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. 7383 _____ Ханова Ю.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2019

Цель работы

Построить обработчик прерываний сигналов таймера. Таким образом, управление будет передано функции, чья точка входа записана в соответствующий вектор прерывания.

Ход работы

Разработан набор функций для выполнения поставленной задачи, функции приведены в табл. 1.

Таблица 1 - Описание функций.

Название функции	**Назначение**
outputAL	Функция вывода символа из AL
outputBP	Функция вывода строки по адресу ES:BP на экран
GetCurs	Читать позицию и размер курсора
SetCurs	Установка позиции курсора
my_int	Создание резидентного прерывания
print	Печать текста
save_int	Сохранение старого прерывания
new_int	Установка резидентного прерывания
TETR_TO_HEX	Вспомогательная функция для работы функции BYTE_TO_HEX
WRD_TO_HEX	Переводит в 16-ую систему счисления 16 разрядное число
BYTE_TO_HEX	Переводит число в коды символов 16-ой системы счисления
interrupt_load	Загрузка резидентного прерывания
int_remove	Удаление резидентного прерывания
main	Основная функция

Примеры работы программы отображены на рис. 1-4.

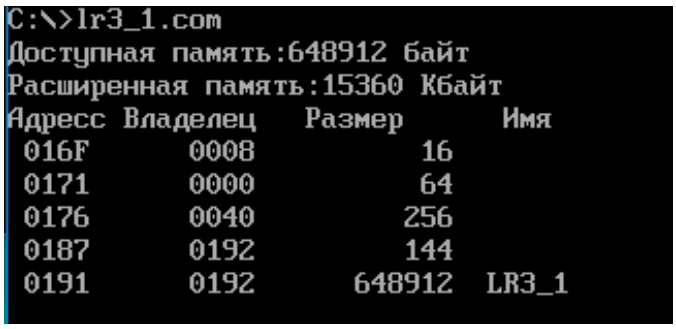

Рис. 1 - Проверка состояния памяти до запуска утилиты, при помощи lr3_1.com.

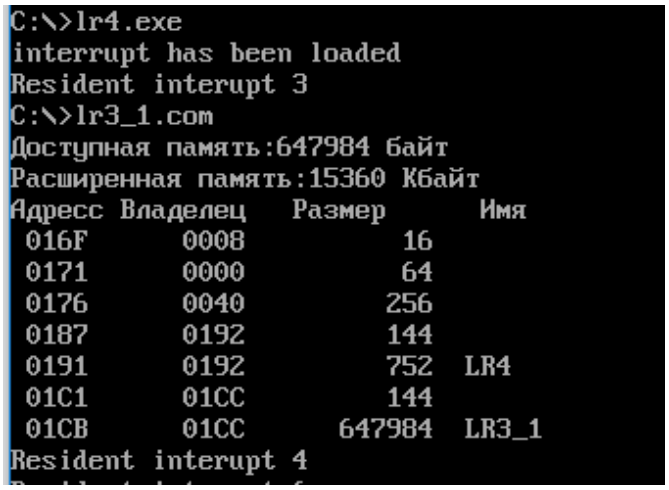

Рис. 2 - Загрузка утилиты lr4 в память и очередной вывод состояния памяти

Рис. 3 - Повторный запуск lr4

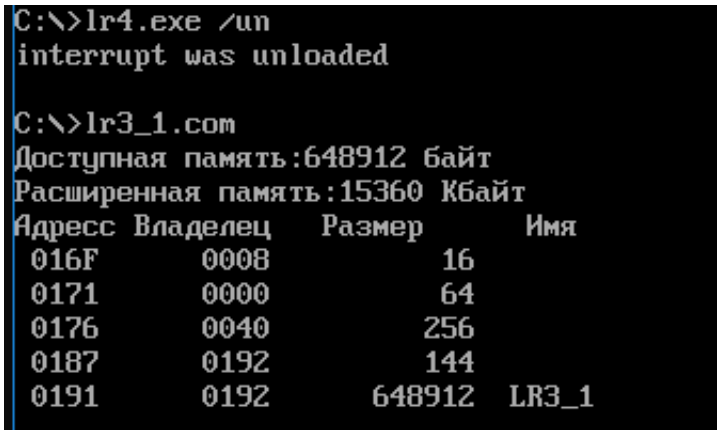

Рис. 4 - Запустим программу lr4.exe с ключом выгрузки /un и последующий вызов

Выводы.

