**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. 7381 |  | Габов E. C. |
| Преподаватель |  | Ефремов М. А. |

Санкт-Петербург

2019

**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.

**Последовательность работы программы**

1. Проверяет, установлено ли пользовательское прерывание;
2. Устанавливает пользовательское прерывание и оставляет его резидентным в памяти;
3. Восстанавливает системное прерывание, удаляя пользовательское и высвобождая занимаемую им память.

Пользовательское прерывание выполняет следующее действие: при вводе цифр 0-9 с основной клавиатуры заменяет их символами верхнего регистра

**Описание функций**

|  |  |
| --- | --- |
| **Название** | **Назначение** |
| MY\_INT | Осуществляет обработку прерывания |
| OLD\_INT | Сохраняет сегмент и смещение системного прерывания |
| SET\_NEW\_INT | Устанавливает вместо системного пользовательское прерывание |
| LOAD\_MY\_INT | Оставляет прерывание резидентным в памяти |
| DELETE\_MY\_INT | Восстанавливает системное прерывание и освобождает память, занимаемую резидентом |
| PRINT | Вывод строки на экран |

**Ход работы**

Запустим программу и попробуем ввести цифры (рис. 1):



Рисунок 1

Проверим память, запустив 3 лабораторную работу (рис. 2):

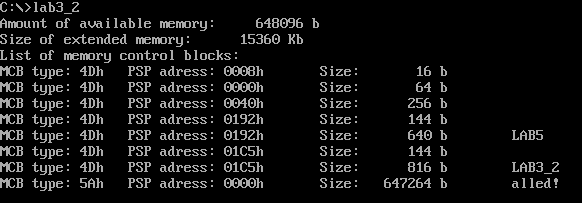


Рисунок 2

После завершения программы, область, установленная резидентной из памяти, не высвобождается.

При повторном вызове программы (рис. 3):



Рисунок 3

Выгрузим прерывание и попробуем ввести цифры (рис. 4):



Рисунок 4

Снова проверим память (рис. 5):

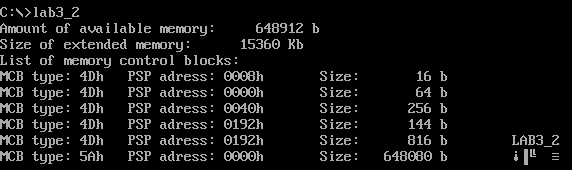


Рисунок 5

Область памяти, ране выделенная как резидентная, высвобождена.

**Ответы на контрольные вопросы:**

1) Какого типа прерывания использовались в работе?

В работе использовались программные (int 21h и int 16h) и аппаратные прерывания (int 09h)

2) Чем отличается скан-код от кода ASCII?

Скан-код – код, присвоенный каждой клавише клавиатуры, с помощью которого можно опознать, какая клавиша была нажата. Скан-коды жестко привязаны к каждой клавише.

ASCII код – код, используемый для представления символов в памяти компьютера (код может различаться в зависимости от таблицы кодировки или ее разновидности).

**Вывод**

В результате выполнения данной лабораторной работы были исследованы работа и возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры обработчиков прерываний.

Приложение А

1. Lab5.asm

code segment

assume cs:code, ds:data, ss:Tstack

start\_mem:

PSP dw 0

KEEP\_CS dw 0

KEEP\_IP dw 0

Int9 dd 0

Scancode\_1 db 2

Scancode\_2 db 3

Scancode\_3 db 4

Scancode\_4 db 5

Scancode\_5 db 6

Scancode\_6 db 7

Scancode\_7 db 8

Scancode\_8 db 9

Scancode\_9 db 10

Scancode\_0 db 11

KEEP\_AX dw 0

KEEP\_SS dw 0

KEEP\_SP dw 0

MY\_STACK dw 100 DUP(?)

end\_of\_my\_stack:

push\_main macro

mov CS:KEEP\_AX, AX

mov CS:KEEP\_SS, SS

mov CS:KEEP\_SP, SP

mov AX, SEG MY\_STACK

mov SS, AX

mov SP, offset end\_of\_my\_stack

push ax

push bx

push cx

push dx

endm

pop\_main macro

pop dx

pop cx

pop bx

pop ax

mov ax, CS:KEEP\_SS

mov ss, ax

mov SP, CS:KEEP\_SP

mov ax, CS:KEEP\_AX

endm

my\_int proc far

jmp body

Int\_Tag dw 1234h

body:

push\_main

in al, 60h

mov ch, 00h

cmp al, cs:[Scancode\_1]

jne next1

mov cl, '!'

