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. 9383 _____ Крейсманн К.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 управления 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

Шаг1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет такие же функции, как в программе ЛР4, а именно:

1) Проверяет, установлено ли пользовательское прерывание с вектором 1Ch.

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.

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) Сохранять 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

- Был написан и отлажен программный модуль типа .EXE, который выполняет следующие функции:

1) Проверяет, установлено ли пользовательское прерывание с вектором 09h.

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 прерывания 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.

- Была запущена программа, работа прерывания отображается на экране. Выполнена проверка работы обработчика прерывания при вводе различных символов (при вводе Inc+k выводится ‘D’)(Рисунок 1).

Рисунок 1 - Демонстрация работы модуля

- Выполнена проверка размещения прерывания в памяти, для этого запущена программа lab3_1.com. (Рисунок 2)

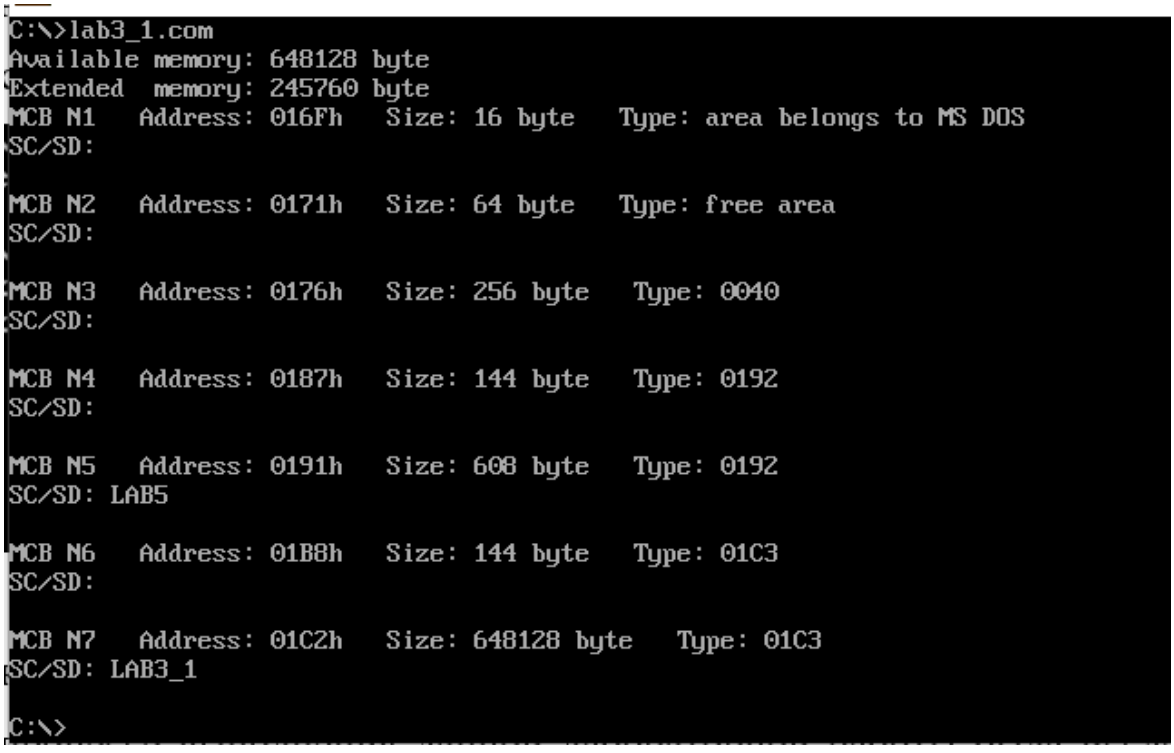

Рисунок 2 - Проверка размещения в памяти

- Выполнена проверка, что программа определяет установленный обработчик. (Рисунок 3)

Рисунок 3 - Проверка установленного обработчика

- Программа запущена с ключом выгрузки ‘/un’. Запущена программа lab3_1.com, чтобы убедиться, что память занятая резидентом освобождена. (Рисунок 4)

