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. 0382 Самулевич В.А.

Преподаватель Ефремов М.А.

Санкт-Петербург
2022

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 передает управление по соответствующему адресу вектора прерывания. Обработчик прерываний получает управление и выполняет соответствующие действия.

В лабораторной работе № 4 предлагается построить обработчик прерываний сигналов таймера. Эти сигналы генерируются аппаратурой через определенные интервалы времени и, при возникновении такого сигнала, возникает прерывание с определенным значением вектора. Таким образом, управление будет передано функции, чья точка входа записана в соответствующий вектор прерывания.

Задание.

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить 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 осуществляется 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 освобождении памяти, занимаемой резидентом. Затем осуществляется выход по функции 4Ch прерывания int 21h.

Для того, чтобы проверить установку прерывания, можно поступить следующим образом. Прочитать адрес, записанный в векторе прерывания.

Предположим, что этот адрес указывает на точку входа в установленный резидент. На определенном, известном смещении в теле резидента располагается сигнатура, некоторый код, который идентифицирует резидент. Сравнив известное значение сигнатуры с реальным кодом, находящимся в резиденте, можно определить, установлен ли резидент. Если значения совпадают, то резидент установлен. Длину кода сигнатуры должна быть достаточной, чтобы сделать случайное совпадение маловероятным. Программа должна содержать код устанавливаемого прерывания в виде удаленной процедуры. Этот код будет работать после установки при возникновении прерывания. Он должен выполнять следующие функции:

1) Сохраняет стек прерванной программы (регистры SS и SP) в рабочих переменных и восстановить при выходе.

2) Организовать свой стек.

3) Сохранить значения регистров в стеке при входе и восстановить их при выходе.

4) При выполнении тела процедуры накапливать общее суммарное число прерываний и выводить на экран. Для вывода на экран следует использовать прерывание int 10h, которое позволяет непосредственно выводить информацию на экран.

5) Функция прерывания должна содержать только переменные, которые она использует

Шаг 2. Запустите отлаженную программу и убедитесь, что резидентный обработчик прерывания 1Ch установлен. Работа прерывания должна отображаться на экране, а также необходимо проверить размещение прерывания в памяти. Для этого запустите программу ЛР 3, которая отображает 4 карту памяти в виде списка блоков MCB. Полученные результаты поместите в отчет.

Шаг 3. Запустите отлаженную программу еще раз и убедитесь, что программа определяет установленный обработчик прерываний. Полученные результаты поместите в отчет.

Шаг 4. Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

Выполнение работы.

В процессе написания соответствующего .exe модуля были реализованы следующие функции:

- Select_mode - определяет режим, в котором должна работать программа (загрузка или выгрузка пользовательского прерывания) и, в зависимости от результата, вызывает load_interrupt (аргументы командной строки отсутствуют) или unload_interrupt (передан аргумент \un).
- load_interrupt- устанавливает резидентную функцию User_interrupt в качестве обработчика прерывания сигналов таймера. В случае, если эта функция уже была установлена, выводится соответствующее сообщение.
- unload_interrupt – Восстанавливает стандартный обработчик прерывания для таймера и освобождает память, занимаемую пользовательским резидентом. Если стандартный обработчик уже установлен и ничего изменять не требуется, выводится соответствующее сообщение.
- User_interrupt – созданный обработчик, который накапливает суммарное число своих вызовов и выводит его на экран.

Результат работы написанной программы, вместе с картой памяти в виде списка блоков MCB, представлен на рисунке 1.

