**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Операционные системы»**

Тема: «**Обработка стандартных прерываний**»

|  |  |  |
| --- | --- | --- |
| Студент гр. 6381 |  | Попов Н.В. |
| Преподаватель |  | Губкин А.Ф. |

Санкт-Петербург

2018

**Постановка задачи.**

Цель работы: построение обработчика прерываний сигналов таймера.

Описание функций:

1. PRINT\_STR: выводит строку, помещенную в DX.
2. SET\_CURS: установка позиции курсора.
3. GET\_CURS: определение позиции и размера курсора.
4. OUTPUT\_AL: вывод символа в AL в текущее положение курсора.
5. ROUT: обработчик прерывания.
6. CHECK\_INT: проверка, установлено ли пользовательское прерывание с вектором 1Ch.
7. LOADING\_INT: загрузка резидентно пользовательского прерывания.
8. IS\_UNLOAD: проверка на необходимость загрузки прерывания.
9. UNLOADING\_INT: выгрузка пользовательского прерывания.

**Ход работы.**

1. Оценим состояние памяти до запуска программы l4.exe с помощью ранее разработанной программы l3.com:

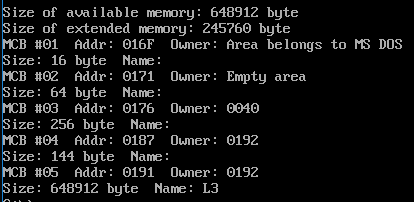


Рис. 1. Результат работы программы l3.com

1. Запуск программы l4.exe:



Рис. 2. Результат запуска программы l4.exe.

Как видно на Рис. 2, резидентный обработчик прерывания 1Ch установлен, и счетчик работает корректно, то есть прерывания от таймера срабатывают.

1. После загрузки резидента в память, проверим его размещение в памяти с помощью программы l3.com, которая отображает карту памяти в виде списка блоков MCB:



Рис. 3. Результат работы программы l3.com после загрузки резидентного обработчика прерывания 1Ch в память.

Исходя из результата убеждаемся, что обработчик прерывания остается в памяти резидентно.

1. Запустим программу l4.exe ещё раз:

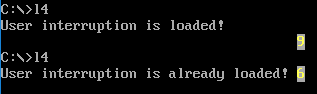


Рис. 4. Результат повторного запуска программы l4.exe.

Можно убедиться, в том, что программа определяет, что обработчик уже находится в памяти резидентно.

1. Запустим программу l4.exe с ключом выгрузки /un:



Рис. 5. Результат работы программы l4.exe, запущенной с ключом /un.

Наблюдаем, что резидентный обработчик прерывания был выгружен так, как об этом нас информирует сообщение, выведенное на экран программой, а также то, что счетчик больше не присутствует на экране.

1. Убедимся в том, что занятая резидентным обработчиком прерывания 1Сh память освобождена, запустив программу l3.com:

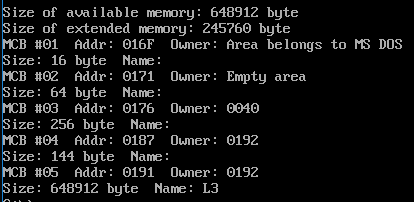


Рис. 6. Результат работы программы lab3\_1.com выгрузки пользовательского прерывания.

Исходя из результата убеждаемся, что обработчик прерывания был выгружен.

**Ответы на контрольные вопросы.**

1. *Как реализован механизм прерывания от часов?*

Прерывание 1Ch вызывается автоматически при каждом тике системного таймер (18,2 раз в секунду). После вызова, сохраняется содержимое регистров, затем определяется источник прерывания, по номеру которого определяется смещение в таблице векторов прерываний. Полученный адрес сохраняется в регистры СS:IP, после чего управление передается по этому адресу, т.е. выполняется запуск обработчика прерываний и происходит его выполнение. После выполнения, происходит возврат управления прерванной программе.