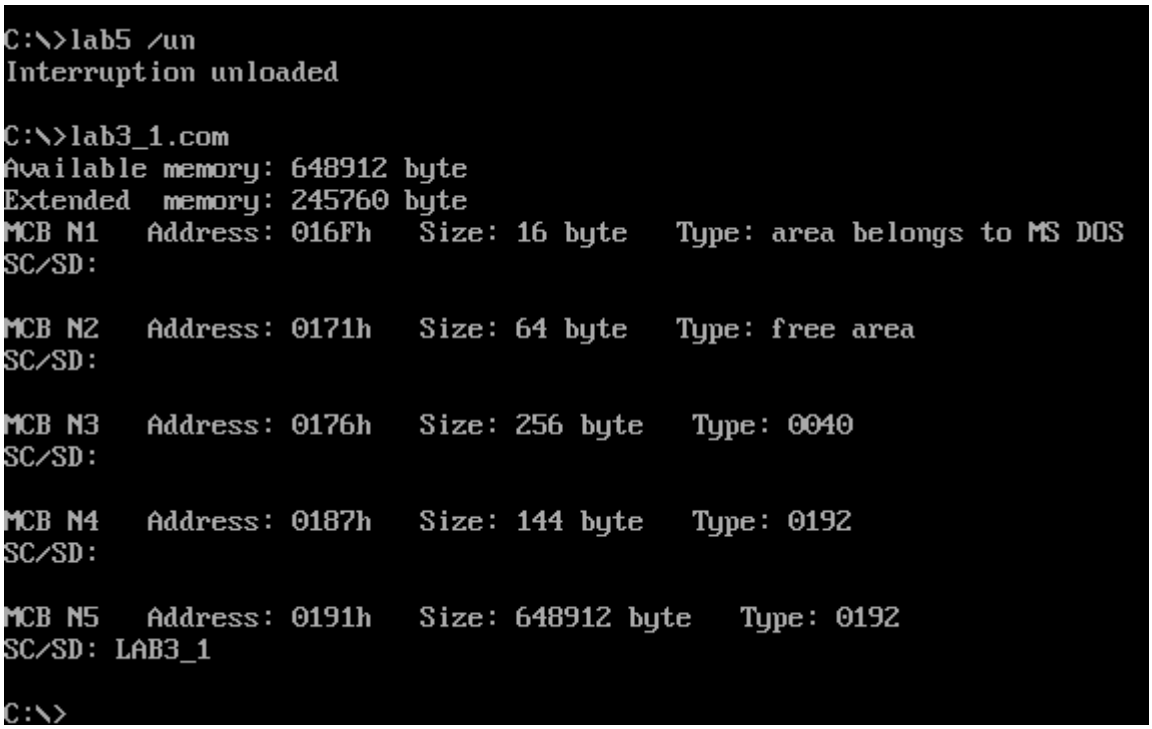

Рисунок 4 - Проверка выгрузки обработчика

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ ПО ЛАБОРАТОРНОЙ №5

1) Какого типа прерывания использовались в работе?

1. Аппаратные – обеспечивают реакцию процессора на события, происходящие асинхронно по отношению к исполняемому программному коду. (прерывания от контроллера клавиатуры)

2. Программные – вызываются с помощью команды int, для обращение к специальным функциям операционной системы. (21h, 16h).

2) Чем отличается скан-код от кода ASCII?

С помощью скан-кода определяется какая клавиша нажата на клавиатуре. ASCII-код – это кодировка символов.

Выводы.

Была исследована возможность встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

Приложение А.

