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 _____ Гладких А.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

ПОСТАНОВКА ЗАДАЧИ

Цель работы.

Применить теоретические знания о работе обработчика прерываний. Построить обработчик прерываний сигналов таймера.

Сведения о функциях и структурах данных управляющей программы.

Процедура	Описание
WRITEWRD	Функция печати строки на экран
WRITEBYTE	Функция печати символа на экран
ENDLINE	Функция печати символов переноса строки
OUTPUTAL	Вывод на экран содержимого регистра AL
OUTPUTBP	Вывод на экран содержимого регистра BP
SETCURSORFORINT	Функция установки курсора в нужную позицию
GETCURSOR	Функция получения позиции курсора
MY_INTERRUPTION	Функция прерывания
CHECK_CLI_OPT	Проверка наличия параметров командной строки
CHECK_LOADED	Проверка загрузки пользовательского прерывания
LOAD_INTERRUPTION	Функция загрузки пользовательского прерывания в таблицу прерываний
UNLOAD_INTERRUPTION	Функция возвращения стандартного прерывания
MAIN	Главная функция программы

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 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.

Шаг 5. Ответьте на контрольные вопросы.

Исходный код.

Исходный код представлен в приложении А.

РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ ПРОБЛЕМ

Шаг 1. Был написан и отлажен программный модуль типа .EXE, который выполняет поставленные в задании функции.

Шаг 2. Написанный модуль был отлажен и запущен. Резидентный обработчик прерываний был установлен и размещен в памяти.

```
Interruption counter: 0985
D:\>lab.exe
Interruption is loaded successfully

D:\>lab3.com
Available Memory (bytes):644336
Extended Memory (kbytes):15360

MCB Table:
MCB #1 Address: 016F  PSP TYPE: belongs MSDOS       Size: 0001  SC/SD:
MCB #2 Address: 0171  PSP TYPE: free area           Size: 0004  SC/SD:
MCB #3 Address: 0176  PSP TYPE: 0040                Size: 0010  SC/SD:
MCB #4 Address: 0187  PSP TYPE: 0192                Size: 0009  SC/SD:
MCB #5 Address: 0191  PSP TYPE: 0192                Size: 0113  SC/SD: LAB
MCB #6 Address: 02A5  PSP TYPE: 02B0                Size: 0009  SC/SD:
MCB #7 Address: 02AF  PSP TYPE: 02B0                Size: 004A  SC/SD: LAB3
MCB #8 Address: 02FA  PSP TYPE: free area           Size: 9D04  SC/SD: î♦v .ë▲
```

Рисунок 1 - Иллюстрация работы .EXE-модуля и корректной установки и размещения прерывания

Шаг 3. Программа корректно определяет установленный обработчик прерываний.

```
DOSBox 0.74-3, Cpu speed:   3000 cycles, Frameskip  0, Program:  DOSBOX
Interruption counter: 5399
D:\>lab.exe
Interruption is loaded successfully

D:\>lab3.com
Available Memory (bytes):644336
Extended Memory (kbytes):15360

MCB Table:
MCB #1 Address: 016F  PSP TYPE: belongs MSDOS       Size: 0001  SC/SD:
MCB #2 Address: 0171  PSP TYPE: free area           Size: 0004  SC/SD:
MCB #3 Address: 0176  PSP TYPE: 0040                Size: 0010  SC/SD:
MCB #4 Address: 0187  PSP TYPE: 0192                Size: 0009  SC/SD:
MCB #5 Address: 0191  PSP TYPE: 0192                Size: 0113  SC/SD: LAB
MCB #6 Address: 02A5  PSP TYPE: 02B0                Size: 0009  SC/SD:
MCB #7 Address: 02AF  PSP TYPE: 02B0                Size: 004A  SC/SD: LAB3
MCB #8 Address: 02FA  PSP TYPE: free area           Size: 9D04  SC/SD: î♦v .ë▲

D:\>lab.exe
Interruption is already loaded
```

Рисунок 2 - Иллюстрация корректного определения установленного обработчика прерываний

Шаг 4. Была запущена отлаженная программа с ключом выгрузки. Резидентный обработчик был выгружен, сообщения прерывания также перестали выводиться. Память была успешна освобождена.