1. *Какого типа прерывания использовались в работе?*

В данной лабораторной работе использовались: аппаратные (int 1Ch) и программные прерывания (int 21h, int 10h).

**Заключение.**

В ходе данной лабораторной работы был построен обработчик прерываний сигналов таймера.

**Приложение А**

**Исходный код L3.asm**

PROG SEGMENT

ASSUME CS:PROG, DS:PROG, ES:NOTHING, SS:NOTHING

ORG 100H

START:

jmp main

;Данные

avail\_mem db 13,10,"Size of available memory: $"

exten\_mem db 13,10,"Size of extended memory: $"

str\_MCB db "MCB #0 Addr: Owner: $"

str\_size db 13, 10, "Size: $"

str\_own\_name db " Name: $"

str\_own\_1 db "Empty area$"

str\_own\_2 db "Area belongs to OS XMS UMB driver $"

str\_own\_3 db "Area of excluded upper driver memory $"

str\_own\_4 db "Area belongs to MS DOS $"

str\_own\_5 db "Area occuped by control block 386MAX UMB $"

str\_own\_6 db "Area blocked 386MAX $"

str\_own\_7 db "Area belongs 386MAX UMB$"

str\_owner db " $"

str\_Byte db " byte$"

str\_enter db 13,10,"$"

fr\_mem db "Freeing memory: $"

al\_mem db "Allocating memory: $"

er7 db "Failed. Memory management blocks destroyed",13,10,"$"

er8 db "Failed. Not enough memory",13,10,"$"

er9 db "Failed. ES contains invalid address",13,10,"$"

er\_unk db "Failed. Unknown error",13,10,"$"

success db "Success",13,10,"$"

;---------------------Procedures-------------------

TETR\_TO\_HEX PROC NEAR

and al,0fh

cmp AL,09

jbe NEXT

add AL,07

NEXT:

add AL,30h

ret

TETR\_TO\_HEX ENDP

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

BYTE\_TO\_HEX PROC near;байт в AL переводится в два символа шестн. числа в AX

push CX

mov AH,AL

call TETR\_TO\_HEX

xchg AL,AH

mov CL,4

shr AL,CL

call TETR\_TO\_HEX ; в AL старшая цифра

pop CX ; в AH младшая

ret

BYTE\_TO\_HEX ENDP

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

WRD\_TO\_HEX PROC near ;перевод в 16 с/с 16-ти разрядного числа

push BX ;AX - число, DI - адрес последего символа

mov BH,AH

call BYTE\_TO\_HEX

mov [DI],AH

dec DI

mov [DI],AL

dec DI

mov AL,BH

call BYTE\_TO\_HEX

mov [DI],AH

dec DI

mov [DI],AL

pop BX

ret

WRD\_TO\_HEX ENDP

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

BYTE\_TO\_DEC PROC near ; перевод в 10с/с, SI - адрес поля младшей цифры

push CX

push DX

xor AH,AH

xor DX,DX

mov CX,10

loop\_bd:

div CX

or DL,30h

mov [SI],DL

dec SI

xor DX,DX

cmp AX,10

jae loop\_bd

cmp AL,00h

je end\_l

or AL,30h

mov [SI],AL

end\_l:

pop DX

pop CX

ret

BYTE\_TO\_DEC ENDP

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

PRINT\_STR PROC near ;Печать строки, помещенной в DX

push ax

mov ah,09h

int 21h

pop ax

ret

PRINT\_STR ENDP

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

PRINT\_SIZE PROC near

mov bx,10h

mul bx

mov bx,0ah

xor cx,cx

del:

div bx

push dx

inc cx

xor dx,dx

cmp ax,0

jnz del

writeSymb:

