

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. 0382

Корсунов А.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

Постановка задачи

Цель работы.

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

Сведения о функциях и структурах данных управляющей программы.

Процедура	Описание
INTERRUPT	Функция прерывания
CHECK_UN	Проверка наличия параметров командной строки
WRITE_MESSAGE_WORD	Вывод строки на экран
IS_LOADED	Проверка загрузки пользовательского прерывания
LOAD	Загрузка обработчика прерываний
UNLOAD	Выгрузка обработчика прерываний
MAIN	Главная функция программы

Задание.

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет следующие функции:

- 1) Проверяет, установлено ли пользовательское прерывание с вектором 1Ch.
- 2) Устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено, и 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.
- 4) 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.

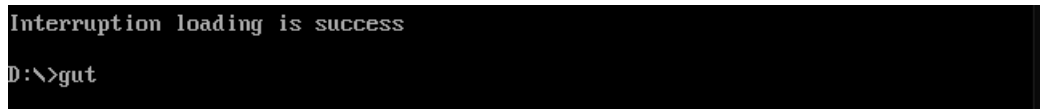
Шаг 5. Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

Шаг 6. Ответьте на контрольные вопросы.

Выполнение работы

Шаг 1. Для выполнения лабораторной работы был написан и отлажен программный модуль типа .EXE, который выполняет требуемые по заданию функции.

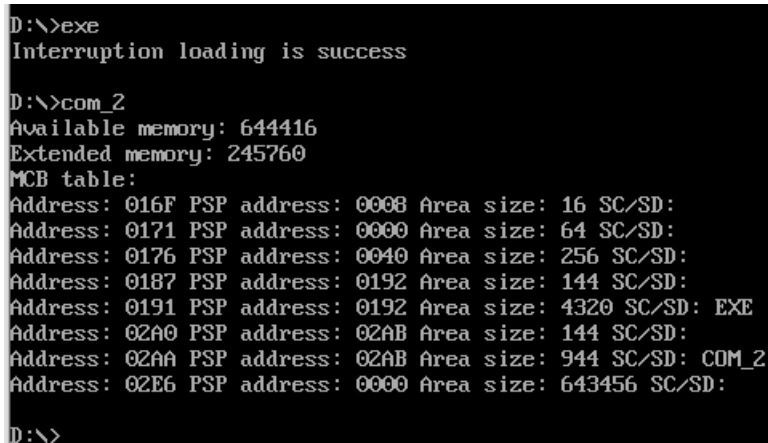
Шаг 2. Был запущена и отлажена программа.



```
D:\>gut
Interruption loading is success
```

Рисунок 1 — Иллюстрация работы программы (загрузка прерываний в память, в консоль вводятся символы ‘q’, ‘r’, ‘j’, которые заменяются на ‘g’, ‘u’, ‘t’)

Шаг 3. Была запущена программа из ЛР 3, которая проверяет размещение прерывания в памяти.



```
D:\>exe
Interruption loading is success

D:\>com_2
Available memory: 644416
Extended memory: 245760
MCB table:
Address: 016F PSP address: 0008 Area size: 16 SC/SD:
Address: 0171 PSP address: 0000 Area size: 64 SC/SD:
Address: 0176 PSP address: 0040 Area size: 256 SC/SD:
Address: 0187 PSP address: 0192 Area size: 144 SC/SD:
Address: 0191 PSP address: 0192 Area size: 4320 SC/SD: EXE
Address: 02A0 PSP address: 02AB Area size: 144 SC/SD:
Address: 02AA PSP address: 02AB Area size: 944 SC/SD: COM_2
Address: 02E6 PSP address: 0000 Area size: 643456 SC/SD:

D:\>
```

Рисунок 2 — Иллюстрация работы программы «com_2.com»

Шаг 4. Была запущена отлаженная программа еще раз.

```
D:\>exe
Interruption is loaded
D:\>_
```

Рисунок 3 — Иллюстрация повторного запуска программы (программа определила установленный обработчик прерываний)

Шаг 5. Была запущена отлаженная программа с ключом выгрузки.

```
D:\>exe /un
Interruption is restored

D:\>com_2
Available memory: 648912
Extended memory: 245760
MCB table:
Address: 016F PSP address: 0000 Area size: 16 SC/SD:
Address: 0171 PSP address: 0000 Area size: 64 SC/SD:
Address: 0176 PSP address: 0040 Area size: 256 SC/SD:
Address: 0187 PSP address: 0192 Area size: 144 SC/SD:
Address: 0191 PSP address: 0192 Area size: 944 SC/SD: COM_2
Address: 01CD PSP address: 0000 Area size: 647952 SC/SD: t*7e ä¹
D:\>_
```

Рисунок 4 — Иллюстрация работы программы с ключом выгрузки

Шаг 6. Были даны ответы на контрольные вопросы.

1) Какого типа прерывания использовались в работе?

Ответ. 09h, 16h – аппаратные прерывания, 10h, 21h – программные.

2) Чем отличается скан код от кода ASCII?

Ответ. Скан-код — код клавиши (число) клавиатуры, который обработчик прерываний клавиатуры переводит в код символа, ASCII – код символа из таблицы ASCII.

Вывод.

Было произведено исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

файл exe.asm:

MYSTACK SEGMENT STACK

DW 200 DUP(?)

