МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Операционные системы»

Тема: Обработка стандартных прерываний

Студент гр. 9383 _____ Арутюнян С.Н.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

Цель работы

Построить обработчик прерываний сигналов таймера. Эти сигналы генерируются аппаратурой через определенные интервалы времени и, при возникновении такого сигнала, возникает прерывание с определенным значением вектора.

Выполнение работы

Шаг 1. Был написан и отлажен программный модуль типа .EXE, который выполняет следующие функции:

1. Проверяет, установлено ли пользовательское прерывание с вектором 1Ch.

2. Устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено, и осуществляте выход по функции 4Ch прерывания int 21h.

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

4. Выгрузка прерывания по соответствующему значению параметра в командной строке \un. Выгрузка прерывания состоит в восстановлении стандартного вектора прерываний и освобождения памяти, занимаемой резидентом. Затем осуществляется выход по функции 4ch прерывания int 21h.

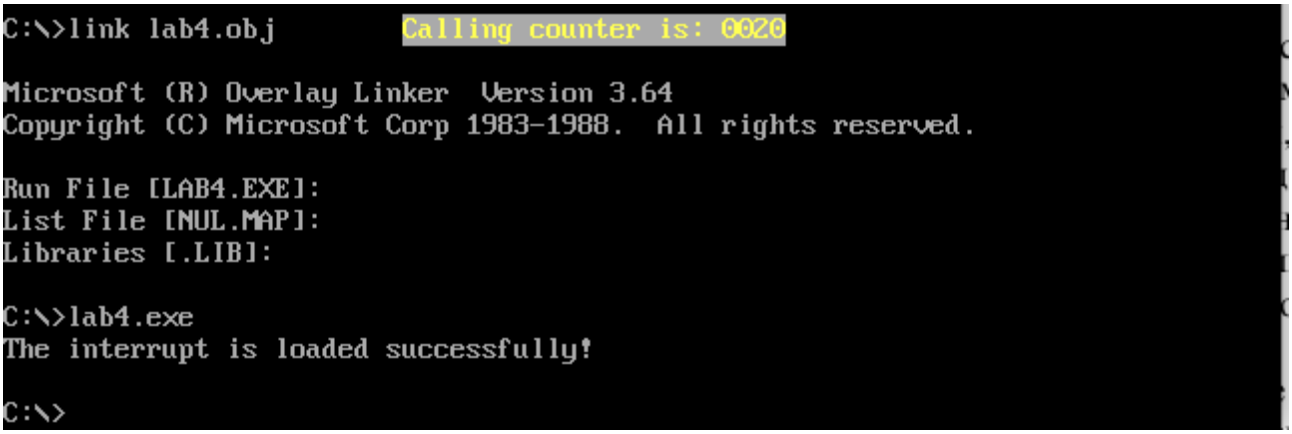

Рис. 1. Работа установленного прерывания

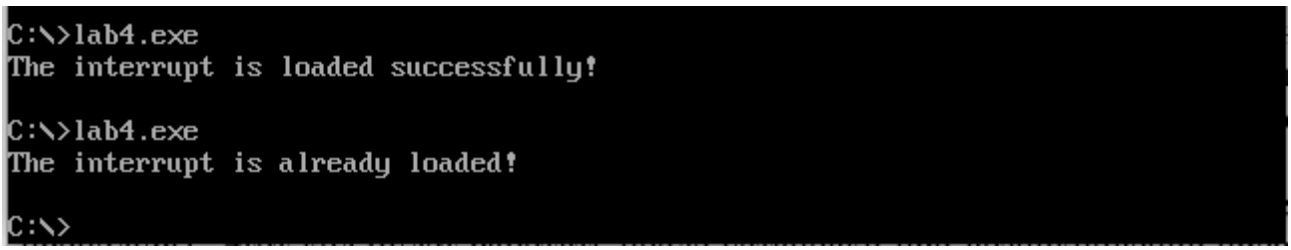

Рис. 2. Пример обработки попытки повторной установки прерывания

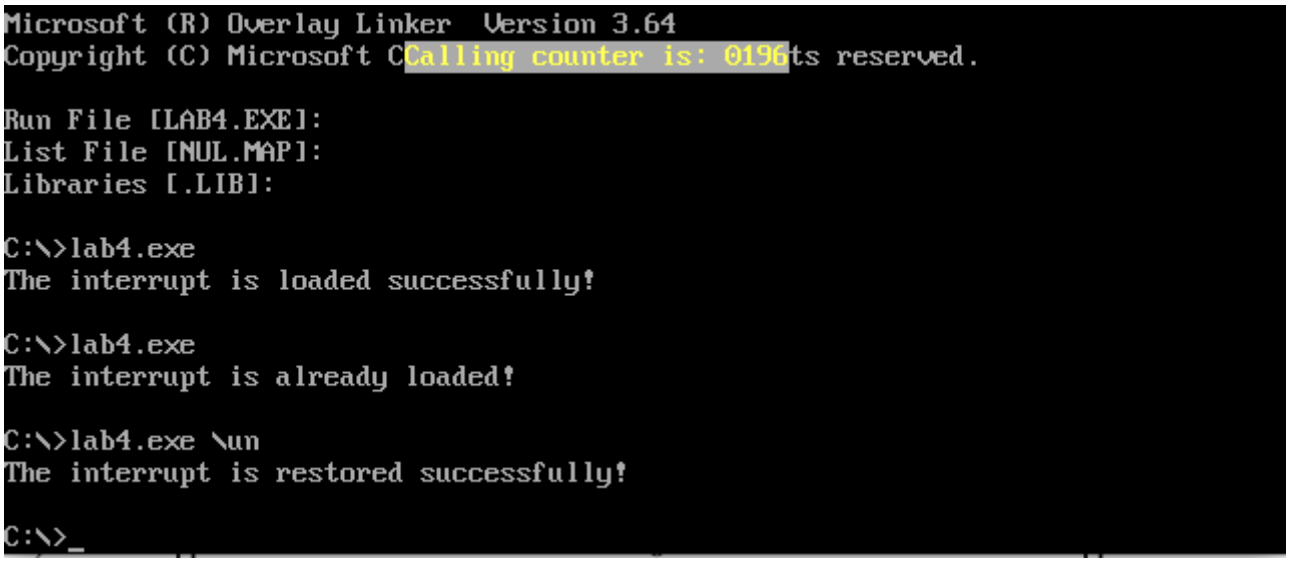

Рис. 3. Пример выгрузки прерывания

(после запуска программы счетчик остановился).

Шаг 2. Отлаженная программа была запущена и я убедился, что резидентный обработчик прерывания 1Ch установлен, т. к. на экране начала отображаться работа прерывания и была запущена программа из лабораторной работы №3:

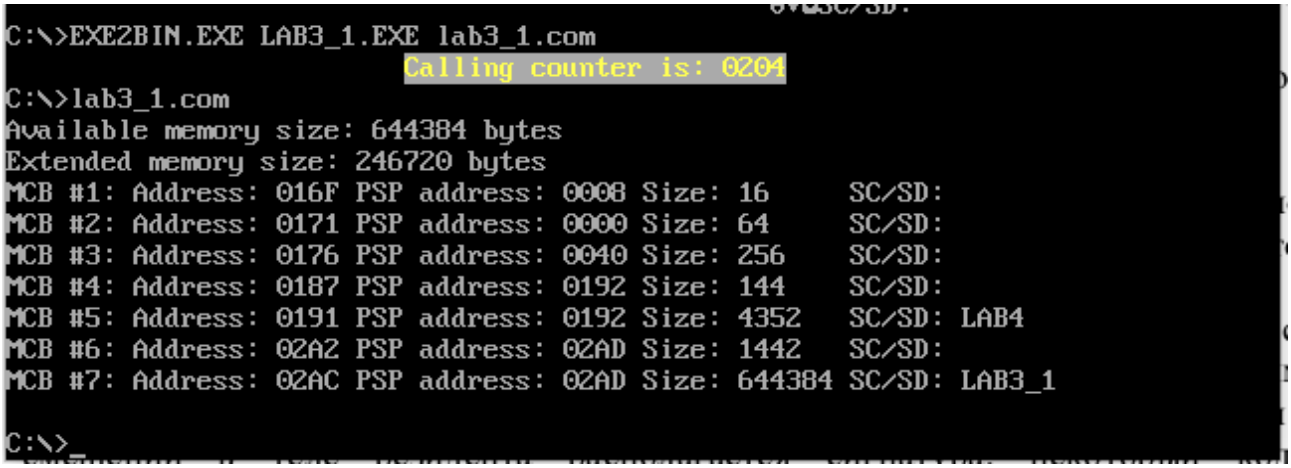

Рис. 4. Пример работы программы из лабораторной работы №3

Контрольные вопросы

1. Как реализован механизм прерывания от часов?