pop dx

or dl,30h

mov ah,02h

int 21h

loop writeSymb

ret

PRINT\_SIZE ENDP

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

AMOUNT\_OF\_AVAILABLE\_MEMORY PROC NEAR

mov ah,4Ah

mov bx,0FFFFh

int 21h

mov ax,bx

lea dx,avail\_mem

call PRINT\_STR

call PRINT\_SIZE

lea dx,str\_Byte

call PRINT\_STR

ret

AMOUNT\_OF\_AVAILABLE\_MEMORY ENDP

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

EXTENDED\_MEMORY\_SIZE PROC NEAR

mov al,30h

out 70h,al

in al,71h

mov bl,al

mov al,31h

out 70h,al

in al,71h

mov bh,al

mov ax,bx

lea dx,exten\_mem

call PRINT\_STR

call PRINT\_SIZE

lea dx,str\_Byte

call PRINT\_STR

ret

EXTENDED\_MEMORY\_SIZE ENDP

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

CHAIN\_OF\_MEMORY\_CONTROL\_BLOCKS PROC NEAR

mov ah,52h

int 21h

mov ax,es:[bx-2]

mov es,ax

xor cx,cx

inc cx

lea dx, str\_enter

call PRINT\_STR

nextMCB:

lea si, str\_MCB

add si, 6

mov al,cl

push cx

call BYTE\_TO\_DEC

mov ax,es

lea di,str\_MCB

add di,18

call WRD\_TO\_HEX

xor ah,ah

mov al,es:[0]

push ax

mov ax,es:[1]

lea dx,str\_MCB

call PRINT\_STR

cmp ax,0000h

je g1

cmp ax,0006h

je g2

cmp ax,0007h

je g3

cmp ax,0008h

je g4

cmp ax,0FFFAh

je g5

cmp ax,0FFFDh

je g6

cmp ax,0FFFEh

je g7

lea di,str\_owner

add di, 3

call WRD\_TO\_HEX

lea dx,str\_owner

call PRINT\_STR

jmp go

g1:

lea dx,str\_own\_1

call PRINT\_STR

jmp go

g2:

lea dx,str\_own\_2

call PRINT\_STR

jmp go

g3:

lea dx,str\_own\_3

call PRINT\_STR

jmp go

g4:

lea dx,str\_own\_4

call PRINT\_STR

jmp go

g5:

lea dx,str\_own\_5

call PRINT\_STR

jmp go

g6:

lea dx,str\_own\_6

call PRINT\_STR

jmp go

g7:

lea dx,str\_own\_7

call PRINT\_STR

go:

mov ax,es:[3]

lea dx,str\_size

call PRINT\_STR

call PRINT\_SIZE

lea dx,str\_Byte

call PRINT\_STR

lea dx,str\_own\_name

call PRINT\_STR

mov cx,8

xor di,di

write:

mov dl,es:[di+8]

mov ah,02h

int 21h

inc di

loop write

mov ax,es:[3]

mov bx,es

add bx,ax

inc bx

mov es,bx

pop ax

pop cx

inc cx

cmp al,5ah

je exit

cmp al,4dh

jne exit

lea dx,str\_enter

call PRINT\_STR

jmp nextMCB

exit:

ret

CHAIN\_OF\_MEMORY\_CONTROL\_BLOCKS ENDP

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

FREE\_MEM PROC NEAR

lea dx,fr\_mem

call PRINT\_STR

lea BX, newstk

mov CL,04h

add BX,10Fh

shr BX, CL

mov AH,4Ah

int 21h

call TEST\_ERROR

ret

FREE\_MEM ENDP

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

MEMORY\_ALLOCATION PROC NEAR

lea dx,al\_mem

call PRINT\_STR

mov ah,48h

mov bx,1000h

int 21h

call TEST\_ERROR

ret

MEMORY\_ALLOCATION ENDP

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

TEST\_ERROR PROC NEAR

jnc ok

cmp ax,07h

lea dx,er7

call PRINT\_STR

jmp ex

cmp ax,08h

lea dx,er8

call PRINT\_STR

jmp ex

cmp ax,09h

lea dx,er9

call PRINT\_STR

jmp ex

lea dx,er\_unk

call PRINT\_STR

jmp ex

ok:

lea dx,success

call PRINT\_STR

ex:

ret

TEST\_ERROR ENDP

;------------------------CODE----------------------

main:

; call FREE\_MEM

; call MEMORY\_ALLOCATION

call AMOUNT\_OF\_AVAILABLE\_MEMORY

call EXTENDED\_MEMORY\_SIZE

call CHAIN\_OF\_MEMORY\_CONTROL\_BLOCKS

xor AL,AL

mov AH,4Ch

int 21H

dw 64 dup(?)

newstk=$

PROG ENDS

END START

**Исходный код L4.asm**

INT\_STACK SEGMENT

DW 100h DUP(?)

INT\_STACK ENDS

;////////////////////////////////////////////////////////

STACK SEGMENT STACK

DW 64 DUP (?)

STACK ENDS

;////////////////////////////////////////////////////////

DATA SEGMENT

INT\_ALR\_LOADED DB 'User interruption is already loaded!',10,13,'$'

INT\_UNLOADED DB 'User interruption is unloaded!',10,13,'$'

INT\_LOADED DB 'User interruption is loaded!',10,13,'$'

INT\_NOT\_LOADED db 'User interruption is not loaded',13,10,'$'

DATA ENDS

;////////////////////////////////////////////////////////

CODE SEGMENT

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

start: jmp MAIN

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

PRINT\_STR PROC

push ax

mov ah, 09h

int 21h

pop ax

ret

PRINT\_STR ENDP

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

SET\_CURS PROC ;установка позиции курсора

push AX

push BX

push CX

mov AH,02h

mov BH,00h

int 10h

pop CX

pop BX

pop AX

ret

SET\_CURS ENDP

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

GET\_CURS PROC ;определение позиции и размера курсора

push AX

push BX

push CX

mov AH,03h

mov BH,00h

int 10h

pop CX

pop BX

pop AX

ret

GET\_CURS ENDP

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

OUTPUT\_AL PROC

push ax

push bx

push cx

mov ah,09h

mov bh,0

mov cx,1

int 10h

pop cx

pop bx

pop ax

ret

OUTPUT\_AL ENDP

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

ROUT PROC FAR ;обработчик прерывания

jmp ROUT\_CODE

ROUT\_DATA:

SIGNATURE DB '0000'

KEEP\_CS DW 0

KEEP\_IP DW 0

KEEP\_PSP DW 0

COUNT db 0

COUNT\_1 db 0

COUNT\_2 db 0

COUNT\_3 db 0

COUNT\_4 db 0

KEEP\_SS dw 0

KEEP\_SP dw 0

KEEP\_DX dw 0

ROUT\_CODE:

push ax

mov KEEP\_SS,ss

mov KEEP\_SP,sp

mov ax, seg INT\_STACK

mov ss, ax

mov sp, 100h

push ax

push bx

push cx

push dx

push ds

push es

call GET\_CURS

push dx

mov dx,01625h

call SET\_CURS

cmp COUNT,0AH

jl rout\_skip

mov count,0h

rout\_skip:

mov al,COUNT

or al,30h

call OUTPUT\_AL

pop dx

call SET\_CURS

inc COUNT

jmp int\_end

call GET\_CURS

push dx

mov dh,22

mov dl,40

call SET\_CURS

cmp COUNT\_1,10

jl next\_it

mov COUNT\_1,0

add COUNT\_2,1

cmp COUNT\_2,10

jl next\_it

mov COUNT\_2,0

add COUNT\_3,1

cmp COUNT\_3,10

jl next\_it

mov COUNT\_3,0

add COUNT\_4,1

cmp COUNT\_4,10

jl next\_it

mov COUNT\_4,0

next\_it:

mov al,COUNT\_1

add al,30h

call OUTPUT\_AL

mov dl,39

call SET\_CURS

mov al,COUNT\_2

add al,30h

call OUTPUT\_AL

mov dl,38

call SET\_CURS

mov al,COUNT\_3

add al,30h

call OUTPUT\_AL

mov dl,37

call SET\_CURS

mov al,COUNT\_4

add al,30h

call OUTPUT\_AL

inc COUNT\_1

pop dx

call SET\_CURS

int\_end:

pop es

pop ds

pop dx

pop cx

pop bx

pop ax

mov ss, KEEP\_SS

mov sp, KEEP\_SP

mov al,20h

out 20h,al

pop ax

iret

end\_inter:

ROUT ENDP

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

CHECK\_INT PROC

mov AH,35h

mov AL,1Ch

int 21h

mov SI, offset SIGNATURE

sub SI, offset ROUT

mov AX,'00' ;сравнивание известных значений сигнатуры

cmp AX,ES:[BX+SI]

jne NO\_LOADED

cmp AX,ES:[BX+SI+2]

jne NO\_LOADED

mov AX,1h

ret

NO\_LOADED:

mov AX,0h

ret

CHECK\_INT ENDP

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

LOADING\_INT PROC near

push ax

push cx

push bx

push dx

push ds

mov ah, 35h

mov al, 1Ch

int 21h

mov KEEP\_IP, bx

mov KEEP\_CS, es

mov ax, SEG ROUT

mov dx, OFFSET ROUT

mov ds,ax

mov ah, 25h

mov al, 1Ch

int 21h

mov dx, OFFSET end\_inter

mov cl,4

shr dx,cl

inc dx

add dx, CODE

sub dx, KEEP\_PSP

mov ah, 31h

int 21h

pop ds

pop dx

pop bx

pop cx

pop ax

ret

LOADING\_INT ENDP

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

IS\_UNLOAD PROC near

push di

mov di, 81h

cmp byte ptr [di+0], ' '

jne bad\_key

cmp byte ptr [di+1], '/'

jne bad\_key

cmp byte ptr [di+2], 'u'

jne bad\_key

cmp byte ptr [di+3], 'n'

jne bad\_key

cmp byte ptr [di+4], 0Dh

jne bad\_key

cmp byte ptr [di+5], 0h

jne bad\_key

pop di

mov al,1

ret

bad\_key:

pop di

mov al,0

ret

IS\_UNLOAD ENDP

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

UNLOADING\_INT PROC near

push ax

push dx

mov ah, 35h

mov al, 1Ch

int 21h

cli

push ds

mov dx, es:KEEP\_IP

mov ax, es:KEEP\_CS

mov ds, ax

mov ah, 25h

mov al, 1Ch

int 21h

pop ds

mov es, es:KEEP\_PSP

push es

mov es, es:[2Ch]

mov ah, 49h

int 21h

pop es

mov ah, 49h

int 21h

sti

pop dx

pop ax

ret

UNLOADING\_INT ENDP

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

MAIN:

push ds

sub ax,ax

push ax

mov cs:KEEP\_PSP, es

call CHECK\_INT

cmp al, 1

je loaded

call IS\_UNLOAD

cmp al, 1

je not\_loaded

mov dx, offset INT\_LOADED

mov ax,DATA

mov ds,ax

mov ah, 9

int 21h

call LOADING\_INT

jmp end\_prog

not\_loaded:

mov dx, offset INT\_NOT\_LOADED

mov ax,DATA

mov ds,ax

mov ah, 9

int 21h

jmp end\_prog

loaded:

call IS\_UNLOAD

cmp al, 1

je need\_to\_unload

mov dx, offset INT\_ALR\_LOADED

mov ax,DATA

mov ds,ax

mov ah, 9

int 21h

jmp end\_prog

need\_to\_unload:

call UNLOADING\_INT

mov dx, offset INT\_UNLOADED

mov ax,DATA

mov ds,ax

mov ah, 9

int 21h

jmp end\_prog

end\_prog:

xor al,al

mov ah,4ch

int 21h

LAST\_BYTE:

CODE ENDS

END START