В процессе выполнения данной лабораторной работы был построен обработчик прерываний сигналов таймера. Код программы представлен в приложении А.

Ответы на контрольные вопросы.

- Как реализован механизм прерывания от часов?

Сначала сохраняется содержимое регистров, потом определяется источник прерывания, по номеру которого определяется смещение в таблице векторов прерывания, сохраняется в CS:IP, передаётся управление по адресу CS:IP и происходит выполнение обработчика, и в конце происходит возврат управления прерванной программе. Прерывания генерируются системным таймером с частотой 18,206 Гц.

- Какого типа прерывания использовались в работе?

В программе использовались пользовательское прерывание по таймеру 1Ch, вектор прерывания 03h (используется отладчиками, чтобы перехватывать управление, когда программа достигает указанного пользователем адреса), 02h (посылает символ из DL на стандартный вывод) и программные прерывания 21h и 10h.

ПРИЛОЖЕНИЕ А

lr4.asm

```
code segment
	assume cs:code, ds:data, ss:stack_seg
	stack_seg SEGMENT STACK
	DW 64 DUP(?)
stack_seg ENDS
start_m:
	PSP dw 0
	KEEP_CS dw 0
	KEEP_IP dw 0
	LOAD_COUNT dw 0
	LOAD_STR db '1234567890'
	MESSAGE db 'Resident interupt     $'

my_int proc far
	jmp body_my_int_m
	INT_SET_FLAG dw 1111h
		ss_keeper dw ?
	sp_keeper dw ?
	ax_keeper dw ?
	inter_stack dw 64 dup (?)
body_my_int_m:

	mov ss_keeper, ss
	mov sp_keeper, sp
	mov ax_keeper, ax
	mov ax, seg inter_stack
	mov ss, ax
	mov sp, 0
	mov ax, ax_keeper
	push ax
	push bx
	push cx
	push dx
	push si
	push ds

	mov ax, seg code
	mov ds, ax
	inc cs:LOAD_COUNT
	cmp cs:LOAD_COUNT, 10
	jnz next_my_int_m
	mov cs:LOAD_COUNT, 0
next_my_int_m:
	lea bx, cs:LOAD_STR
	add bx, cs:LOAD_COUNT
```

```
        mov al, [bx]
        lea si, cs:MESSAGE
        add si, 18
        mov [si], al
        call GetCurs
        push dx
        dec dh
        mov dl, 0
        call SetCurs

        push bp
        push es
        mov ax, seg code
        mov es, ax
        lea bp, ES:MESSAGE

        call calc_cx

        mov ax, 1301h
        mov bx, 0007h
        call GetCurs

        int 10h

        pop es
        pop bp
        pop dx
        call SetCurs
        pop ds
        pop si
        pop dx
        pop cx
        pop bx
        pop ax
        mov al, 20h
        out 20h, al
              mov ax, ss_keeper
        mov ss, ax
        mov ax, ax_keeper
        mov sp, sp_keeper
        iret
my_int endp
resident_end:

;end_res=$

outputAL proc
```

```
        ;call setCurs
        push ax
        push bx
        push cx
        mov ah, 09h   ;писать символ в текущей позиции курсора
        mov bh, 0     ;номер видео страницы
        mov cx, 1     ;число экземпляров символа для записи
        int 10h      ;выполнить функцию
        pop cx
        pop bx
        pop ax
        ret
outputAL endp

outputBP proc near
        push ax
        push bx
        push dx
        push cx
        mov ah, 13h
        mov al, 1
        mov bh, 0
        mov dh, 22
        mov dl, 0
        int 10h
        pop cx
        pop dx
        pop bx
        pop ax
        ret
outputBP  endp

GetCurs      proc near
        push ax
        push bx
        push cx
        mov ah,03h
        mov bh,0
        mov bl, 7
        int 10h
        pop cx
        pop bx
        pop ax
        ret
GetCurs      endp

SetCurs      proc near
```

```
        push ax
        push bx
        mov ah,02h
        mov bh,0
        mov bl, 07
        int 10h
        pop bx
        pop ax
        ret
SetCurs      endp

calc_cx proc
        push dx
        push bp
        dec bp
        xor cx,cx
        dec cx
loop_cx:
        inc cx
        inc bp
        mov dl,es:[bp]
        cmp dl,'$'
        jnz loop_cx
        pop bp
        pop dx
        ret
calc_cx endp



print macro string
        push dx