Сигнал от часов генерируется аппаратурой через определенные интервалы времени (около 18 раз в секунду). За каждым таким сигналом следует возникновение прерывания с вектором 1Ch. Соответственно, при генерации сигнала управление передается функции, определенной в таблице прерываний с номером 1Ch.

2. Какого типа прерывания использовались в работе?

В работе использовались программные (21h и 10h, т. е. выход в DOS и получение информации о курсоре) и аппаратные прерывания (1Ch, т. е. прерывание таймера).

Заключение

В процессе выполнения лабораторной работы был изучен механизм работы аппаратного таймера, а также были получены навыки реализации собственных резидентных прерываний.

Приложение А

```
AStack SEGMENT STACK
   dw 256 DUP(?)   ; 1 килобайт
AStack ENDS

DATA SEGMENT

NEWLINE db 0dh, 0ah, '$'
INT_ALREADY_LOADED db "The interrupt is already loaded!", 0dh, 0ah, '$'
INT_IS_NOT_LOADED db "There is no our loaded interrupt!", 0dh, 0ah, '$'
INT_LOADED_SUCCESS db "The interrupt is loaded successfully!", 0dh, 0ah, '$'
INT_IS_RESTORED_SUCCECCFULLY db "The interrupt is restored successfully!", 0dh,
0ah, '$'

DATA ENDS



CODE SEGMENT
   ASSUME cs:CODE, ds:DATA, ss:AStack



PRINT_NEWLINE proc near

   push ax
   push dx

   mov dx, offset NEWLINE
   mov ah, 9h
   int 21h

   pop dx
   pop ax
```

```
    ret

PRINT_NEWLINE endp

WRITE_STRING proc near

    push ax

    mov ah, 9h
    int 21h

    pop ax
    ret

WRITE_STRING endp


MY_INTERRUPT proc far

    jmp start_interrupt

    INTERRUPT_COUNTER db 'Calling counter is: 0000$'
    INTERRUPT_SIGN dw 7777h

    INTERRUPT_KEEP_IP     dw 0
    INTERRUPT_KEEP_CS     dw 0
    INTERRUPT_PSP_ADDRESS dw 0
    INTERRUPT_KEEP_SS     dw 0
    INTERRUPT_KEEP_SP     dw 0
    INTERRUPT_KEEP_AX     dw 0
    INTERRUPT_STACK dw 16 dup(?)
```

```
start_interrupt:
    mov INTERRUPT_KEEP_SP, sp
    mov INTERRUPT_KEEP_AX, ax
    mov ax, ss
    mov INTERRUPT_KEEP_SS, ax
    mov ax, INTERRUPT_KEEP_AX

    mov sp, offset start_interrupt
    mov ax, seg INTERRUPT_STACK
    mov ss, ax

    push ax
    push cx
    push dx


interrupt_handling:
    ; получаем инфу о курсоре
    mov ah, 03h
    mov bh, 0
    int 10h

    ; сохраняем текущее положение курсора
    push dx

    ; устанавливаем курсор в центр
    mov ah, 02h
    mov bh, 0ah
    mov dh, 0ch
    mov dl, 19h
```

```
    int 10h

    push si
    push cx
    push ds
    push bp

    mov ax, seg INTERRUPT_COUNTER
    mov ds, ax
    mov si, offset INTERRUPT_COUNTER
    add si, 20

    mov cx, 4

increment_loop:
    mov bp, cx
    mov ah, [si + bp]
    inc ah

    mov [si + bp], ah
    cmp ah, 3ah
    jne output_message

    mov ah, '0'
    mov [si + bp], ah

    loop increment_loop

output_message:
    pop bp
    pop ds
    pop cx
```

```
    pop si

    push es
    push bp

    mov ax, seg INTERRUPT_COUNTER
    mov es, ax
    mov ax, offset INTERRUPT_COUNTER
    mov bp, ax

    ; выводим счетчик на экран
    mov ah, 13h
    ; просим курсор не сдвигаться
    mov al, 0h
    ; длина строки
    mov cx, 24
    ; номер страницы
    mov bh, 0
    int 10h

    pop bp
    pop es

    pop dx
    mov ah, 02h
    mov bh, 0h
    int 10h

exit_interrupt:
    pop dx
    pop cx
    pop ax
```

```
    mov INTERRUPT_KEEP_AX, ax
    mov sp, INTERRUPT_KEEP_SP
    mov ax, INTERRUPT_KEEP_SS
    mov ss, ax
    mov ax, INTERRUPT_KEEP_AX

    mov al, 20h
    out 20h, al
