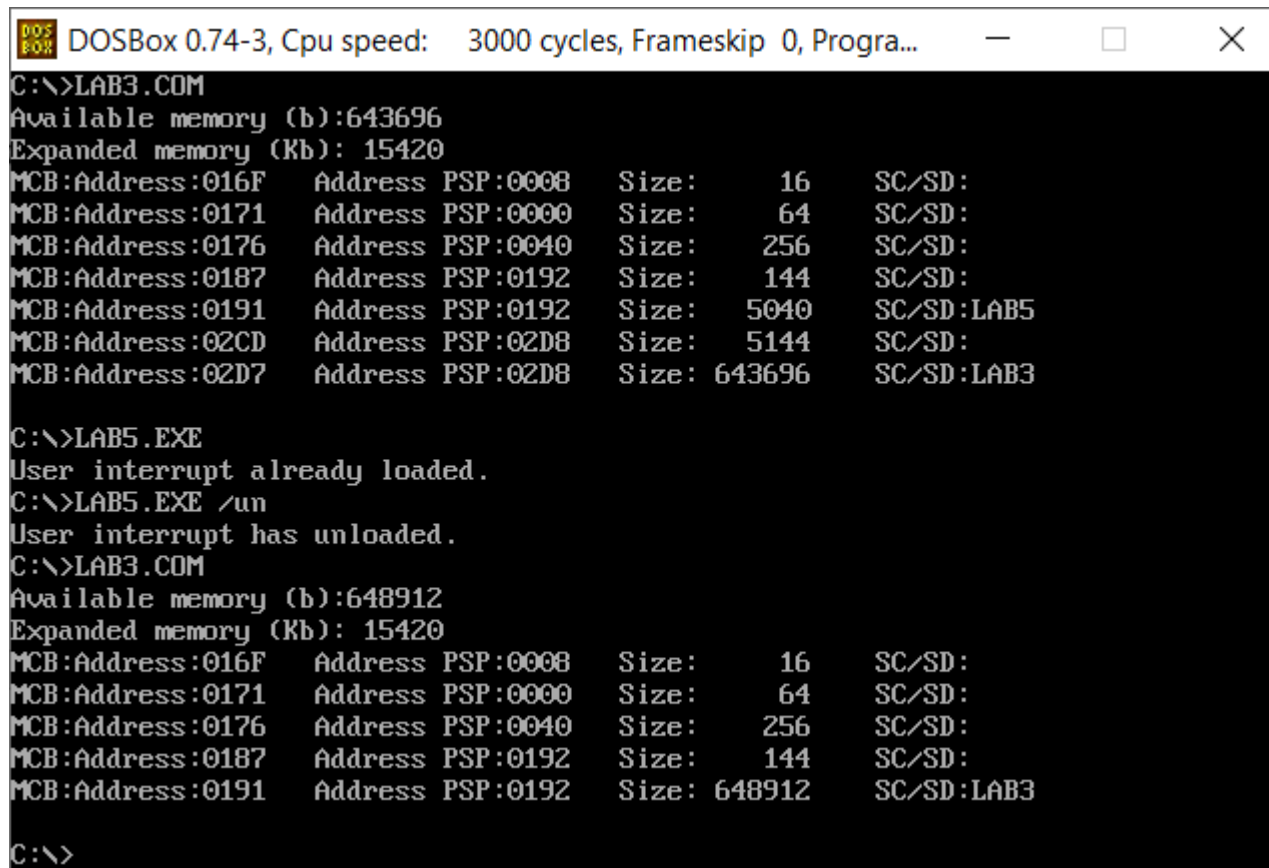
Рис. 2: Проверка размещения резидентного обработчика в памяти.

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

```
C:\>LAB5.EXE
User interrupt already loaded.
```

Рис. 3: Проверка на то, что программа определяет установленный обработчик прерываний.

Шаг 5. Программа была запущена с ключом выгрузки /un для того, чтобы убедиться, что резидентный обработчик прерывания выгружен и память, занятая резидентом, освобождена.



