**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

**отчет**

по лабораторной работе №**5**

по дисциплине «Операционные системы»

на тему: Сопряжение стандартного и пользовательского обработчиков прерываний

|  |  |  |
| --- | --- | --- |
| Студент гр. 6381 |  | Кухарев М.А. |
| Преподаватель |  | Губкин А.Ф. |

Санкт-Петербург

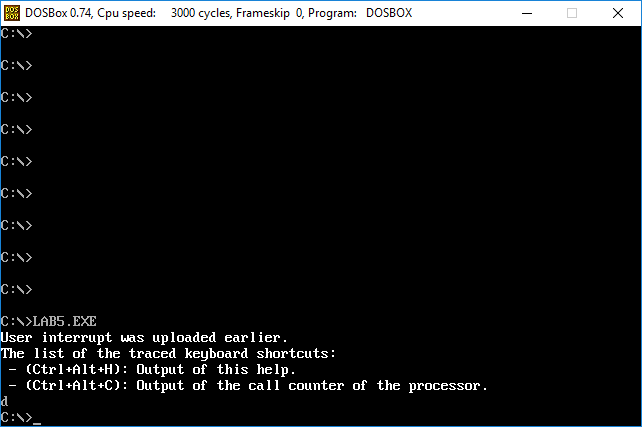
2018

**Цель работы:**

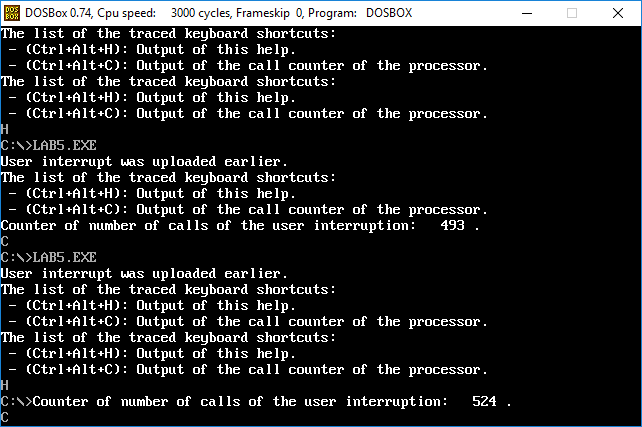
Исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Пользовательский обработчик прерываний получает управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре. Он обрабатывает скан-код и осуществляет определенные действия, если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

**Ход работы:**

1. Написан программный модуль .EXE, который выполняет такие же функции, как в программе лабораторной работы №4, будем использовать прерывание от клавиатуры int 09h. По нажатию комбинаций клавиш Ctrl+Alt+H и Ctrl+Alt+C прерывание будет выводить соответствующие сообщения. Если скан-код не совпадает с данными, то осуществляется передача управления стандартному обработчику прерывания.
2. Запуск программы. Введем любые символы, кроме заданных:

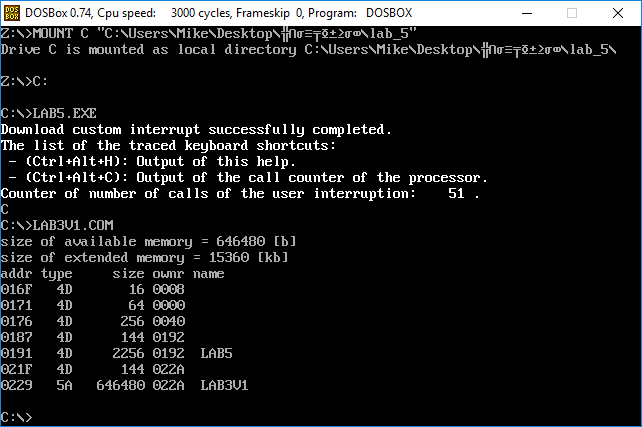


(рис. 1)

Управление передается стандартному обработчику. Введем комбинации клавиш Ctrl+Alt+H и Ctrl+Alt+C : 

(рис. 2)