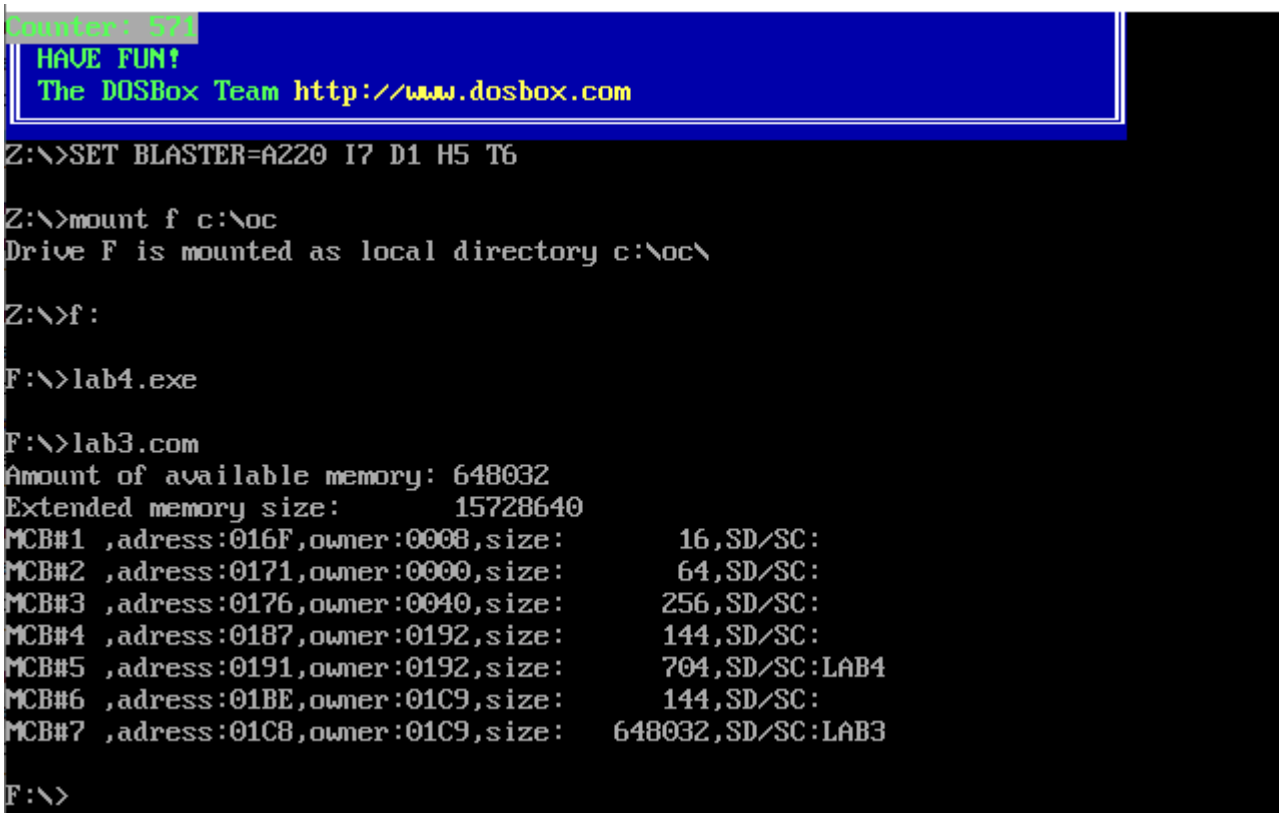

Рисунок 1 – результат работы написанной программы вместе с картой памяти.

По счетчику в левом верхнем углу, а также по блоку MCB#5, который хранит резидентный обработчик, можно понять, что установка User_interruption прошла успешно.

Результат повторного запуска lab4.exe представлен на рисунке 2.

Рисунок 2 – результат повторного запуска программы.

Т.к. пользовательский обработчик уже был загружен, выводится соответствующее сообщение, и программа завершается.

Результат запуска lab4.exe с ключом выгрузки /un, а также обновленная карта памяти представлена на рисунке 3.

