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 _____ Бочаров Г.С.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2022

Постановка задачи

Цель работы.

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

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

Функции, написанные в ходе выполнения:

1. *print_word* — выводит строку, из регистра dx
2. *interrupt* - обработчик прерывания клавиатуры.
3. *load_interrupt* — загружает функцию обработчика прерывания в память и оставляет ее резидентной.
4. unload_interrupt— восстанавливает исходный обработчик прерывания и освобождает память.
5. *check_cmd_tail* — проверяет хвост командной строки.

check_interrupt— проверяет, загружено ли прерывание в память.

Шаг 1. Для выполнения лабораторной работы был написан и отлажен программный модуль типа .EXE, который выполняет поставленную задачу по обработке прерывания

Шаг 2. Был запущена и отлажена программа.

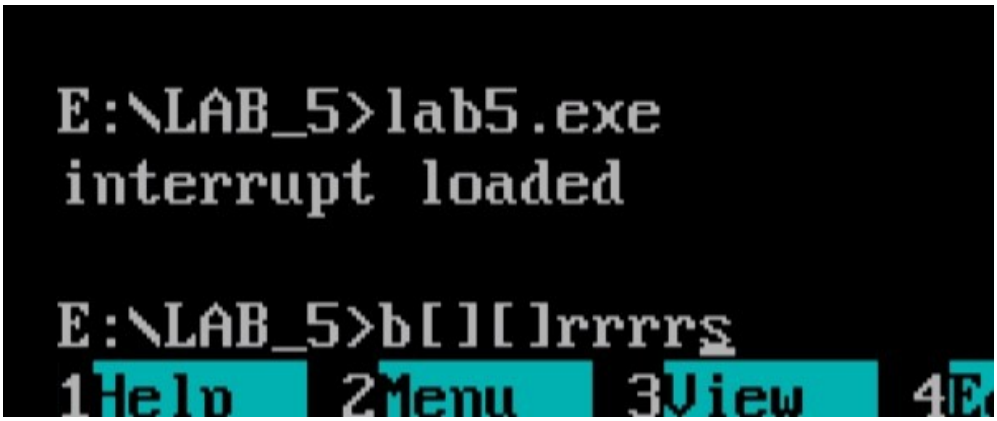

Рисунок 1 — Иллюстрация работы программы (загрузка прерывания в память, в консоль водятся символ ‘a’ замена на ‘b’, символ ‘[’ - дополняется символом ‘]’, ‘r’ - дублируется)

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

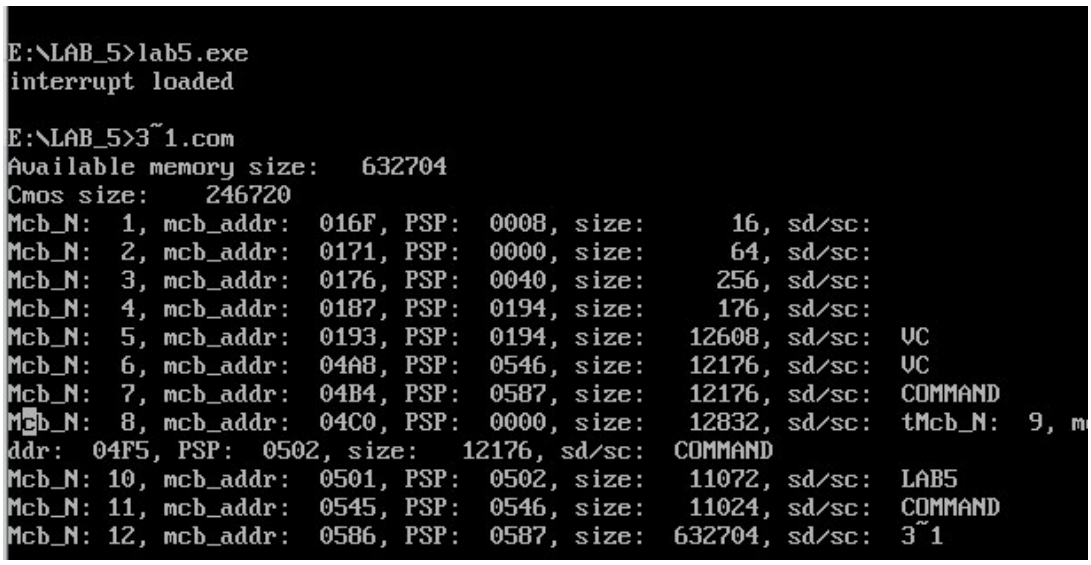

Рисунок 2 — Иллюстрация работы программы «3.1.com» из ЛБ_3

Шаг 4. Программа была запущена еще раз.