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
C:\>LAB3.COM
Available memory (b):643696
Expanded memory (Kb): 15420
MCB:Address:016F   Address PSP:0008   Size:    16   SC/SD:
MCB:Address:0171   Address PSP:0000   Size:    64   SC/SD:
MCB:Address:0176   Address PSP:0040   Size:   256   SC/SD:
MCB:Address:0187   Address PSP:0192   Size:   144   SC/SD:
MCB:Address:0191   Address PSP:0192   Size:  5040   SC/SD:LAB5
MCB:Address:02CD   Address PSP:02D8   Size:   5144   SC/SD:
MCB:Address:02D7   Address PSP:02D8   Size: 643696   SC/SD:LAB3

C:\>LAB5.EXE
User interrupt already loaded.
C:\>LAB5.EXE /un
User interrupt has unloaded.
C:\>LAB3.COM
Available memory (b):648912
Expanded memory (Kb): 15420
MCB:Address:016F   Address PSP:0008   Size:    16   SC/SD:
MCB:Address:0171   Address PSP:0000   Size:    64   SC/SD:
MCB:Address:0176   Address PSP:0040   Size:   256   SC/SD:
MCB:Address:0187   Address PSP:0192   Size:   144   SC/SD:
MCB:Address:0191   Address PSP:0192   Size: 648912   SC/SD:LAB3

C:\>
```

Рис. 4: Демонстрация того, что выгрузка резидентного обработчика прерываний происходит успешно.

Ответы на контрольные вопросы

1. Какого типа прерывания использовались в работе?

Использовались следующие прерывания:

- 09h и 16h – аппаратное прерывание
- 21h – программное прерывание.

2. Чем отличается скан-код от кода ASCII?

Скан-код – это уникальное число, однозначно определяющее нажатую клавишу, в то время как ASCII – это код символа из таблицы ASCII.

Выводы.

Исследованы возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Lab5.asm:

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

DATA SEGMENT
    IS_LOAD DB 0
    IS_UNLOAD DB 0
    STRING_LOAD DB "USER INTERRUPT HAS LOADED.$"
    STRING_LOADED DB "USER INTERRUPT ALREADY LOADED.$"
    STRING_UNLOAD DB "USER INTERRUPT HAS UNLOADED.$"
    STRING_NOT_LOADED DB "USER INTERRUPT IS NOT LOADED.$"
DATA ENDS

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

WRITESTRING PROC NEAR
    PUSH AX
    MOV AH, 09H
    INT 21H
    POP AX
    RET
WRITESTRING ENDP

INTERRUPT PROC FAR
    JMP INTER_START

KEEP_DATA:
    KEEP_IP DW 0
    KEEP_CS DW 0
    KEEP_PSP DW 0
    KEEP_AX DW 0
    KEEP_SS DW 0
    KEEP_SP DW 0
    INTERRUPT_STACK DW 256 DUP(0)
    KEY DB 0
```

```

SIGN DW 1234H

INTER_START:
    MOV KEEP_AX, AX
    MOV KEEP_SP, SP
    MOV KEEP_SS, SS
    MOV AX, SEG INTERRUPT_STACK
    MOV SS, AX
    MOV AX, OFFSET INTERRUPT_STACK
    ADD AX, 256
    MOV SP, AX

    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH SI
    PUSH ES
    PUSH DS

    MOV AX, SEG KEY
    MOV DS, AX

    IN AL, 60H
    CMP AL, 20H
    JE KEY_D
    CMP AL, 11H
    JE KEY_W
    CMP AL, 23H
    JE KEY_H

    PUSHF
    CALL DWORD PTR CS:KEEP_IP
    JMP INTER_END

KEY_D:
    MOV KEY, '!'
    JMP NEXT

KEY_W:
    MOV KEY, '$'
    JMP NEXT

KEY_H:

```



```

        MOV KEY, '#'

NEXT:
        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
        MOV CH, 00H
        INT 16H
        OR AL, AL
        JZ INTER_END
        MOV AX, 0040H
        MOV ES, AX
        MOV AX, ES:[1AH]
        MOV ES:[1CH], AX
        JMP PRINT_KEY

INTER_END:
        POP DS
        POP ES
        POP SI
        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

