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. 9382 _____ Павлов Р.В.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

Постановка задачи.

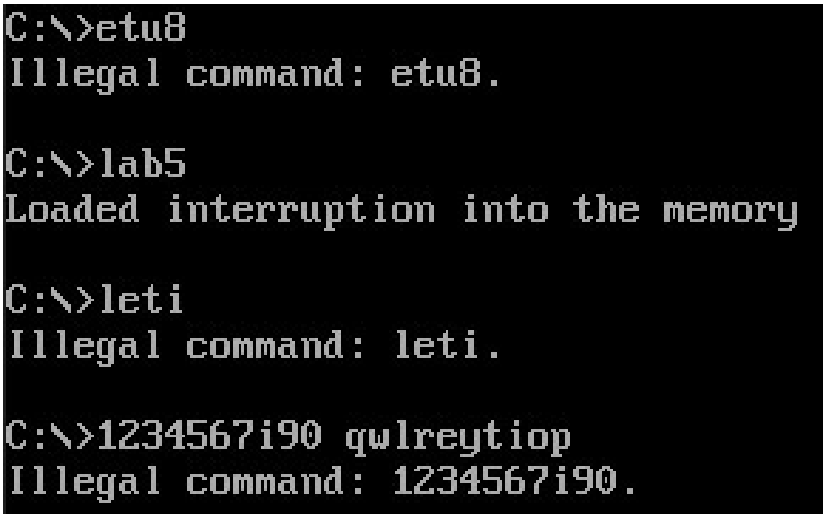
Цель работы: Исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Пользовательский обработчик прерывания получает управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре. Он обрабатывает скан-код и осуществляет определенные 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.

Функции и структуры данных:

Название процедуры	**Описание процедуры**
main	Вызов процедур загрузки и выгрузки, проверки на наличие пользовательского вектора
inter	Обработчик прерываний от клавиатуры
is_lodd	Сигнализирует о том, загружена ли функция или нет
is_tounld	Проверяет ввод пользователя на наличие флага выгрузки
load	Загружает обработчик прерываний в память
unload	Выгружает обработчик прерываний из памяти

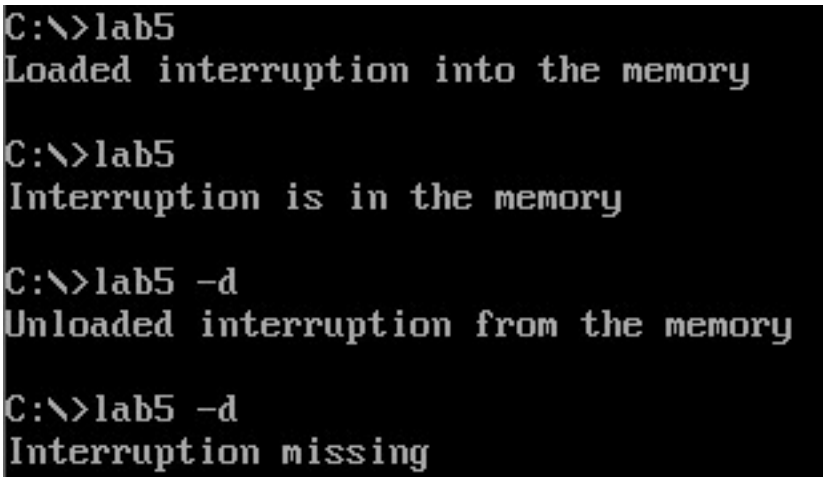
Ход работы.

1) Написан обработчик прерываний. Процедура получает сигнал от клавиатуры, в ответ на который устанавливает разрешающий бит и переходит к обработке символа (те символы, которые не рассматриваются, передаются для обработки стандартному обработчику прерываний). После этого обработанный символ заносится в буфер и выводится на экран.

