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. 9381 _____ Любимов В.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

Цель работы

В архитектуре компьютера существуют стандартные прерывания, за которыми закреплены определенные вектора прерываний. Вектор прерываний хранит адрес подпрограммы обработчика прерываний. При возникновении прерывания, аппаратура компьютера передаёт управление и выполняет соответствующие действия. В этой лабораторной работе предлагается построить обработчик прерываний сигналов таймера. Эти сигналы генерируются аппаратурой через определённые интервалы времени и, при возникновении такого сигнала, возникает прерывание с определённым значением вектора. Таким образом, управление будет передано функции, чья точка входа записана в соответствующий вектор прерывания.

Описание функций и структуры данных

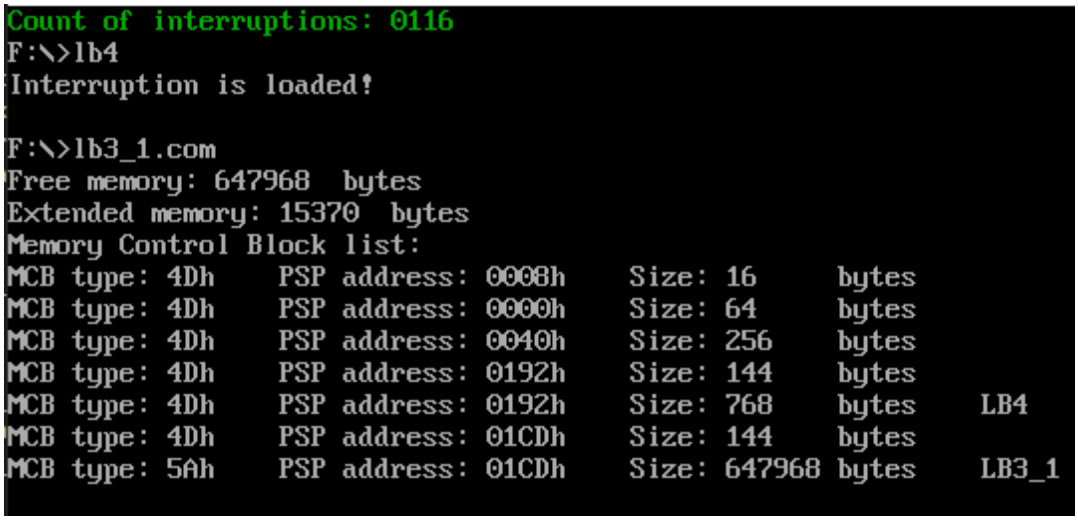
1. MY_INTERRUPTION– реализация требуемого прерывания.
2. IS_INTER_WORKING – проверяет, выполняется ли наше прерывание.
3. UNLOAD_INTER – выгружает наше прерывание и восстанавливает исходное состояние обработчика.
4. LOAD_INTER – загружает наше прерывание.
5. PRINT_MES – при помощи функции 9h из прерывания 21h выводит строку на экран.
6. IS_END_CMD – проверяет условие выгрузки нашего прерывания, то есть была ли программа запущена с флагом ‘/un’.

Ход выполнения работы

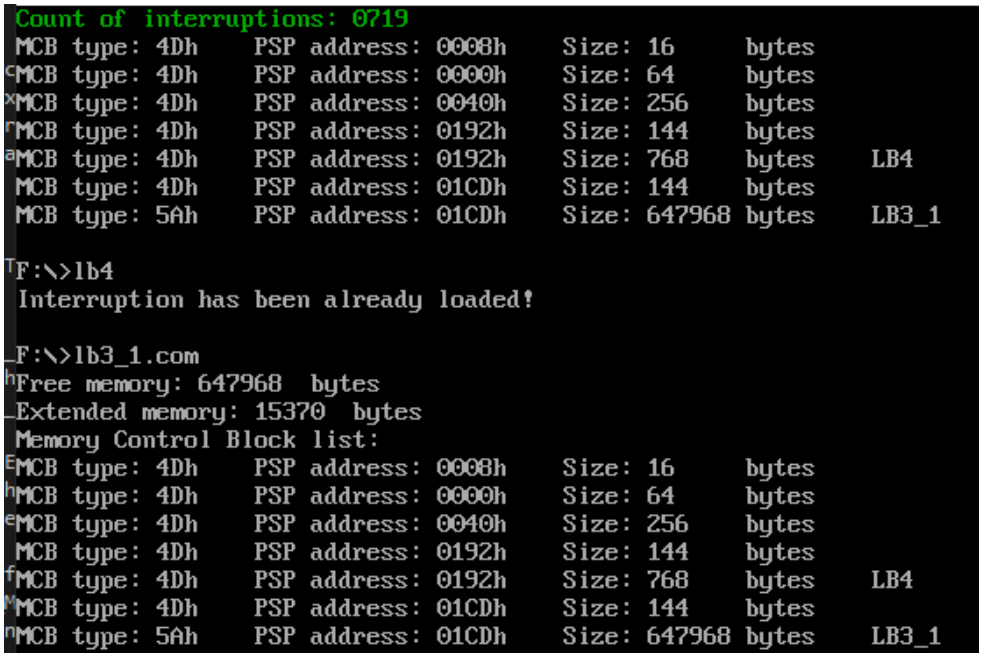
1. Разработан и написан .EXE модуль, реализующий следующие возможности:
    - Проверка установки пользовательского прерывания с вектором 1Ch.