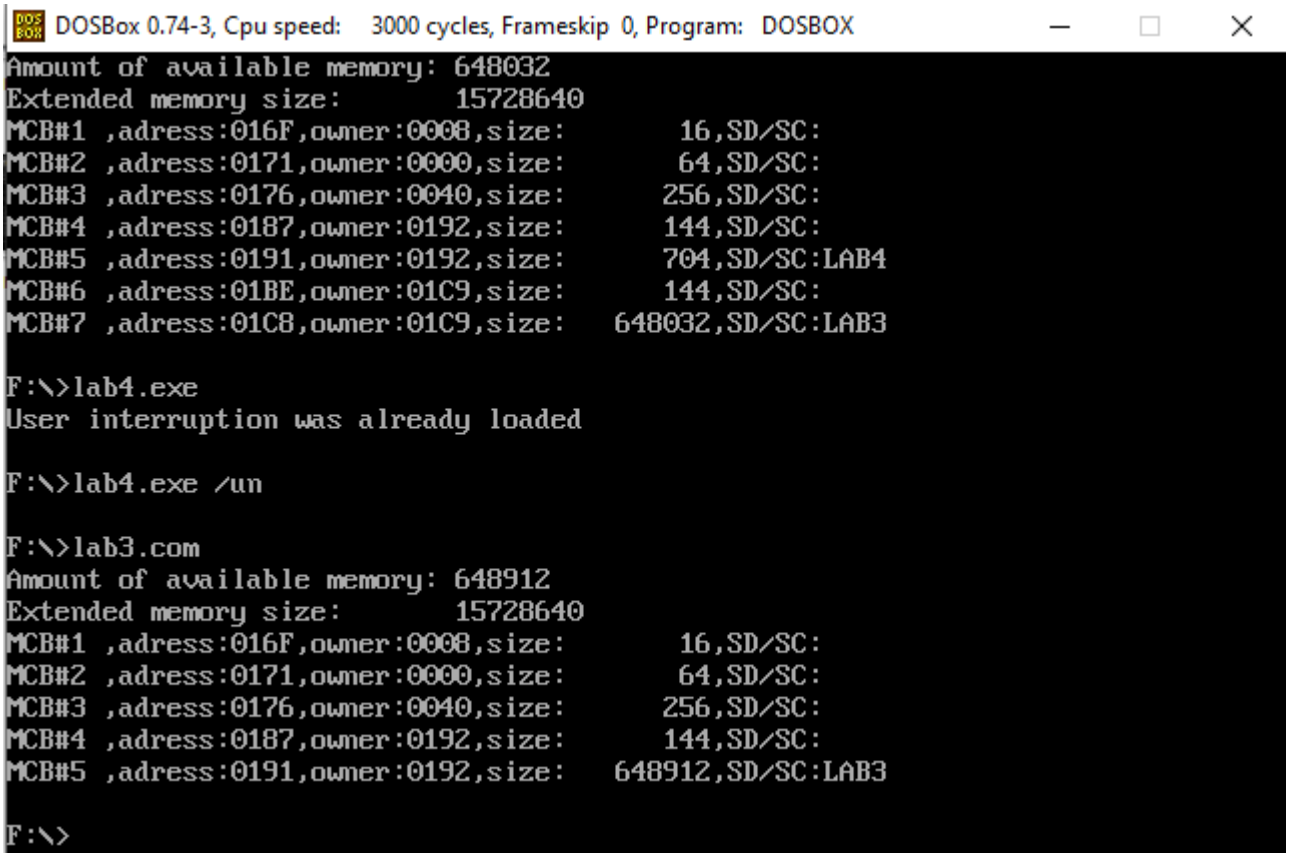

Рисунок 3- результат работы программы с ключем выгрузки /un, вместе с обновленным списком блоков MCB.

Исчезновение с экрана таймера, а также отсутствие блоков MCB, которые принадлежали бы lab4, говорит о том, что выгрузка обработчика и освобождение памяти прошли успешно.

Контрольные вопросы.

1) Как реализован механизм прерывания от часов?

   Аппаратура компьютера генерирует специальный сигнал, 18 раз в секунду. После каждого сигнала вызывается прерывание системного таймера (1ch).

2) Какого типа прерывания использовались в работе?

   Программные (21h, 10h) и аппаратные(1ch).

Выводы.