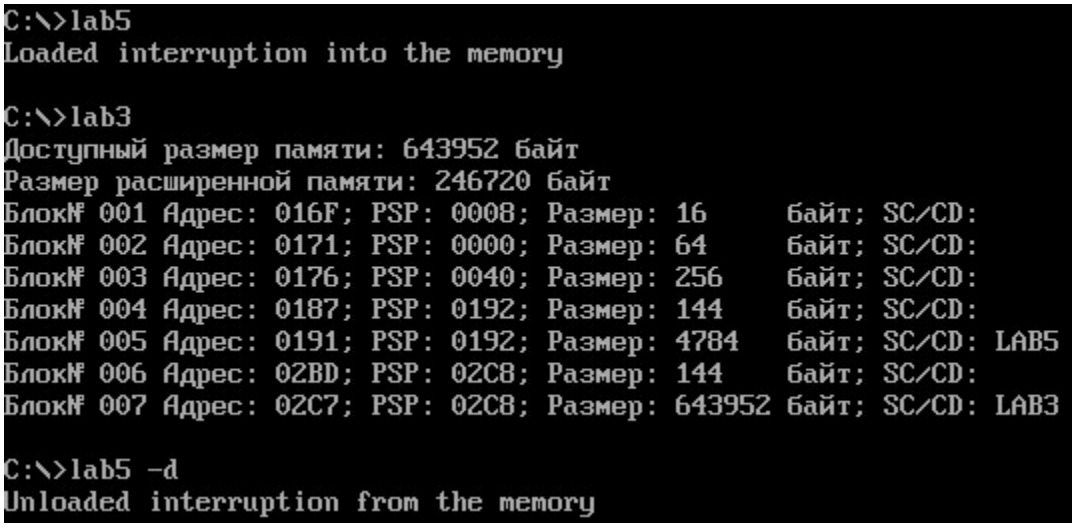


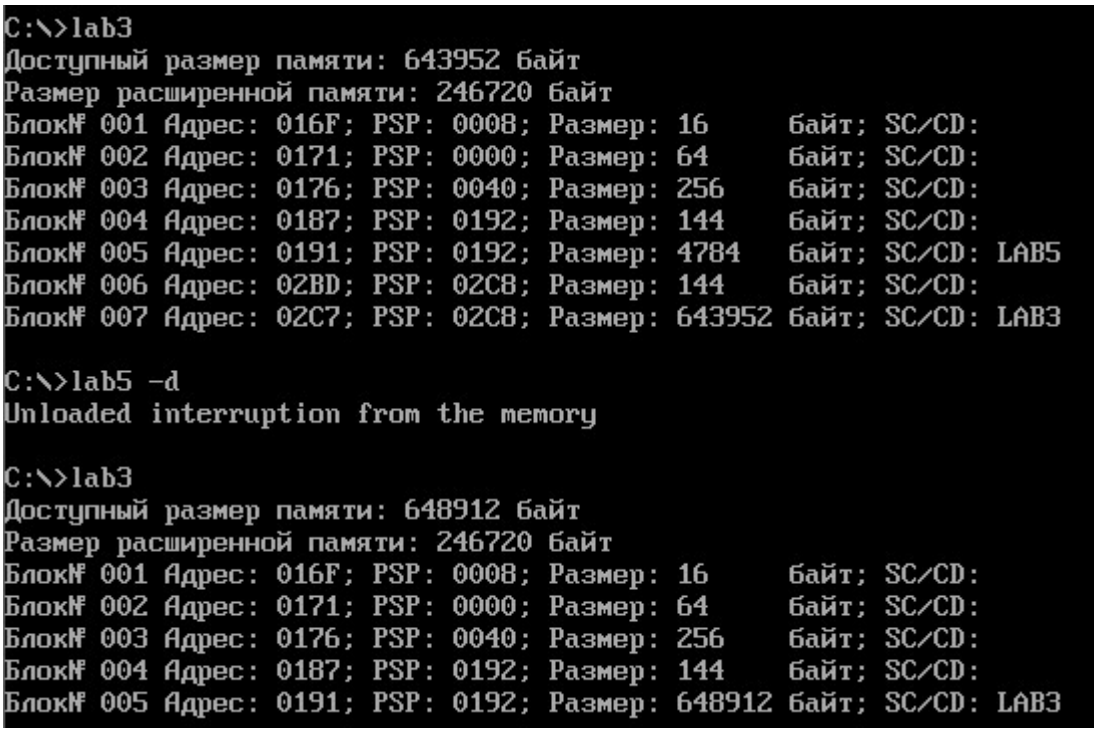
2) Реализованы процедуры загрузки обработчика в память и выгрузки из неё. При загрузке программе(сегменту) выделяется память, а вектор продолжает указывать на неё даже после завершения работы. При выгрузке исходный вектор восстанавливается, а память, используемая программой ранее, освобождается.

3) Реализованы процедуры проверки пользовательского ввода на предмет выбора загрузки или выгрузки.



4) В главной процедуре на основании ввода и текущего состояния обработчика принимается решение о выводе того или иного сообщения и предприятии какого-либо действия либо бездействии.





Ответы на контрольные вопросы.

Сегментный адрес недоступной памяти

1. Прерывания DOS – 21h и BIOS – 09h и 16h.
2. Коды ASCII используются для кодирования текста, а скан-коды – определённые идентификаторы клавиш на клавиатуре, которые распознаются драйвером.

Выводы.

В результате выполнения лабораторной работы реализована операция загрузки обработчика прерывания в память, при этом в случаях, когда обработать какой-либо сигнал нельзя, он передаётся стандартному обработчику прерываний.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД

- имя файла : lab5.asm

```
astack segment stack
      dw 64 dup(0)
astack ends

data segment
      loading_msg db 'Loaded interruption into the memory', 10, 13, '$'
      loaded db 'Interruption is in the memory', 10, 13, '$'
      unloading_msg db 'Unloaded interruption from the memory', 10, 13, '$'
      missing_msg db 'Interruption missing', 10, 13, '$'
      lodd db 0
      tounld db 0
data ends

code segment
assume cs:code, ds:data, ss:astack

inter proc far
    jmp init
            substack dw 128 dup(0)        ; Вложенный стек, хранилица для cs
и ip
            ip1 dw 0
            cs1 dw 0
            psp1 dw 0
            ax1 dw 0
            ss1 dw 0
            sp1 dw 0
            id dw 0a492h
            k db 0

    init:
            mov ax1, ax
            mov sp1, sp
            mov ss1, ss
            mov ax, seg substack
            mov ss, ax
            mov ax, offset substack       ; настройка стека
            add ax, 256
            mov sp, ax

            push ax
            push bx
            push cx
            push dx
            push si
        push es
        push ds

      in al, 60h
      cmp al, 12h ; сравнение скан-кодов
      je k1
      cmp al, 14h
      je k2
      cmp al, 16h
```

```
      je k3
      cmp al, 9h
      je k4

      pushf
      call dword ptr cs:ip1   ; передача управления стандартному обработчику,
если ни один не подошёл
      jmp exit1

      k1:
            mov k, 'l'
            jmp recall
      k2:
            mov k, 'e'              ; запись соответствующего символа
            jmp recall
      k3:
            mov k, 't'
            jmp recall
      k4:
            mov k, 'i'

      recall:
            in al, 61h
            mov ah, al
            or al, 80h
            out 61h, al             ; установка быита разрешения
            xchg al, al
            out 61h, al
            mov al, 20h
            out 20h, al

      to_buff:
            mov ah, 05h
            mov cl, k
            mov ch, 00h
            int 16h
            or al, al
            jz exit1
            mov ax, 0040h           ; запись символа в буфер
            mov es, ax
            mov ax, es:[1ah]
            mov es:[1ch], ax
            jmp to_buff

      exit1:
            pop ds
            pop es
            pop si
            pop dx
            pop cx
            pop bx
            pop   ax

            mov sp, sp1
            mov ax, ss1             ; восстановление исходных значений сег-
ментов программы
            mov ss, ax
            mov ax, ax1

            mov al, 20h
            out 20h, al
```

```
      iret
inter endp

seg_end:

is_lodd proc near
      push es
      push ax
      push bx
      push si

      mov ah, 35h
      mov al, 09h
      int 21h
      mov si, offset id
      sub si, offset inter    ; если id не соответствует заданному, значит,
стандартное прерывание
      mov ax, es:[bx + si]
      cmp   ax, id
      jne exit2
      mov lodd, 1

      exit2:
            pop si
            pop bx
            pop ax
            pop es
      retn
is_lodd endp

print_string proc near
      push    ax
      mov     ah, 09h
      int     21h
      pop     ax
    retn
print_string endp

load proc
      push ax
      push bx
      push cx
      push dx
      push es
      push ds

      mov ax, 3509h
      int 21h
      mov cs1, es
      mov ip1, bx
      mov ax, seg inter             ; получение вектор
      mov ds, ax
      mov dx, offset inter

      mov ax, 2509h
      int 21h
      pop   ds
      mov dx, offset seg_end        ; запись в него своего обработчика
      mov cl, 4
      shr dx, cl
      add   dx, 110h
```

```
      mov ax, 3100h                        ; программа остаётся в памяти
      int 21h

      pop es
      pop dx
      pop cx
      pop bx
      pop ax
      ret
load endp

unload proc
      cli
      push ax
      push bx
      push dx
      push ds
      push es
      push si

      mov ah, 35h
      mov al, 09h
      int 21h
      mov si, offset ip1
      sub si, offset inter
      mov dx, es:[bx + si]                 ; вернуть прежние значения cs и ip
вектора
      mov ax, es:[bx + si + 2]

      push ds
      mov ds, ax
      mov ah, 25h
      mov al, 09h                                ; записать его заново
      int 21h
      pop ds
      mov ax, es:[bx + si + 4]
      mov es, ax
      push es
      mov ax, es:[2ch]                     ; очистить память, ранее занимаемую
программой
      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
unload endp

is_tounld proc
      push ax
      push es
```

```
      mov ax, psp1
      mov es, ax
      cmp byte ptr es:[82h], '-'
      jne exit3
      cmp byte ptr es:[83h], 'd'          ; если в хвост консоли тэг "-d",
программа выгружается из памяти
      jne exit3
      mov tounld, 1

      exit3:
            pop es
            pop ax
            ret
is_tounld endp

main proc
      push ds
      xor ax, ax
      push ax
      mov ax, data
      mov ds, ax
      mov psp1, es

      call is_tounld
      call is_lodd

      cmp lodd, 1
      je ldd
      cmp tounld, 1
      je miss

      mov dx, offset loading_msg
      call print_string
      call load
      jmp _exit_

      miss:
            mov dx, offset missing_msg          ; главная процедура, ветвле-
ние в зависимости от ввода пользователя
            call print_string
            jmp _exit_
      ldd:
            cmp tounld, 1
            je unld
            mov dx, offset loaded
            call print_string
            jmp _exit_
      unld:
            mov dx, offset unloading_msg
            call print_string
            call unload
      _exit_:
            mov ax, 4c00h
            int 21h
      main endp
code    ends
```