Рисунок 3 — Иллюстрация повторного запуска программы (программа выводит сообщение, что обработчик прерывания уже загружен)

Шаг 5. Была запущена отлаженная программа с ключом выгрузки.

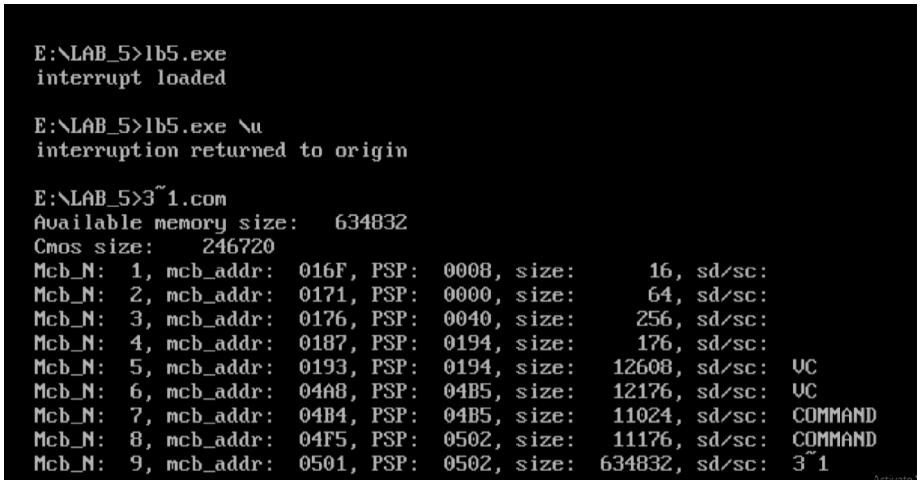

Рисунок 4 — Иллюстрация работы программы с ключом выгрузки. (Память занимаемая обработчиком освобождена)