```

INTERRUPT ENDP

END_ITTER:

CHECK_LOAD PROC NEAR

PUSH AX

PUSH BX

PUSH SI

MOV AH, 35H

MOV AL, 09H

INT 21H

MOV SI, OFFSET SIGN

SUB SI, OFFSET INTERRUPT

MOV AX, ES:[BX + SI]

CMP AX, SIGN

JNE END_LOAD

MOV IS_LOAD, 1

END_LOAD:

POP SI

POP BX

POP AX

RET

CHECK_LOAD ENDP

CHECK_UNLOAD PROC NEAR

PUSH AX

PUSH ES

MOV AX, KEEP_PSP

MOV ES, AX

CMP BYTE PTR ES:[82H], '/'

JNE END_CHECK

CMP BYTE PTR ES:[83H], 'U'

JNE END_CHECK

CMP BYTE PTR ES:[84H], 'N'

JNE END_CHECK

```

        MOV IS_UNLOAD, 1

END_CHECK:
        POP ES
        POP AX

        RET

CHECK_UNLOAD ENDP


INTERRUPT_LOAD PROC NEAR
        PUSH AX
        PUSH BX
        PUSH CX
        PUSH DX
        PUSH DS
        PUSH ES

        MOV AH, 35H
        MOV AL, 09H
        INT 21H
        MOV KEEP_CS, ES
        MOV KEEP_IP, BX
        MOV AX, SEG INTERRUPT
        MOV DX, OFFSET INTERRUPT
        MOV DS, AX
        MOV AH, 25H
        MOV AL, 09H

        INT 21H

        POP DS

        MOV DX, OFFSET END_ITTER
        MOV CL, 4H
        SHR DX, CL
        ADD DX, 10FH
        INC DX
        XOR AX, AX
        MOV AH, 31H
        INT 21H

```

```

    POP ES
    POP DX
    POP CX
    POP BX
    POP AX

    RET
INTERRUPT_LOAD ENDP

INTERRUPT_UNLOAD PROC NEAR
    CLI
    PUSH AX
    PUSH BX
    PUSH DX
    PUSH DS
    PUSH ES
    PUSH SI

    MOV AH, 35H
    MOV AL, 09H
    INT 21H
    MOV SI, OFFSET KEEP_IP
    SUB SI, OFFSET INTERRUPT
    MOV DX, ES:[BX+SI]
    MOV AX, ES:[BX+SI+2]

    PUSH DS
    MOV DS, AX
    MOV AH, 25H
    MOV AL, 09H
    INT 21H
    POP DS

    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

    STI

    POP SI
    POP ES
    POP DS
    POP DX
    POP BX
    POP AX

    RET

INTERRUPT_UNLOAD ENDP

BEGIN PROC
    PUSH DS
    XOR AX, AX
    PUSH AX

    MOV AX, DATA
    MOV DS, AX
    MOV KEEP_PSP, ES

    CALL CHECK_LOAD
    CALL CHECK_UNLOAD
    CMP IS_UNLOAD, 1
    JE UNLOAD
    MOV AL, IS_LOAD
    CMP AL, 1
    JNE LOAD
    MOV DX, OFFSET STRING_LOADED
    CALL WRITESTRING
    JMP FINISH

LOAD:
    MOV DX, OFFSET STRING_LOAD
    CALL WRITESTRING
    CALL INTERRUPT_LOAD

```

```

        JMP    FINISH

UNLOAD:
        CMP    IS_LOAD, 1
        JNE    NOT_LOADED
        MOV    DX, OFFSET STRING_UNLOAD
        CALL   WRITESTRING
        CALL   INTERRUPT_UNLOAD
        JMP    FINISH

NOT_LOADED:
        MOV    DX, OFFSET STRING_NOT_LOADED
        CALL   WRITESTRING

FINISH:
        XOR    AL, AL
        MOV    AH, 4CH
        INT    21H

BEGIN ENDP

CODE ENDS
END BEGIN

```