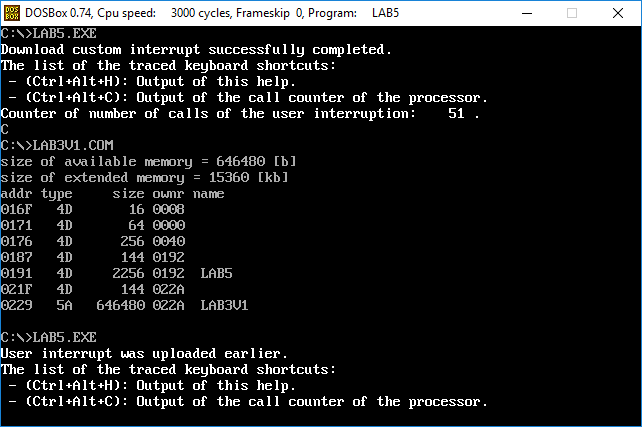
1. Проверка размещения программы в памяти. Воспользуемся программой из лабораторной роботы №3:



(рис. 3)

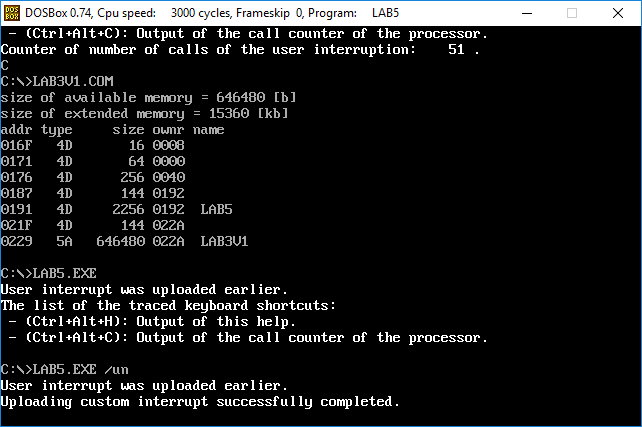
По рис. 3 видно, что резидент выгружен в память и используется.

1. Запустим программу еще раз: программа определяет установленный обработчик прерывания



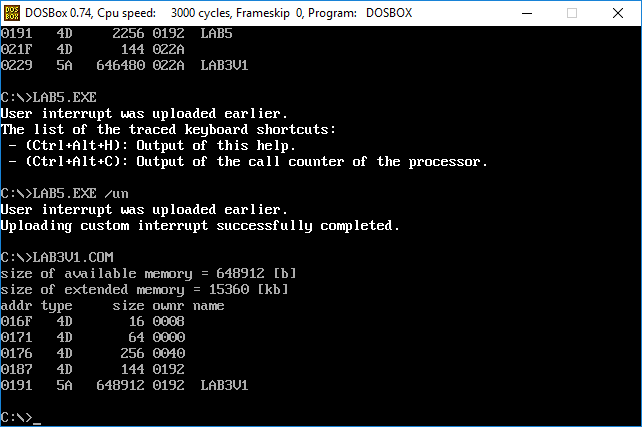
(рис. 4)

1. Запустим программу с ключом выгрузки /un (рис. 5):



(рис. 5)

Запустим программу лабораторной работы №3: резидентный обработчик прерывания выгружен (рис. 6):



(рис. 6)

**Ответы на контрольные вопросы:**

1. *Какого типа прерывания использовались в работе?*

В работе использовались программные и аппаратные (прерывания от контроллера клавиатуры при нажатии клавиш) прерывания.

1. *Чем отличается скан-код от кода ASCII?*

С помощью скан-кода драйвер клавиатуры распознает, какая именно клавиша была нажата. А ASCII – это всего лишь таблица всех имеющихся символов, которые могут быть выведены в консоль. К скан-кодам относятся также функциональные клавиши, клавиши управления.

**Вывод:**

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

**Приложение: Код программы**

.SEQ

L5\_CODE SEGMENT

ASSUME CS: L5\_CODE, DS: L5\_DATA, ES: NOTHING, SS: L5\_STACK

START: jmp l5\_start

L5\_DATA SEGMENT

PSP\_SIZ = 10h

STK\_SIZ = 10h

CMD\_BUF db 80h dup (00h)

HK\_KEY1 = 23h

HK\_ASC1 = 'H'

HK\_KEY2 = 2Eh

HK\_ASC2 = 'C'

L5\_SIGN db 'String = signature.', 0Dh, 0Ah, '$'

CMD\_ERR db 'Error! At the end of the command line detected an invalid parameter.', 0Dh, 0Ah, '$'

UNL\_ERR db 'Error!The program started with the key "/un" before the implementation of interrupts.', 0Dh, 0Ah, '$'

