**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 о**бработчиков прерываний**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9381 |  | Прибылов Н.А. |
| Преподаватель |  | Ефремов М.А. |

Санкт-Петербург

2021

## Цель работы.

Исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Пользовательский обработчик прерываний получает управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре. Он обрабатывает скан-код и осуществляет определенные действия, если скан-код совпадает с определёнными кодами, которые он должен обрабатывать. Если скан-код не совпадает с этими кодами, то управление передается стандартному прерыванию.

## **Ход работы**.

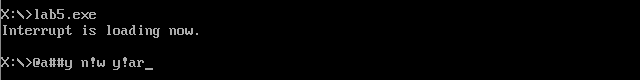
1) Написан и отлажен программный модуль типа .EXE, который выполняет такие же функции, как и в лабораторной работе №4, а именно:

• Проверяет, установлено ли пользовательское прерывание с вектором 09h.

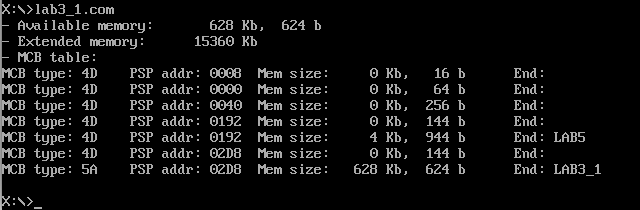
• Если прерывание не установлено, то устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний. Адрес точки входа в стандартный обработчик прерывания находится в теле пользовательского обработчика. Осуществляется выход по функции 4Ch прерывания int21h.

• Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h.

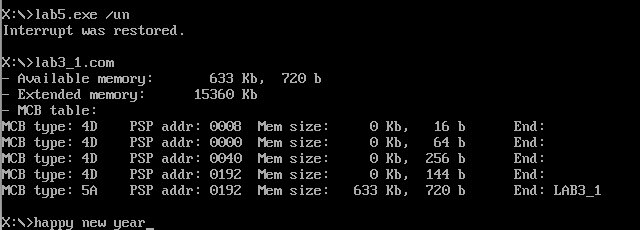
2. Была запущена отлаженная программа и проверено, что резидентный обработчик прерывания 09h установлен (заменяет символы, вводимые с клавиатуры: h → @, p → #, e → !).

Программа запущена

3. Было проверено размещение прерывания в памяти. Для этого запущена программа из лаб. работы №3, которая отображает карту памяти в виде списка блоков МСВ.

Резидент находится в памяти

4. Была запущена отлаженная программа с ключом выгрузки и проверено, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а занятая резидентом память освобождена. Для этого также была запущена программа лабораторной работы №3.

Резидент выгружен из памяти

**Контрольные вопросы.**

**1. Какого типа прерывания использовались в работе?**

Прерывания DOS (int 21h), прерывания BIOS (09h, 16h).

**2. Чем отличается скан код от кода ASCII?**

Скан-код – это код, который клавиатура передаёт системе. Тем самым система определяет, какая клавиша (или комбинация клавиш) была нажата. ASCII-код – это таблица кодировок для печатных символов.

Таким образом скан-код – это номер клавиши, а ASCII-код – код, соответствующий обозначению на этой клавише.

## Выводы.

В ходе данной работы производилось исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Было написано пользовательское прерывание от клавиатуры, которое анализирует скан-коды, выполняет вывод сообщения результата нажатия, а при несовпадении скан-кода передает управление стандартному обработчику.

# Приложение А Исходный код программы

Название файла: lab5.asm

CODE SEGMENT

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

STACK SEGMENT stack

dw 256 dup(0)

STACK ends

new\_interrupt PROC FAR

jmp Start

intData:

key\_value db 0

new\_stack dw 256 dup(0)

signature dw 2468h

keep\_ip dw 0

keep\_cs dw 0

keep\_psp dw 0

keep\_ax dw 0

keep\_ss dw 0

keep\_sp dw 0

Start:

mov keep\_ax, ax

mov keep\_sp, sp

mov keep\_ss, ss

mov ax, seg new\_stack

mov ss, ax

mov ax, offset new\_stack

add ax, 256

mov sp, ax

push ax

push bx

push cx

push dx

push si

push es

push ds

mov ax, seg key\_value

mov ds, ax

in al, 60h

cmp al, 12h ;e

je key\_e

cmp al, 19h ;p

je key\_p

cmp al, 23h ;h

je key\_h

pushf

call dword ptr cs:keep\_ip

jmp end\_interrupt

key\_e:

mov key\_value, '!'