Шаг 6. 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:

```
assume cs:code, ds:data, ss:stack

stack segment stack
dw 128 dup(?)
stack ends

code segment

word_to_dec proc near    ; input ax !, output ds:si
    push cx
    push dx
    push ax
    xor dx,dx
    mov cx,10
loop_bd:
    div cx
    or dl,30h
    mov [si],dl
    dec si
    xor dx,dx
    cmp ax,10
    jae loop_bd
    cmp al,00h
    je end_l
    or al,30h
    mov [si],al
end_l:
    pop ax
    pop dx
    pop cx
    ret
word_to_dec endp
```

```
get_curs proc near

push ax
push bx

mov ah, 03h
mov bh, 0
int 10h

pop bx
pop ax

ret
get_curs endp


set_curs proc near

push ax
push bx
mov ah, 02h
mov bh, 0
int 10h

pop bx
pop ax

ret
set_curs endp


outputbp proc  ;es:bp
push ax
push bx
push dx
push cx
mov ah,13h ; ôóíêöèÿ
mov al,1 ; sub function code
```

```
mov bh,0 ; âèäåî ñòðàíèöà
;mov dh,22 ; dh,dl = ñòðîêà, êîëîíêà (ñ÷èòàÿ îò 0)
;mov dl,0
int 10h
pop cx
pop dx
pop bx
pop ax
 ret
outputbp endp


interrupt proc far
jmp start_int

interrupt_data:

      interrupt_id  DW 0c204h
      keep_ip DW 0
      keep_cs DW 0
      keep_psp DW 0
      keep_ax DW 0
      keep_ss DW 0
      keep_sp DW 0
      new_stack DW 128 DUP(0)
      char DB 0

    start_int:
   ; save
      mov keep_sp, sp
      mov keep_ax, ax
      mov keep_ss, ss
      ; stack
      mov sp, offset start_int
      mov ax, seg new_stack
      mov ss, ax


        push ax
```

```
    push bx
      push cx
    push dx
    push si
    push es
    push ds

mov ax, seg char
mov ds, ax




  ; reset keyboard state
  push ax
  push ds
  xor     ax,ax
    mov     ds,ax
  mov     byte ptr ds:0417h, al
  pop ds
  pop ax



in al, 60h   ;ñ÷èòûâàíèå íîìåðà êëàâèøè


    cmp al, 9Ah    ;   close [
    je close_braket


    cmp al, 1eh  ; a->b
    je change_a_to_b


     ; r -> rr
    cmp al, 93h
    je double_r
```

```
    pushf

    call dword ptr cs:keep_ip

    jmp end_ll

    change_a_to_b:
           mov char, 'b'
      jmp hardware_interrupt

    close_braket:
      mov char, ']'
         jmp hardware_interrupt

    double_r:
        mov char, 'r'
    ;hardware interrupt
hardware_interrupt:
      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 to buffer
print_char:
      mov ah, 05h
      mov cl, char
      mov ch, 00h
      int 16h
      or al, al
      jz end_ll
clear_buffer:
      mov ax, 0040h
      mov es, ax
      mov ax, es:[1ah]
      mov es:[1ch], ax
```

```
      jmp print_char

    end_ll:
      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
my_int_end :
interrupt endp



load_interrupt proc near
push ax
push bx
push dx
push es

mov ah, 35h
mov al, 09h
int 21h
mov keep_ip, bx
mov keep_cs, es      ; old int

push ds
    mov dx, offset interrupt
      mov ax, seg interrupt
```

```
    mov ds, ax
    mov ah, 25h
      mov al, 09h
    int 21h
pop ds


mov dx, offset interrupt_successfully_loaded
call print_word

mov dx, offset my_int_end ; mk resident
mov cl,4
shr dx,cl
inc dx
mov ax, cs
    sub ax, keep_psp
    add dx, ax
    xor ax, ax
mov ah,31h
int 21h                     ; exit dos



pop es
pop dx
pop bx
pop ax
ret
load_interrupt endp


unload_interrupt proc near
push ax
push bx
push dx
push es

mov ah, 35h
mov al, 09h
```

```
int 21h; es:bx - int adr

; restre old int
cli
push ds
mov dx, es:[keep_ip]
mov ax, es:[keep_cs]
mov ds, ax
mov ah, 25h
mov al, 09h
int 21h

pop ds
sti

mov dx, offset returned_original_interrupt
call print_word

; mem free
mov ax, es:[keep_psp]
mov es, ax
push es
mov ax, es:[2ch]
mov es, ax
mov ah, 49h
int 21h
pop es
int 21h

pop es
pop dx
pop bx
pop ax

ret
unload_interrupt endp
```

```
check_interrupt proc near ; al  0 no, 1 - yes
push bx
push dx
push es

mov ah, 35h
mov al, 09h
int 21h

mov si, offset interrupt_id
sub si, offset interrupt
mov dx, es:[bx + si]
mov al, 0
cmp dx, 0c204h ; signature
jne fin_

int_set_:
mov al, 1

fin_:
pop es
pop dx
pop bx

ret
check_interrupt endp



check_cmd_tai proc near
; al  0 no, 1 - yes
push bx

mov al, 0
mov bh, es:[82h]   ; es:[81h] cmd tail
cmp bh, '\'
jne end_
mov bh, es:[83h]
cmp bh, 'u'
```

```
jne end_
mov al, 1

end_:

pop bx
ret

check_cmd_tai endp


print_word proc near
push ax
      mov ah, 09h
      int 21h
pop ax
      ret
print_word endp


main proc far

mov ax, data
mov ds, ax
mov keep_psp, es

call check_cmd_tai
cmp al, 1
je start_unload_int

call check_interrupt
cmp al, 1
jne start_load

mov dx, offset interrupt_already_loaded
call print_word
jmp endl

start_load:
```

```
    call load_interrupt

    start_unload_int:

    call check_interrupt
    cmp al, 0
    je interrupt_not_loaded_
    call unload_interrupt
    jmp endl

    interrupt_not_loaded_:
    mov dx, offset interrupt_not_loaded
    call print_word
    jmp endl

    endl:
    mov ah, 4ch
    int 21h

    main endp
    code ends

    data segment

    interrupt_successfully_loaded db      'interrupt loaded',
        0dh, 0ah, '$'
    interrupt_already_loaded db           'interruption         already
loaded',        0dh, 0ah, '$'
    returned_original_interrupt db        'interruption   returned   to
origin',   0dh, 0ah, '$'
    interrupt_not_loaded db               'interruption not loaded',
        0dh, 0ah, '$'
    data ends

    end main
```