MYSTACK ENDS

DATA SEGMENT

LOADED db 'Interruption is loaded', 0DH, 0AH, '\$'

LOADED_YES db 'Interruption loading is success', 0DH, 0AH, '\$'

LOADED_NO db 'Interruption loading is not success', 0DH, 0AH, '\$'

LOADED_RESTORED db 'Interruption is restored', 0DH, 0AH, '\$'

DATA ENDS

CODE SEGMENT

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

WRITE_MESSAGE_WORD PROC near

push AX

mov AH, 9

int 21h

pop AX

ret

WRITE_MESSAGE_WORD ENDP

INTERRUPT PROC far ;обработчик прерываний

jmp begin

sign dw 7777h
keep_IP dw 0 ; для хранения сегмента
keep_CS dw 0 ; и смещения прерывания
psp_address dw ?
keep_SS dw 0
keep_SP dw 0
keep_AX dw 0
my_stack dw 16 dup(?)
key DB 0

begin:

mov keep_SP, SP
mov keep_AX, AX
mov keep_SS, SS

mov SP, offset begin
mov AX, seg my_stack
mov SS, AX

push AX ;сохранение изменяемых регистров
push CX
push DX
push ES
push SI

mov AX, seg key
mov DS, AX

in AL, 60h ;читать ключ

cmp AL, 10h ;это требуемый код?
je key_q ;да, активизировать обработку

cmp AL, 13h
je key_r

cmp AL, 24h
je key_j

pushf
call dword ptr CS:keep_IP
jmp fin

key_q:
mov key, 'g'
jmp next_step

key_r:
mov key, 'u'
jmp next_step

key_j:
mov key, 't'
jmp next_step

next_step:
in al,61h ;взять значение порта управления клавиатурой
mov ah,al ;сохранить его
or al,80h ;установить бит разрешения для клавиатуры
out 61h,al ;и вывести его в управляющий порт
xchg ah,al ;извлечь исходное значение порта
out 61h,al ;и записать его обратно

mov al,20h ; послать сигнал "конец прерывания"
out 20h,al ; контроллеру прерываний 8259

key_print:

mov ah,05h ; Код функции
mov cl, key ; Пишем символ в буфер клавиатуры
mov ch,00h ;
int 16h ;
or al,al ; проверка переполнения буфера
jz fin
mov AX, 0040h
mov ES, AX
mov AX, ES:[1AH]
mov ES:[1ch], AX
jmp key_print

fin:

pop SI
pop ES
pop DX
pop CX
pop AX

mov SP, keep_SP
mov SS, keep_SS
mov AX, keep_AX
mov AL, 20h
out 20h, al

iret

```

    INTERRUPT_last:
INTERRUPT ENDP

CHECK_UN PROC near ;kk
    push AX
    push BP

    mov CL, 0h
    mov BP, 81h
    mov AL, ES:[BP + 1]
    cmp AL, '/'
    jne finish

    mov AL, ES:[BP + 2]
    cmp AL, 'u'
    jne finish

    mov AL, ES:[BP + 3]
    cmp AL, 'n'
    jne finish

    mov CL, 1h

    finish:
        pop BP
        pop AX
    ret
CHECK_UN ENDP

```

IS_LOADED PROC near ;kk

push AX

push DX

push ES

push SI

mov CL, 0h

mov AH, 35h

mov AL, 09h ; с вектором 09h

int 21h

mov SI, offset sign

sub SI, offset INTERRUPT

mov DX, ES:[BX+SI]

cmp DX, sign

jne if_end

mov CL, 1h

if_end:

pop SI

pop ES

pop DX

pop AX

ret

IS_LOADED ENDP

LOAD PROC near

push AX

push CX

push DX

call IS_LOADED

cmp CL, 1h

je already_load

mov psp_address, ES

;загрузка обработчика прерывания

mov AH, 35h ; функция получения вектора

mov AL, 09h ; номер вектора

int 21h

mov keep_CS, ES ; запоминание сегмента

mov keep_IP, BX ; и смещения

push ES

push BX

push DS

;настройка прерывания

lea DX, INTERRUPT ; смещение для процедуры в DX

mov AX, seg INTERRUPT ; сегмент процедуры

mov DS, AX ; помещаем в DS

mov AH, 25h ; функция установки вектора

mov AL, 09h ; номер вектора

int 21h ;меняем прерывание

pop DS

pop BX

pop ES

```
mov DX, offset LOADED_YES
call WRITE_MESSAGE_WORD
lea DX, INTERRUPT_last
mov CL, 4h
shr DX, CL
inc DX
add DX, 100h
xor AX,AX
mov AH, 31h
int 21h
```

```
jmp end_load
```

```
already_load:
mov DX, offset LOADED
call WRITE_MESSAGE_WORD
```

```
end_load:
    pop DX
    pop CX
    pop AX
```

```
ret
```

```
LOAD ENDP
```

```
UNLOAD PROC near ;kk
```

```
push AX
```

```
push SI
```

call IS_LOADED

cmp CL, 1h

jne not_load

;при выгрузке обработчика прерывания

cli

push DS

push ES

mov AH, 35h

mov AL, 09h

int 21h ; восстанавливаем вектор

mov SI, offset keep_IP

sub SI, offset INTERRUPT

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 LOADED_RESTORED

call WRITE_MESSAGE_WORD

jmp end_unload

not_load:

mov DX, offset LOADED_NO

call WRITE_MESSAGE_WORD

end_unload:

pop SI

pop AX

ret

UNLOAD ENDP

MAIN PROC far

sub AX, AX

mov AX, DATA

mov DS, AX

call CHECK_UN ;проверка на /un

cmp CL, 0h

```
jne un  
call LOAD ;загрузить  
jmp end_main
```

```
un:  
call UNLOAD ;выгрузить
```

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

```
CODE ENDS
```

```
END MAIN
```