    iret


INTERRUPT_SIZE:
MY_INTERRUPT endp


CHECK_ALREADY_LOADED proc near

    push ax
    push dx
    push es
    push si

    mov cl, 0ah
    mov ah, 35h
    mov al, 1ch
    int 21h

    mov cl, 0
    mov si, offset INTERRUPT_SIGN
    sub si, offset MY_INTERRUPT
    mov dx, es:[bx + si]
```

```
    cmp dx, INTERRUPT_SIGN
    jne check_loaded_exit

    mov cl, 1h

check_loaded_exit:
    pop si
    pop es
    pop dx
    pop ax
    ret

CHECK_ALREADY_LOADED endp



LOAD_INTERRUPT proc near
    push ax
    push es
    push bx
    push dx

    ; сначала проверяем, установлено ли прерывание в 1ch
    call CHECK_ALREADY_LOADED
    cmp cl, 1
    je int_already_exists

    ; сохраняем информацию об изначальном прерывании
    mov INTERRUPT_PSP_ADDRESS, es
    mov ah, 35h
    mov al, 1ch
    int 21h
```

```
mov INTERRUPT_KEEP_CS, es
mov INTERRUPT_KEEP_IP, bx


; загружаем прерывание
push ds
mov dx, offset MY_INTERRUPT
mov ax, seg MY_INTERRUPT
mov ds, ax
mov ah, 25h
mov al, 1ch
int 21h
pop ds

; оповещаем о том, что все ок
mov dx, offset INT_LOADED_SUCCESS
call WRITE_STRING

; остаемся резидентными
mov dx, offset INTERRUPT_SIZE   ; LEA DX, INTERRUPT_SIZE?
mov cl, 4
shr dx, cl

inc dx
add dx, 100h

xor ax, ax
mov ah, 31h
int 21h


jmp load_int_exit
```

```
int_already_exists:
    mov dx, offset INT_ALREADY_LOADED
    call WRITE_STRING

load_int_exit:
    pop dx
    pop bx
    pop es
    pop ax

    ret

LOAD_INTERRUPT endp


UNLOAD_INTERRUPT proc near
    push ax
    push si
    push dx

    ; проверяем, установлено ли прерывание
    call CHECK_ALREADY_LOADED
    cmp cl, 0
    je interrupt_is_not_loaded


    ; отключаем прерывания
    cli

    push ds
```

```
push es

; достаем адрес текущего загруженного прерывания
mov ah, 35h
mov al, 1ch
int 21h

; достаем из загруженного прерывания инфу о предыдущем прерывании
mov si, offset INTERRUPT_KEEP_IP
sub si, offset MY_INTERRUPT
mov dx, es:[bx + si]
mov ax, es:[bx + si + 2]
mov ds, ax

; заменяем текущее прерывание тем прерыванием, которое он заменил
mov ah, 25h
mov al, 1ch
int 21h

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
```

```
    pop es
    pop ds

    ; включаем прерывания обратно
    sti


    mov dx, offset INT_IS_RESTORED_SUCCECCFULLY
    call WRITE_STRING

    jmp unload_int_exit

interrupt_is_not_loaded:
    mov dx, offset INT_IS_NOT_LOADED
    call WRITE_STRING


unload_int_exit:
    pop dx
    pop si
    pop ax
    ret

UNLOAD_INTERRUPT endp


CHECK_INPUT proc near

    push ax
    push bx
    push es
```

```
    mov ah, 62h
    int 21h

    ; в bx - адрес начала PSP
    mov es, bx
    mov al, es:[80h]
    cmp al, 4
    ; если количество символов != 3, то нужно загрузить прерывание
    jne interrupt_set_label

    ; иначе, если было передано \un, то выгружаем
    mov al, es:[82h]
    cmp al, '\'
    jne interrupt_set_label
    mov al, es:[83h]
    cmp al, 'u'
    jne interrupt_set_label
    mov al, es:[84h]
    cmp al, 'n'
    jne interrupt_set_label

    call UNLOAD_INTERRUPT
    jmp check_input_exit


interrupt_set_label:
    call LOAD_INTERRUPT

check_input_exit:
    pop es
    pop bx
    pop ax
```

```
    ret

CHECK_INPUT endp


Main proc far
    mov ax, DATA
    mov ds, ax

    ; проверка аргументов коммандной строки
    call CHECK_INPUT


    ; выход в DOS
    xor al, al
    mov ah, 4ch
    int 21h
Main endp


CODE  ENDS
END Main
```