STA\_LOA db 'Download custom interrupt successfully completed.', 0Dh, 0Ah, '$'

STA\_ALR db 'User interrupt was uploaded earlier.', 0Dh, 0Ah, '$'

STA\_UNL db 'Uploading custom interrupt successfully completed.', 0Dh, 0Ah, '$'

INF\_USG db 'The list of the traced keyboard shortcuts:', 0Dh, 0Ah, '$'

INF\_HK1 db ' - (Ctrl+Alt+H): Output of this help.', 0Dh, 0Ah, '$'

INF\_HK2 db ' - (Ctrl+Alt+C): Output of the call counter of the processor.', 0Dh, 0Ah, '$'

INT\_CNT db 'Counter of number of calls of the user interruption: .', 0Dh, 0Ah, '$'

PRM\_ERR db 'Failed to release the memory used by the program. Error code: H.', 0Dh, 0Ah, '$'

ENM\_ERR db 'Unable to free the memory occupied by the environment. Error code: H.', 0Dh, 0Ah, '$'

KEEP\_IP dw ?

KEEP\_CS dw ?

IT\_CNTR dw 0

CHR\_EOT = '$'

INF\_CLR = 0Fh

L5\_DATA ENDS

L5\_STACK SEGMENT STACK

db STK\_SIZ \* 10h dup (?)