- Если прерывание не установлено, то устанавливается резидентная функция для обработки прерывания и настраивает вектор прерываний. Производится выход по функции 4Ch прерывания 21h.
- Если прерывание установлено, то выводится сообщение об этом и производится выход по функции 4Ch прерывания 21h.
- Если программа запущена с флагом /un, то пользовательское прерывание выгружается, то есть восстанавливается стандартный вектор прерываний и освобождается память, занятая резидентом. Производится выход по функции 4Ch прерывания 21h.

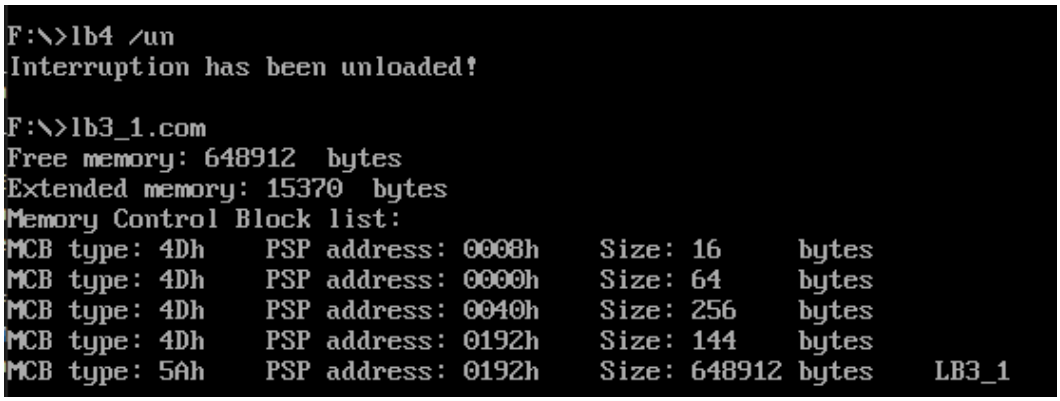
2. Запускаем программу. Прерывание загружается и выводится строка с количеством вызовов прерывания. Также выведен список блоков MCB.



3. Запускаем программу повторно. Выведено сообщение о том, что прерывание уже загружено. Обработчик прерываний корректно определяется, то есть остался резидентным в памяти.



4. Программа запущена с ключом выгрузки /un. Выведенный список блоков управления памятью показывает, что обработчик прерывания выгружен и память освобождена. Счётчик более не отображается.



Ответы на контрольные вопросы

1. Как реализован механизм прерывания от часов?

1. Поступает сигнал прерывания, который является периодическим

2. Значения регистров сохраняются

3. В таблице векторов устанавливается смещение по номеру источника прерывания

4. Сохраняем это смещение в CS:IP

5. Выполняем прерывание

6. Восстанавливаем данные прерванного процесса

7. Возвращаем управление прерванной программе

2. Какого типа прерывания использовались в работе?

Аппаратное прерывание от часов (1Ch). Программные прерывания функций BIOS (10h) и функций DOS (21h).

Вывод

В ходе выполнения работы была написана программа обработчика прерывания от сигналов таймера и изучен процесс установки резидентной программы в память и её выгрузка из памяти.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