        push ax
        xor ax, ax
        xor dx, dx
        lea dx, string
        mov AH,09h
    int 21h
        pop ax
        pop dx
endm

save_int proc near
        push ax
        push bx
        push cx
        push dx
```

```
        push es
        push di
        mov ax, 351Ch
        int 21h
        mov cs:KEEP_IP, bx
        mov cs:KEEP_CS, es
        pop di
        pop es
        pop dx
        pop cx
        pop bx
        pop ax
        ret
save_int endp

new_int proc near
        push ax
        push bx
        push cx
        push dx
        push ds
        mov dx, offset my_int
        mov ax, seg my_int
        push ax
        push dx

    print INSTALL_INT

        xor al, al
        mov ah, 1
        pop dx
        pop ax
        mov ds, ax
        mov ax, 251Ch
        int 21h
        pop ds
        pop dx
        pop cx
        pop bx
        pop ax
        ret
new_int endp


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
;Byte in AL converted to two HEX symbols in AX
       push CX
       mov AH,AL
       call TETR_TO_HEX
       xchg AL,AH
       mov CL,4
       shr AL,CL
       call TETR_TO_HEX ; in AL high order digit
       pop CX ;in AH low
       ret
BYTE_TO_HEX ENDP
;-------------------------------

WRD_TO_HEX PROC near
       push BX
       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


interrupt_load proc near

       mov dx, seg code
       add dx, (resident_end-start_m)
       mov cl, 4
       shr dx, cl ;div 16
       inc dx
```

```
        mov ah, 31h
        int 21h
        ret
        ;      stack_area dw 128 dup (?)
interrupt_load endp
        ;end_res=$

int_remove proc near
        cli
        push ax
        push bx
        push cx
        push dx
        push ds
        push es
        push di

        mov ax, 351Ch
        int 21h
        mov ax, es:[2]
        mov cs:KEEP_CS, ax
        mov ax, es:[4]
        mov cs:KEEP_IP, ax

        mov ax, cs:KEEP_CS
        mov dx, cs:KEEP_IP
        lea di, DELETE_INT
        add di, 60
        mov ax, cs:KEEP_CS
        call WRD_TO_HEX
        add di, 8
        mov ax, cs:KEEP_IP
        call WRD_TO_HEX

        print DELETE_INT

        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 ax, 251Ch
        int 21h

        pop di
        pop es
        pop ds
        pop dx
        pop cx
        pop bx
        pop ax
        sti
        ret
int_remove 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
        jnz empty_tail

        mov ax, es:[82h]
        cmp al, '/'
        jnz empty_tail
        cmp ah, 'u'
        jnz empty_tail
        mov ah, es:[84h]
        cmp ah, 'n'
        jnz empty_tail
        mov DEL_FLAG, 1
        jmp next_main_m
empty_tail:

next_main_m:
        mov ax, 351Ch
        int 21h
        mov ax, es:[bx+3]
        cmp ax, 1111h
        jz already_loaded
```

```
        cmp DEL_FLAG, 1
        jz mark

        call save_int
        call new_int
        call interrupt_load

        jmp fin
already_loaded:
        cmp DEL_FLAG, 1
        jz int_unload
        print ALREADY_LOAD
        jmp fin
int_unload:
        call int_remove
        jmp fin
mark:
        print NOT_LOADED
        jmp fin
fin:
        xor al, al
        mov ah, 4Ch
        int 21h
        ret
;       stack_area dw 128 dup (?)
main endp
;end_res=$
code ends

data segment
        DEL_FLAG db 0
        ENDL db      10, 13, '$'
        SPACE db ' $'
        DEFAULT_INT  db 'Default interrupt loaded', 10, 13, '$'
        INSTALL_INT  db 'interrupt has been loaded', 10, 13, '$'
        ALREADY_LOAD db 'interrupt is already loaded', 10, 13, '$'
        PRESS db 'Press any key', 10, 13, '$'
        DELETE_INT db 'interrupt was unloaded', 10, 13, '$'
        NOT_LOADED db 'interrupt is not loaded ', 10, 13, '$'

data ends
end main
```