Исходный код программы lab4.asm:

```
 dosseg
.model small
.stack 400h

.data
   mstack dw 100h dup(?)
   int_is_load dw ?
   cmd_line_flag dw 0
   str_is_not_load db "Interruption didn't load",0dh,0ah,'$'
   str_load     db "Interruption loaded",0dh,0ah,'$'
   str_unload   db "Interruption unloaded",0dh,0ah,'$'
   str_already_loaded db "Interruption is already loaded",0dh,0ah,'$'

.code
jmp m

WRITE_STR proc near
                push ax
                mov ah,9h
                int 21h
                pop ax
                ret
WRITE_STR ENDP


MY_INT PROC FAR
      jmp process
      _code dw 0abcdh
```

```
        keep_ip dw 0
        keep_cs dw 0
        temp_ss dw 0
        temp_sp dw 0
        PSP_0 dw 0
        PSP_1 dw 0
        keep_ax DW 0

process:
        cli
        mov keep_ax,ax
        mov temp_ss,ss
        mov temp_sp,sp
        mov ax,seg mstack
        mov ss,ax
        mov ax,offset mstack
        add ax,100H
        mov sp,ax
        sti

        push ax
        push bx
        push cx
        push dx

        in al,60h
        cmp al,25h ;скан-код - k?
        jnz STANDARD ;если нет то переходим к стандартному
        ;проверяем нажат ли Ins
        mov ax,0040h
```

```
        mov es,ax
        mov al,es:[18h]
        and al,10000000b
        je STANDARD ;если не нажат то переходим к стандартному

        ;следующий ход необходим для отработки аппаратного прерывания

        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 ;извлесь исходное значение порта
        out 61h,al ;и записать его обратно
        mov al,20h ;послать сигнал о конце прерывания
        out 20h,al ;контроллеру прерываний 8259


PUSH_SYMB:      ;запись символа в буфер клавиатуры
        mov ah,05h ;код функции
        mov cl,'D' ;пишем символ в буфер клавиатуры
        mov ch,00h
        int 16h
        or al,al ;проверка переполнения бефера
        jnz skip; если переполнен идем skip
        jmp end_p
skip:           ;очистить буфер и повторить
        cli
        mov ax,es:[1ah] ;адрес начала буфера
                mov es:[1ch],ax ;в адрес конца буфера
                sti
```

```
        jmp push_symb

STANDARD:

        pop dx
        pop cx
        pop bx
        pop ax
        mov ax,keep_ax
        mov ss,temp_ss
        mov sp,temp_sp
        jmp dword ptr keep_ip

end_p:

        pop dx
        pop cx
        pop bx
        pop ax

        mov ax,keep_ax
        mov al,20h
            out 20h,al
        mov ss,temp_ss
        mov sp,temp_sp
        iret
MY_INT ENDP

empty_func proc
empty_func endp
```

```
is_LOAD PROC NEAR
        mov ah,35h
        mov al,09h
        int 21H

        mov dx,es:[bx+3]
        cmp dx,0abcdh
        je isLoad
        mov int_is_load,0
        jmp endOfIsLoad

isLoad:
        mov int_is_load,1

endofisload:
ret
is_LOAD endp


UNLOAD PROC NEAR
        call is_load
        cmp int_is_load,1
        jne metka1
        mov ah,35h
        mov al,09h
        int 21h;получаем вектор
        cli
        push ds
        mov dx,es:[bx+5]
```

```
        mov ax,es:[bx+7]
        mov ds,ax
        mov ah,25h
        mov al,09h
        int 21h ;восстанавливаем вектор
        pop ds
        sti

        mov dx,offset str_unload
        call write_str

        push es
        mov cx,es:[bx+13]
        mov es,cx
        mov ah,49h
        int 21h
        pop es
        mov cx,es:[bx+15]
        mov es,cx
        int 21h

        jmp metka2
metka1:
        mov dx,offset str_is_not_load
        call write_str
metka2:
    ret
UNLOAD ENDP

LOAD PROC NEAR
```

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

        push ds
        mov dx,offset MY_INT
        mov ax,seg MY_INT
        mov ds,ax
        mov ah,25h
        mov al,09h
        int 21h
        pop ds

ret
LOAD endp


CHECK_CMD_LINE PROC NEAR

        mov di,82h
        mov al,es:[di]
        cmp al,'/'
        jnz end_of_check
        inc di

        mov al,es:[di]
        cmp al,'u'
        jnz end_of_check
```

```
        inc di

        mov al,es:[di]
        cmp al,'n'
        jnz end_of_check

        call UNLOAD
        mov cmd_line_flag,1
        jmp ret_check
end_of_check:
        mov cmd_line_flag,0
ret_check:
    ret
CHECK_CMD_LINE ENDP



MAIN PROC FAR
m:
        mov bx,02ch
        mov ax,[bx]
        mov psp_0,ds
        mov psp_1,ax

        mov ax,@data
        mov ds,ax


        call CHECK_CMD_LINE
        cmp cmd_line_flag,0
```

```
        jne exit

        call is_load
        cmp int_is_load,1
        je alreadyLoaded

        call load
        mov dx,offset str_load
        call write_str

        ;оставим резидентной в памяти
        mov dx,offset empty_func
        mov cl,04h
        shr dx,cl ;в параграфы
        add dx,1bh
        mov ah,31h
        mov al,00h
        int 21h
        jmp exit

alreadyLoaded:
        mov dx,offset str_already_loaded
        call  write_str
exit:
        mov ah,4ch
        int 21h
MAIN ENDP
end
```