jmp next\_key

key\_p:

mov key\_value, '#'

jmp next\_key

key\_h:

mov key\_value, '@'

next\_key:

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\_key:

mov ah, 05h

mov cl, key\_value

mov ch, 00h

int 16h

or al, al

jz end\_interrupt

mov ax, 40h

mov es, ax

mov ax, es:[1ah]

mov es:[1ch], ax

jmp print\_key

end\_interrupt:

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

new\_interrupt endp

\_end:

is\_int\_loaded proc

push ax

push bx

push si

mov ah, 35h

mov al, 09h

int 21h

mov si, offset signature

sub si, offset new\_interrupt

mov ax, es:[bx + si]

cmp ax, signature

jne end\_proc

mov IS\_LOADED, 1

end\_proc:

pop si

pop bx

pop ax

ret

is\_int\_loaded endp

int\_load proc

push ax

push bx

push cx

push dx

push es

push ds

mov ah, 35h

mov al, 09h

int 21h

mov keep\_cs, es

mov keep\_ip, bx

mov ax, seg new\_interrupt

mov dx, offset new\_interrupt

mov ds, ax

mov ah, 25h

mov al, 09h

int 21h

pop ds

mov dx, offset \_end

mov cl, 4h

shr dx, cl

add dx, 10fh

inc dx

xor ax, ax

mov ah, 31h

int 21h

pop es

pop dx

pop cx

pop bx

pop ax

ret

int\_load endp

unload\_interrupt 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 keep\_ip

sub si, offset new\_interrupt

mov dx, es:[bx + si]

mov ax, es:[bx + si + 2]

push ds

mov ds, ax

mov ah, 25h

mov al, 09h

int 21h

pop ds

mov ax, es:[bx + si + 4]

mov es, ax

push es

mov ax, es:[2ch]

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\_interrupt endp

is\_unloaded\_ proc

push ax

push es

mov ax, keep\_psp

mov es, ax

cmp byte ptr es:[82h], '/'

jne end\_unload

cmp byte ptr es:[83h], 'u'

jne end\_unload

cmp byte ptr es:[84h], 'n'

jne end\_unload

mov IS\_UNLOADED, 1

end\_unload:

pop es

pop ax

ret

is\_unloaded\_ endp

PRINT proc near

push ax

mov ah, 09h

int 21h

pop ax

ret

PRINT endp

begin proc

push ds

xor ax, ax

push ax

mov ax, DATA

mov ds, ax

mov keep\_psp, es

call is\_int\_loaded

call is\_unloaded\_

cmp IS\_UNLOADED, 1

je unload

mov al, IS\_LOADED

cmp al, 1

jne load

mov dx, offset STRING\_INT\_ALREADY\_LOADED

call PRINT

jmp end\_begin

load:

mov dx, offset STRING\_INT\_IS\_LOADING

call PRINT

call int\_load

jmp end\_begin

unload:

cmp IS\_LOADED, 1

jne not\_loaded

mov dx, offset STRING\_INT\_RESTORED

call PRINT

call unload\_interrupt

jmp end\_begin

not\_loaded:

mov dx, offset STRING\_INT\_NOT\_LOADED

call PRINT

end\_begin:

xor al, al

mov ah, 4ch

int 21h

begin endp

CODE ends

DATA SEGMENT

IS\_LOADED db 0

IS\_UNLOADED db 0

STRING\_INT\_IS\_LOADING db "Interrupt is loading now.", 0dh, 0ah, "$"

STRING\_INT\_ALREADY\_LOADED db "Interrupt is already loaded.", 0dh, 0ah, "$"

STRING\_INT\_RESTORED db "Interrupt was restored.", 0dh, 0ah, "$"

STRING\_INT\_NOT\_LOADED db "Interrupt was not loaded.", 0dh, 0ah, "$"

DATA ends

end begin