Было произведено исследование структуры обработчиков стандартных прерываний, был построен обработчик прерываний сигналов таймера.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab3.asm

```
AStack SEGMENT STACK
    DW 12 DUP(?)
AStack ENDS

DATA SEGMENT
    FLAG DB "/un",0DH
    already_loaded_message DB "User interruption was already loaded",
0Dh,0AH,'$'
    already_unloaded_message DB "user interrupt not set",0Dh,0Ah,'$'

DATA ENDS

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

User_interrupt PROC FAR

jmp interrupt_start

    SIGNATURE DW 5555
    KEEP_PSP DW 0
    KEEP_SS DW 0
    KEEP_SP DW 0
    KEEP_CS DW 0
    KEEP_IP DW 0
    OUTPUT DB "Counter:    "
    INT_STACK DW 100 DUP(0)

interrupt_start:
    mov KEEP_SS, SS
    mov KEEP_SP,SP
    mov SP, SEG INT_STACK
    mov SS,SP
    mov SP,offset interrupt_start

    push AX
    push BX
    push CX
    push DX
    push ES
    push BP

    mov SI,offset OUTPUT
    add SI,11

inc_loop:
    mov AL,CS:[SI]
    cmp AL,32
    jne skip_transform
    mov AL, 48
```

```
skip_transform:
    cmp AL, 57
    jne inc_end
    mov BYTE PTR CS:[SI],48
    dec SI
    jmp inc_loop

inc_end:
    inc AL
    mov CS:[SI],AL


    mov AH,13h
    mov AL,0
    mov BH,0
    mov BL,122
    mov CX,12
    mov DX,0
    mov BP,SEG OUTPUT
    mov ES,BP
    mov BP,offset OUTPUT
    int 10h

    pop BP
    pop ES
    pop DX
    pop CX
    pop BX
    pop AX

    mov SS,KEEP_SS
    mov SP,KEEP_SP
    mov AL,20h
    out 20h,AL
    iret
User_interrupt ENDP


Main PROC FAR
    push DS
    sub AX,AX
    push AX
    mov AX,DATA
    mov DS,AX
    mov WORD PTR KEEP_PSP,ES
    call Select_mode

    ret
Main ENDP

Select_mode PROC NEAR
    push AX
    push SI
    push DI

    mov AL,32
    mov DI,081h
```

```
skip_spaces:
    SCASB
    je skip_spaces

    dec DI
    mov SI,offset  FLAG

cmp_loop:
    mov AL, ES:[DI]
    cmp AL, DS:[SI]
    jne load_mode
    cmp AL,0dh
    je unload_mode
    inc SI
    inc DI
    jmp cmp_loop

load_mode:
    call load_interrupt
    jmp select_mode_end

unload_mode:
    call unload_interrupt

select_mode_end:

    pop DI
    pop SI
    pop AX

    ret
Select_mode ENDP

load_interrupt PROC NEAR
    push SI
    push AX
    push BX
    push CX
    push DX
    push ES

    mov AH,35h
    mov AL,1ch
    int 21h
    mov SI,offset SIGNATURE
    mov AX, ES:[SI]
    cmp AX,5555
    je already_loaded
    mov KEEP_IP, BX
    mov KEEP_CS, ES

    push DS
    mov DX, offset User_interrupt
    mov AX, seg User_interrupt
    mov DS, AX
    mov AH, 25H
    mov AL, 1CH
```

```
    int 21h
    pop DS
    mov DX, offset Main
    mov CL, 4
    shr DX, CL
    inc DX
    mov AX, CS
    sub AX, KEEP_PSP
    add DX, AX
    mov AX,0
    mov AH, 31h
    int 21h
    jmp load_end
already_loaded:
    mov DX, offset already_loaded_message
    call print
load_end:

    pop ES
    pop DX
    pop CX
    pop BX
    pop AX
    pop SI
    ret
load_interrupt ENDP



unload_interrupt PROC NEAR
    push AX
    push BX
    push DX
    push ES
    push SI


    mov AH,35h
    mov AL,1ch
    int 21h
    mov SI,offset SIGNATURE
    mov AX, es:[SI]
    cmp AX,5555
    jne already_unloaded
    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 SI,offset KEEP_PSP
    mov AX,ES:[SI]
    mov ES,AX
    push ES
    mov AX,ES:[2ch]
```

```
        mov ES,AX
        mov AH,49h
        int 21h
        pop ES
        int 21h
        STI
        jmp unload_end

    already_unloaded:
        mov DX,offset already_unloaded_message
        call print
    unload_end:
        pop SI
        pop ES
        pop DX
        pop BX
        pop AX
        ret
    unload_interrupt ENDP


    print PROC NEAR
        push AX
        mov AH,09h
        int 21h
        pop AX
        ret
    print ENDP


    CODE ENDS
END Main
```