jmp do\_req

next1:

cmp al, cs:[Scancode\_2]

jne next2

mov cl, '@'

jmp do\_req

next2:

cmp al, cs:[Scancode\_3]

jne next3

mov cl, '#'

jmp do\_req

next3:

cmp al, cs:[Scancode\_4]

jne next4

mov cl, '$'

jmp do\_req

next4:

cmp al, cs:[Scancode\_5]

jne next5

mov cl, '%'

jmp do\_req

next5:

cmp al, cs:[Scancode\_6]

jne next6

mov cl, '^'

jmp do\_req

next6:

cmp al, cs:[Scancode\_7]

jne next7

mov cl, '&'

jmp do\_req

next7:

cmp al, cs:[Scancode\_8]

jne next8

mov cl, '\*'

jmp do\_req

next8:

cmp al, cs:[Scancode\_9]

jne next9

mov cl, '('

jmp do\_req

next9:

cmp al, cs:[Scancode\_0]

jne int9do

mov cl, ')'

jmp do\_req

int9do:

pop\_main

jmp cs:[Int9]

do\_req:

in al, 61h

mov ah, al

or al, 80h

out 61h, al

xchg ah, al

out 61H, al

mov al, 20h

out 20h, al

mov ah, 05h

int 16h

or al, al

jnz skip

jmp to\_quit

;Aey i?enoee aooa?a iaai i?inoi onoaiiaeou cia?aiea

;y?aeee 0040:001A ?aaiui cia?aie? y?aeee 0040:001C.

skip:

push es

push si

mov ax, 0040h

mov es, ax

mov si, 001ah

mov ax, es:[si]

mov si, 001ch

mov es:[si], ax

pop si

pop es

to\_quit:

pop\_main

mov al, 20h

out 20h, al

iret

my\_int endp

end\_mem:

old\_int\_save proc near

push\_main

push es

push di

mov ah, 35h

mov al, 09h

int 21h

mov cs:KEEP\_IP, bx

mov cs:KEEP\_CS, es

mov word ptr Int9+2, es

mov word ptr Int9, bx

pop di

pop es

pop\_main

ret

old\_int\_save endp

set\_new\_int proc near

push\_main

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

pop\_main

ret

set\_new\_int endp

load\_my\_int proc near

mov dx, seg code

add dx, (end\_mem-start\_mem)

mov cl, 4

shr dx, cl ;div 16

inc dx

mov ah, 31h

int 21h

ret

load\_my\_int endp

delete\_my\_int proc near

cli

push\_main

push ds

push es

push di

mov ah,35h

mov al,09h

int 21h

mov ax, es:[2]

mov cs:KEEP\_CS, ax

mov ax, es:[4]

mov cs:KEEP\_IP, ax

mov ax, es:[0]

mov cx, ax

mov es, ax

mov ax, es:[2Ch]

mov es, ax

xor ax, ax

mov ah, 49h

int 21h

mov es, cx

xor ax, ax

mov ah, 49h

int 21h

mov dx, cs:KEEP\_IP

mov ax, cs:KEEP\_CS

mov ds, ax

mov ah, 25h

mov al, 09h

int 21h

pop di

pop es

pop ds

pop\_main

sti

ret

delete\_my\_int endp

main proc near

push ds

mov ax, seg data

mov ds, ax

pop cs:PSP

mov es, cs:PSP

mov al, es:[80h]

cmp al, 4

jne Empty\_Tail

mov al, byte PTR es:[82h]

cmp al, '/'

jne Empty\_Tail

mov al, byte PTR es:[83h]

cmp al, 'u'

jne Empty\_Tail

mov al, byte PTR es:[84h]

cmp al, 'n'

jne Empty\_Tail

mov IsDelete, 1

Empty\_Tail:

mov ah, 35h

mov al, 09h

int 21h

mov ax, es:[bx+3]

cmp ax, 1234h

je already\_inst

cmp IsDelete, 1

je not\_inst

call old\_int\_save

call set\_new\_int

call load\_my\_int

jmp exit

already\_inst:

cmp IsDelete, 1

je delete\_my\_int\_main\_m

mov dx, offset Inst\_Mess

mov ah, 09h

int 21h

jmp exit

delete\_my\_int\_main\_m:

call delete\_my\_int

jmp exit

not\_inst:

mov dx, offset Not\_Inst\_Mess

mov ah, 09h

int 21h

jmp exit

exit:

xor al, al

mov ah, 4Ch

int 21h

ret

main endp

code ends

data segment

IsDelete db 0

Inst\_Mess db 'Interrupt is already installed!', 10, 13, '$'

Not\_Inst\_Mess db 'Interrupt is not installed!', 10, 13, '$'

data ends

Tstack segment stack

dw 128 dup (?)

Tstack ends

end main