**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра MOЭBM**

отчет

**по лабораторной работе №5**

**по дисциплине «Операционные системы»**

**Тема: Сопряжение стандартного и пользовательского обработчиков прерываний**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6381 |  | Вергейчик Г. Л. |
| Преподаватель |  | Губкин А. Ф. |

Санкт-Петербург

2018

**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.

**Описание функций**

* ROUT - Функция обработчика прерывания.
* outputAL - Функция вывода символа из AL.
* outputBP - Функция вывода строки по адресу ES:BP на экран
* comprasionCode - Функция сравнения скан-кода с заданными.
* conversionCodeInSymbol - Функция преобразования скан-кода в символ.
* PRINT - Функция печати на экран.

**Описание данных**

* resUnload - Применяется для вывода информации о том, что резидент выгружен из памяти.
* resAlrLoad - Применяется для вывода информации о том, что резидент уже загружен в память.
* resLoad - Применяется для вывода информации о том, что резидент загружен в память.
* isLoad - Флаг указывающий на то, что программа должна быть загружена в память.
* isUnload - Флаг указывающий на то, что программа должна быть выгружена из памяти.
* KEEP\_IP - Применяется для запоминания смещения вектора прерывания.
* KEEP\_CS - Применяется для запоминания сегмента вектора прерывания.
* ID - Переменная предназначенная для проверки загружен ли резидент в память или нет.
* SCAN\_CODE - Массив предназначенный для хранения скан-кодов клавиш клавиатуры.

**Описание работы обработчика прерывания:**

Обработчик реагирует на скан-коды символов 1-0. При нажатии на клавиши 1-0 обработчик заменяет их на символы латинского алфавита a-j соответственно. Т.е пользователь вместо нажатой цифры увидит на экране символ латинского алфавита.

**Ход работы:**

**Шаг 1:**

Загрузка резидента в память(см. Рис1).

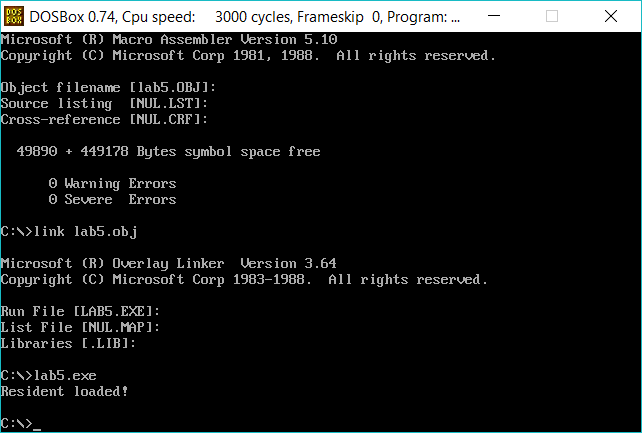


Рис. 1 Загрузка резидента в память

**Шаг 2:**

Проверка работы прерывания введением символов 1-5 (см. Рис 2-3).

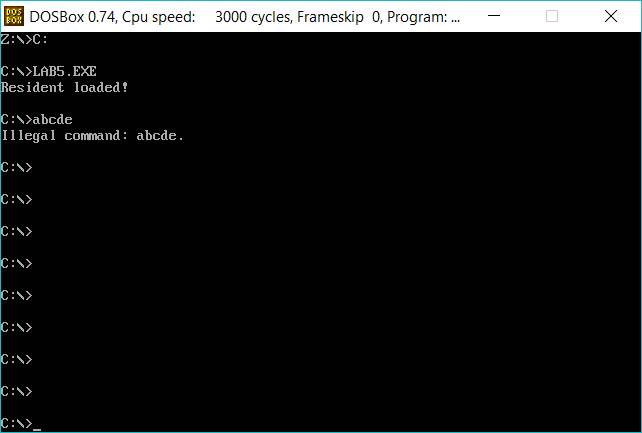


Рис. 2 Проверка работы прерывания введением символов

**Шаг 3:**

Запуск программы с ключом выгрузки /un (рис. 3):

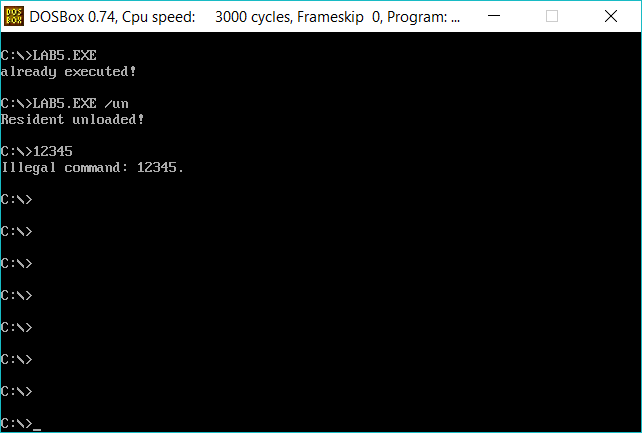


Рис. 3 Запуск программы с ключом выгрузки /un

**Шаг 4:**

Размещение прерывания в памяти(см. Рис4).