L5\_STACK ENDS

;Ghjwtlehs

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

TETR\_TO\_HEX PROC NEAR

and AL, 0Fh

cmp AL, 09h

jbe NEXT

add AL, 07h

NEXT: add AL, 30h

ret

TETR\_TO\_HEX ENDP

;перевод байта из AL в два символа HEX

BYTE\_TO\_HEX PROC NEAR

push CX

mov AH, AL

call TETR\_TO\_HEX

xchg AL, AH

mov CL, 04h

shr AL, CL

call TETR\_TO\_HEX

pop CX

ret

BYTE\_TO\_HEX ENDP

;перевод в HEX слова из AX

WRD\_TO\_HEX PROC NEAR

push AX

push BX

push DI

mov BH, AH

call BYTE\_TO\_HEX

mov DS:[DI], AH

dec DI

mov DS:[DI], AL

dec DI

mov AL, BH

call BYTE\_TO\_HEX

mov DS:[DI], AH

dec DI

mov DS:[DI], AL

pop DI

pop BX

pop AX

ret

WRD\_TO\_HEX ENDP

;перевод байта из AL в DEC

BYTE\_TO\_DEC PROC NEAR

push AX

push CX

push DX

push SI

xor AH, AH

xor DX, DX

mov CX, 0Ah

loop\_bd: div CX

or DL, 30h

mov DS:[SI], DL

dec SI

xor DX, DX

cmp AX, 0Ah

jae loop\_bd

cmp AL, 00h

je end\_l

or AL, 30h

mov DS:[SI], AL

end\_l: pop SI

pop DX

pop CX

pop AX

ret

BYTE\_TO\_DEC ENDP

;перевод слова из AX в DEC

WRD\_TO\_DEC PROC NEAR

push BX

xor BX, BX

call DWRD\_TO\_DEC

pop BX

ret

WRD\_TO\_DEC ENDP

;перевод BX:AX в DEC

DWRD\_TO\_DEC PROC NEAR

push AX

push BX

push CX

push DX

push DI

jmp clear\_dd

cont\_dd: mov AX, CX

mov BX, DX

clear\_dd: xor CX, CX

xor DX, DX

check\_dd: cmp BX, 00h

ja subst\_dd

cmp AX, 0Ah

jb write\_dd

subst\_dd: clc

sub AX, 0Ah

sbb BX, 00h

clc

add CX, 01h

adc DX, 00h

jmp check\_dd

write\_dd: add AX, 30h

mov DS:[DI], AL

dec DI

test CX, CX

jnz cont\_dd

test DX, DX

jnz cont\_dd

pop DI

pop DX

pop CX

pop BX

pop AX

ret

DWRD\_TO\_DEC ENDP

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

; вывод символа

PR\_CHR\_BIOS PROC NEAR

push AX

push BX

push CX

xchg AL, CL

mov AH, 0Fh

int 10h

xchg AL, CL

mov AH, 09h

mov CX, 01h

int 10h

pop CX

pop BX

pop AX

ret

PR\_CHR\_BIOS ENDP

;вывод текста

PR\_STR\_BIOS PROC NEAR

push AX

push BX

push CX

push DX

push DI

push ES

mov AX, DS

mov ES, AX

mov AH, 0Fh

int 10h

mov AH, 03h

int 10h

mov DI, 00h

dsbp\_nxt: cmp byte ptr DS:[BP+DI], CHR\_EOT

je dsbp\_out

inc DI

jmp dsbp\_nxt

dsbp\_out: mov CX, DI

mov AH, 13h

mov AL, 01h

int 10h

pop ES

pop DI

pop DX

pop CX

pop BX

pop AX

ret

PR\_STR\_BIOS ENDP

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

;ќбработчик прерывани¤

INT\_09H\_PRO PROC FAR

push AX

push BX

push CX

push BP

push DI

push DS

push ES

mov AX, L5\_DATA

mov DS, AX

mov AX, 40h

mov ES, AX

inc IT\_CNTR

mov AL, ES:[17h]

and AL, 00001100b

cmp AL, 00001100b

jne orig\_int

in AL, 60h

jmp chk\_hk01

orig\_int: pushf

call dword ptr DS:[KEEP\_IP]

jmp int\_quit

chk\_hk01: cmp AL, HK\_KEY1

jne chk\_hk02

mov AH, AL

in AL, 61h

or AL, 10000000b

out 61h, AL

and AL, 01111111b

out 61h, AL

mov BL, INF\_CLR

lea BP, INF\_USG

call PR\_STR\_BIOS

lea BP, INF\_HK1

call PR\_STR\_BIOS

lea BP, INF\_HK2

call PR\_STR\_BIOS

mov CH, AH

mov CL, HK\_ASC1

jmp buff\_wrt

chk\_hk02: cmp AL, HK\_KEY2

jne orig\_int

mov AH, AL

in AL, 61h

or AL, 10000000b

out 61h, AL

and AL, 01111111b

out 61h, AL

lea DI, INT\_CNT

add DI, 57

mov AX, IT\_CNTR

call WRD\_TO\_DEC

mov BL, INF\_CLR

lea BP, INT\_CNT

call PR\_STR\_BIOS

mov CH, AH

mov CL, HK\_ASC2

jmp buff\_wrt

buff\_wrt: mov AH, 05h

int 16h

cmp AL, 00h

jne int\_quit

jmp int\_quit

int\_quit: mov AL, 20h

out 20h, AL

pop ES

pop DS

pop DI

pop BP

pop CX

pop BX

pop AX

iret

INT\_09H\_PRO ENDP

;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

;начало работы программы

l5\_start: mov BX, L5\_DATA

mov DS, BX

push ES

mov AH, 35h

mov AL, 09h

int 21h

mov KEEP\_CS, ES

mov KEEP\_IP, BX

pop ES

jmp cmds\_buf

;обработка хвоста командной строки

cmds\_buf: xor BH, BH

xor CH, CH

mov CL, ES:[80h]

cmp CL, 00h

je sign\_chk

lea DI, CMD\_BUF

mov SI, 81h

mov AH, 00h

jmp cmds\_chk

cmds\_chk: mov AL, byte ptr ES:[SI]

cmp AL, '"'

jne cmds\_chr

not AH

and AH, 00000001b

jmp cmds\_nxt

cmds\_chr: cmp AL, ' '

jne cmds\_wrt

cmp AH, 00h

jne cmds\_wrt

mov AL, 01h

jmp cmds\_wrt

cmds\_wrt: mov DS:[DI], AL

inc DI

jmp cmds\_nxt

cmds\_nxt: inc SI

loop cmds\_chk

mov AL, 01h

mov DS:[DI], AL

cmp AH, 00h

jne cmds\_err

lea DI, CMD\_BUF

jmp pars\_chk

cmds\_err: mov BL, INF\_CLR

lea BP, CMD\_ERR

call PR\_STR\_BIOS

jmp dos\_quit

;проверка установленного вектора прерывани¤

sign\_chk: mov AX, L5\_DATA

sub AX, L5\_CODE

mov CX, KEEP\_CS

add CX, AX

mov ES, CX

lea DI, L5\_SIGN

lea CX, CMD\_ERR

sub CX, DI

xor BL, BL

jmp sign\_nxt

sign\_nxt: mov AL, DS:[DI]