lab4.asm

```
MYSTACK SEGMENT STACK
       DW 64 DUP(?)
MYSTACK ENDS

CODE SEGMENT
       ASSUME CS:CODE, DS:DATA, SS:MYSTACK


MY_INTERRUPTION PROC FAR
       jmp begin_inter

       keep_cs dw 0
       keep_ip dw 0
       psp_state dw 0
       psp_adress dw 0
       interruotion_set dw 0fedch
       keep_ss dw 0
       keep_sp dw 0
       keep_ax dw 0
       inter_amount_mes db 'Count of interruptions: 0000  $'
       new_stack dw 64 DUP(?)

begin_inter:
       mov    keep_sp, sp
    mov    keep_ax, ax
       mov    keep_ss, SS
    mov    sp, offset begin_inter
    mov    ax, seg new_stack
    mov    SS, ax

       push   ax
       push   bx
       push   cx
       push   dx

       mov    ah, 03h
       mov    bh, 0h
       int    10h

       push   dx

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

       push   si
       push   cx
       push   DS

       mov    ax, seg inter_amount_mes
       mov    DS, ax
       mov    si, offset inter_amount_mes
       add    si, 27
       mov    cx, 4

make_inter:
```

```
        mov     ah, [si]
        inc     ah
        mov     [si], ah
        cmp     ah, 3ah
        jne     get_amount
        mov     ah, 30h
        mov     [si], ah
        dec     si
        loop    make_inter

get_amount:
        pop     ds
        pop     cx
        pop     si

        push    es
        push    bp
        mov     ax, seg inter_amount_mes
        mov     es, ax
        mov     ax, offset inter_amount_mes
        mov     bp, ax
        mov     ah, 13h
        mov     al, 00h
        mov     cx, 28
        mov     bh, 0
        int     10h
        pop     bp
        pop     es

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

        pop     dx
        pop     cx
        pop     bx
        pop     ax

        mov     ss, keep_ss
        mov     ax, keep_ax
        mov     sp, keep_sp
        iret
MY_INTERRUPTION ENDP

IS_INTER_WORKING PROC NEAR
        push    bx
        push    dx
        push    es

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

        mov     dx, es:[bx + 11]
        cmp     dx, 0FEDCh
        je      is_working
        mov     al, 00h
        jmp     end_check

is_working:
        mov al, 01h

end_check:
        pop es
```

```
        pop dx
        pop bx
        ret
IS_INTER_WORKING ENDP

MEMORY_PROC:

IS_END_CMD PROC NEAR
        push   es

        mov    ax, psp_state
        mov    es, ax

        mov    al, es:[81h+1]
        cmp    al, '/'
        jne    wrong_cmd

        mov    al, es:[81h+2]
        cmp    al, 'u'
        jne    wrong_cmd

        mov    al, es:[81h+3]
        cmp    al, 'n'
        jne    wrong_cmd

        mov    al, 0001h
wrong_cmd:
        pop es
        ret
IS_END_CMD ENDP

UNLOAD_INTER PROC NEAR
        push   ax
        push   bx
        push   dx
        push   es

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

        push   ds

        mov    dx, es:[bx + 5]
        mov    ax, es:[bx + 3]
        mov    ds, ax
        mov    ah, 25h
        mov    al, 1ch
        int    21h
        pop    ds
        sti

        mov    dx, offset unload_mes
        call   PRINT_MES

        push   es
        mov    cx, es:[bx + 7]
        mov    es, cx
        mov    ah, 49h
        int    21h

        pop    es
        mov    cx, es:[bx + 9]
        mov    es, cx
        int    21h
```

```
        pop es

        pop dx
        pop bx
        pop ax
        ret
UNLOAD_INTER ENDP

LOAD_INTER PROC NEAR
        push   ax
        push   bx
        push   dx
        push   es

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

        mov    keep_ip, bx
        mov    keep_cs, es

        push   ds
        mov    dx, offset MY_INTERRUPTION
        mov    ax, seg MY_INTERRUPTION
        mov    ds, ax
        mov    ah, 25h
        mov    al, 1ch
        int    21h
        pop    ds

        mov    dx, offset load_mes
        call   PRINT_MES

        pop    es
        pop    dx
        pop    bx
        pop    ax
        ret
LOAD_INTER ENDP

PRINT_MES PROC NEAR
        push   ax
        mov    ah, 09h
        int       21h
        pop    ax
        ret
PRINT_MES ENDP

MAIN PROC FAR
        mov    bx, 02ch
        mov    ax, [bx]
        mov    psp_adress, ax
        mov    psp_state, DS
        xor    ax, ax
        xor    bx, bx

        mov    ax, DATA
        mov    DS, ax

        call   IS_END_CMD
        cmp    al, 01h
        je     unload_start

        call   IS_INTER_WORKING
        cmp    al, 01h
```

```
        jne     no_inter

        mov     dx, offset already_load_mes
        call    PRINT_MES
        jmp     main_end

        mov     ah,4ch
        int     21h

no_inter:
        call    LOAD_INTER

        mov     dx, offset MEMORY_PROC
        mov     cl, 04h
        shr     dx, cl
        add     dx, 1bh
        mov     ax, 3100h
        int     21h

unload_start:
        call    IS_INTER_WORKING
        cmp     al, 00h
        je      no_set_inter
        call    UNLOAD_INTER
        jmp     main_end

no_set_inter:
        mov     dx, offset no_inter_loaded_mes
        call    PRINT_MES

main_end:
        mov     ah, 4ch
        int     21h
MAIN ENDP

CODE ENDS

DATA SEGMENT
        no_inter_loaded_mes db "Interruption is not loaded!", 0DH, 0AH, '$'
        unload_mes db "Interruption has been unloaded!", 0DH, 0AH, '$'
        already_load_mes db "Interruption has been already loaded!", 0DH, 0AH, '$'
        load_mes db "Interruption is loaded!", 0DH, 0AH, '$'
DATA ENDS

         END MAIN
```