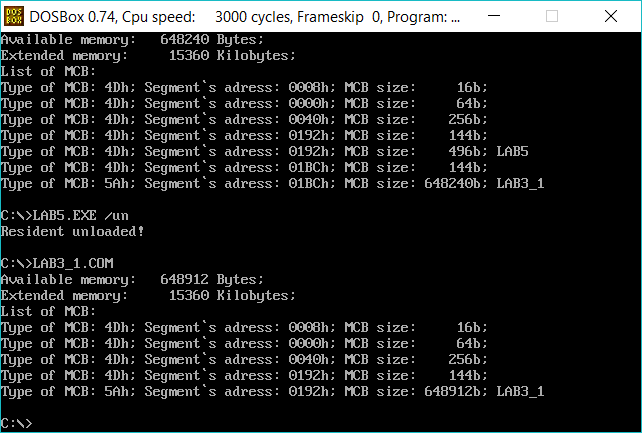


Рис. 4

**Ответы на контрольные вопросы:**

1. Какого типа прерывания использовались в работе?

Были использованы пользовательские прерывания, такие как int 10h и int 21h и аппаратные прерывания.

1. Чем отличается скан код от кода ASCII?

Код ASCII – это код символа из таблицы ASCII, а скан-код – код, который определяется нажатием клавиши или комбинации клавиш, который передаётся клавиатурой.

**Вывод**

В результате выполнения данной лабораторной работы были исследованы организация и управление прерываниями. Была разработана программа, в которой построен обработчик прерываний.

**Приложение**

1. Файл lab5.asm

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

CODE SEGMENT

;-----------------------------------------------------

;Прерывание

ROUT proc far

jmp no\_data

AD\_PSP dw ? ;3 байта

SR\_PSP dw ? ;5 байт

KEEP\_CS dw ? ;7 байт

KEEP\_ID dw ? ;9 байт

ID dw 0AAAAh ;11 байт

ADR dw 0506h

scan\_code db 2h, 3h, 4h, 5h, 6h, 7h, 8h, 9h, 0Ah, 0Bh, 82h, 83h, 84h, 85h, 86h, 87h, 88h, 89h, 8Ah, 8Bh, 00h

KEEP\_SS dw ?

KEEP\_SP dw ?

NEWTABLE db ' abcdefghij'

no\_data:

mov KEEP\_SS,ss

mov KEEP\_SP,sp

mov di, cs

mov ss, di

mov sp, offset STACK\_END

;Сохранение изменяемых регистров

push ax

mov ah,02h

int 16h

test al, 2;сравниваем со скан кодом Shift

jne end\_compare

;Считывание номера клавиши

in al, 60h

;Проверка на требуемые скан-коды

push ds

push ax

mov ax, SEG scan\_code

mov ds, ax

pop ax

mov dx, offset scan\_code

;dx - смещение символов, al - сам символ

push bx

push cx

mov bx, dx

sub ah, ah

;Сравнение кодов

for\_compare:

mov cl, byte ptr [bx]

cmp cl, 0h

je end\_compare

cmp al, cl

jne no\_equally

;Совпадает

mov ah, 01h

no\_equally:

;Не совпадает

inc bx

jmp for\_compare

end\_compare:

pop cx

pop bx

pop ds

cmp ah, 01h

je processing

jmp not\_processing

not\_processing:

;Возврат к стандартному обработчику прерывания

pop ax

mov ss, KEEP\_SS

mov sp, KEEP\_SP

pushf

push KEEP\_CS

push KEEP\_ID

iret

processing:

;Обработка прерывания

push bx

push cx

push dx

cmp al,80h

ja go

push es

push ds

push ax

mov ax, seg NEWTABLE

mov ds, ax

mov bx, offset NEWTABLE

pop ax

xlatb

pop ds

write\_to\_buffer:

;Запись в буфер клавиатуры

mov ah, 05h

mov cl, al

sub ch, ch

int 16h

or al, al

jnz cleaning

pop es

go:

jmp @ret

;Очистка буфера и повторение

cleaning:

push ax

mov ax, 40h

mov es, ax

mov word ptr es:[1Ah], 001Eh

mov word ptr es:[1Ch], 001Eh

pop ax

jmp write\_to\_buffer

@ret:

;Отработка аппаратного прерывания

in al, 61h

mov ah, al

or al, 80h

out 61h, al

xchg ah, al

out 61h, al

mov al, 20h

out 20h, al

;Востановление регистров

pop dx

pop cx

pop bx

pop ax

mov ss, KEEP\_SS

mov sp, KEEP\_SP

iret

ROUT endp

;tsr\_end:

NEW\_STACK dw 64 dup(?) ; стек

STACK\_END:

TEMP:

;ORG tsr\_end

;-----------------------------------------------------

; КОД

BEGIN PROC FAR

mov bx,02Ch

mov ax,[bx]

mov SR\_PSP,ax

mov AD\_PSP,ds

sub ax,ax

xor bx,bx

mov ax, DATA

mov ds, ax

;Проверка на загрузку резидента в память

push es

mov ah, 35h

mov al, 09h

int 21h

mov dx, es:[bx+11]

pop es

cmp dx, ID

jne no\_load

;Резидент загружен

mov isLoad, 1

jmp load

no\_load:

;Резидент не загружен

mov isLoad, 0

load:

;Проверка на выгрузку резидента из памяти

push es

push ax

mov ax, es

mov es, ax

mov cl, es:[80h]

mov dl, cl

sub ch, ch

test cl, cl

jz no\_unload

;Проверка ключа

mov al, es:[82h]

cmp al, '/'

jne no\_unload

mov al, es:[83h]

cmp al, 'u'

jne no\_unload

mov al, es:[84h]

cmp al, 'n'

jne no\_unload

mov isUnload, 1

jmp unload

no\_unload:

mov isUnload, 0

unload:

pop ax

pop es

cmp isLoad, 1

je already\_load

cmp isUnload,0

je nextStep

lea dx,resAlready

mov ah,09h

int 21h

jmp exit

nextStep:

;Если резидент не загружен, то загрузим его

; Загрузка обработчика прерывания

mov ah, 35h

mov al, 09h

int 21h

mov KEEP\_CS, es

mov KEEP\_ID, bx

;Установка прерывания

push ds

mov dx, offset ROUT ;Смещение функции

mov ax, seg ROUT ;Сегмент функции

mov ds, ax

mov ah, 25h ;Номер функции установки вектора

mov al, 09h ;Номер вектора

int 21h

pop ds

;Вывод сообщения о установке прерывания

lea dx, resLoad

mov ah, 09h

int 21h

;Оставление резидентной функции в памяти

mov dx, offset TEMP

mov cl, 4

shr dx, cl

add dx, 9h;

mov ah, 31h

int 21h

already\_load:

;Если резидент уже был загружен

cmp isUnload, 1

je unload\_int

; Не надо выгружать прерывание

lea dx, resAlready

mov ah, 09h

int 21h

jmp exit

unload\_int:

;Необходимо выгрузить прерывание

mov ax, 3509h ; функция получения вектора

int 21H

;Выгрузка обработчика прерывания

cli

push ds

mov dx,es:[bx+9] ;IP стандартного прерывания

mov ax,es:[bx+7] ;CS стандартного прерывания

mov ds, ax

mov ah, 25h

mov al, 09h

int 21h

pop ds

sti

;Вывод сообщения о выгрузке прерывания

lea dx, resUnload

mov ah, 09h

int 21h

;Очистка памяти из под прерывания

push es

mov cx,es:[bx+3]

mov es,cx

mov ah,49h

int 21h

pop es

mov cx,es:[bx+5]

mov es,cx

int 21h

exit:

; Выход в DOS

mov ah, 4ch

int 21h

BEGIN ENDP

;------------------------------------------------------------------

CODE ENDS

;------------------------------------------------------------------

DATA SEGMENT

;ДАННЫЕ

resLoad db 'Resident loaded!', 0DH, 0AH, '$'

resUnload db 'Resident unloaded!', 0DH, 0AH, '$'

resAlready db 'already executed!', 0DH, 0AH, '$'

isLoad db 0

isUnload db 0

DATA ENDS

SSTACK SEGMENT STACK

DW 64 DUP(?)

SSTACK ENDS

END BEGIN