mov AH, ES:[DI]

cmp AL, AH

jne cint\_chk

inc DI

loop sign\_nxt

mov BL, 01h

jmp cint\_chk

;проверка параметров консоли

pars\_chk: cmp byte ptr DS:[DI], 00h

je sign\_chk

cmp byte ptr DS:[DI], 01h

je pars\_nxt

jmp pars\_st1

pars\_st1: cmp byte ptr DS:[DI], '/'

je pars\_st2

cmp byte ptr DS:[DI], '\'

je pars\_st2

jmp pars\_unk

pars\_st2: inc DI

cmp byte ptr DS:[DI], 'u'

je pars\_st3

cmp byte ptr DS:[DI], 'U'

je pars\_st3

jmp pars\_unk

pars\_st3: inc DI

cmp byte ptr DS:[DI], 'n'

je pars\_ex1

cmp byte ptr DS:[DI], 'N'

je pars\_ex1

jmp pars\_unk

pars\_ex1: mov BH, 01h

jmp pars\_nxt

pars\_nxt: inc DI

jmp pars\_chk

pars\_unk: mov BL, INF\_CLR

lea BP, CMD\_ERR

call PR\_STR\_BIOS

jmp dos\_quit

;проверка флагов

cint\_chk: cmp BL, 00h

jne cint\_unf

cmp BH, 00h

je cint\_inj

mov BL, INF\_CLR

lea BP, UNL\_ERR

call PR\_STR\_BIOS

jmp dos\_quit

cint\_inj: push DS

mov AX, seg INT\_09H\_PRO

mov DS, AX

lea DX, INT\_09H\_PRO

mov AH, 25h

mov AL, 09h

int 21h

pop DS

mov BL, INF\_CLR

lea BP, STA\_LOA

call PR\_STR\_BIOS

mov BL, INF\_CLR

mov BL, INF\_CLR

mov BL, INF\_CLR

lea BP, INF\_USG

call PR\_STR\_BIOS

lea BP, INF\_HK1

call PR\_STR\_BIOS

lea BP, INF\_HK2

call PR\_STR\_BIOS

jmp cint\_res

cint\_res: mov AH, 01h

int 21h

mov DX, L5\_STACK

add DX, STK\_SIZ

sub DX, L5\_CODE

add DX, PSP\_SIZ

xor AL, AL

mov AH, 31h

int 21h

jmp dos\_quit

cint\_unf: cmp BH, 00h

jne cint\_unl

mov BL, INF\_CLR

lea BP, STA\_ALR

call PR\_STR\_BIOS

mov BL, INF\_CLR

mov BL, INF\_CLR

mov BL, INF\_CLR

lea BP, INF\_USG

call PR\_STR\_BIOS

lea BP, INF\_HK1

call PR\_STR\_BIOS

lea BP, INF\_HK2

call PR\_STR\_BIOS

jmp dos\_quit

cint\_unl: mov BL,INF\_CLR

lea BP, STA\_ALR

call PR\_STR\_BIOS

cli

push DS

mov AX, ES:[KEEP\_CS]

mov DS, AX

mov DX, ES:[KEEP\_IP]

mov AH, 25h

mov AL, 09h

int 21h

pop DS

sti

mov AX, KEEP\_CS

sub AX, PSP\_SIZ

mov ES, AX

mov BX, ES:[2Ch]

mov AH, 49h

int 21h

jc cint\_per

mov ES, BX

mov AH, 49h

int 21h

jc cint\_eer

mov BL, INF\_CLR

lea BP, STA\_UNL

call PR\_STR\_BIOS

jmp dos\_quit

cint\_per: lea DI, PRM\_ERR

add DI, 64

call WRD\_TO\_HEX

mov BL,INF\_CLR

lea BP, PRM\_ERR

call PR\_STR\_BIOS

jmp dos\_quit

cint\_eer: lea DI, ENM\_ERR

add DI, 64

call WRD\_TO\_HEX

mov BL, INF\_CLR

lea BP, ENM\_ERR

call PR\_STR\_BIOS

jmp dos\_quit

;выход из программы

dos\_quit: mov AH, 01h

int 21h

mov AH, 4Ch

int 21h

L5\_CODE ENDS

END START