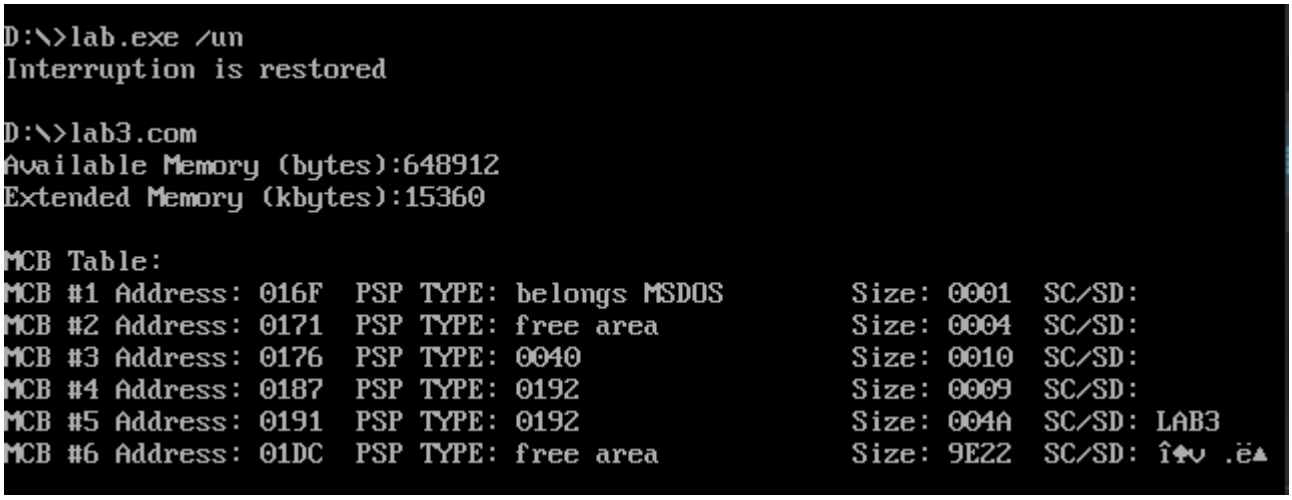

Рисунок 3 - Иллюстрация корректной выгрузки обработчика прерывания

Шаг 5. Была произведена оценка результатов и были отвечены контрольные вопросы:

1. ***Как реализован механизм прерывания от часов?***

   Ответ: прерывание от часов срабатывает примерно 18 раз в секунду. Вызов прерывания происходит с помощью аппаратно генерируемого прерывания int 08h – системного таймера. При вызове прерывания сохраняются регистры CS, IP для последующего возвращения в программу, а затем определяется адрес вызываемого вектора прерывания в таблице прерываний. Адрес помещается в регистры CS и IP. Затем программа передает управление по адресу CS:IP. По завершении работы прерывания программа сообщает системе, что прерывание от времени закончено, посылая сигнал конец-прерывания контроллеру прерываний, и восстанавливает регистры. После этого управление возвращается прерванной программе.

2. ***Какого типа прерывания использовались в работе?***

Ответ: были использованы: 1Ch – аппаратное прерывание, 10h, 21h – программные.

Выводы.

Были применены теоретические знания о работе обработчика прерываний. Был построен и отлажен обработчик прерываний сигналов таймера, который считает количество вызовов и выводит соответствующую информацию на экран.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab.asm

```
ASTACK SEGMENT STACK
   DW 200 DUP(?)
ASTACK ENDS

DATA SEGMENT
        interruption_already_loaded_string db 'Interruption is already loaded', 0DH, 0AH, '$'
        interruption_loaded_successfully_string db 'Interruption is loaded successfully', 0DH, 0AH, '$'
        interruption_not_loaded_string db 'Interruption is not loaded', 0DH, 0AH, '$'
        interruption_restored_string db 'Interruption is restored', 0DH, 0AH, '$'
        test_string db 'test', 0DH, 0AH, '$'
DATA ENDS

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

    WRITEWRD  PROC  NEAR
        push ax
        mov ah, 9
        int 21h
        pop ax
        ret
    WRITEWRD  ENDP

    WRITEBYTE  PROC  NEAR
        push ax
        mov ah, 02h
        int 21h
        pop ax
        ret
    WRITEBYTE  ENDP
```

```
ENDLINE PROC NEAR
    push ax
    push dx

    mov dl, 0dh
    call WRITEBYTE

    mov dl, 0ah
    call WRITEBYTE

    pop dx
    pop ax
    ret
ENDLINE ENDP

OUTPUTAL PROC NEAR
    push ax
    push bx
    push cx
    mov ah, 09h ;писать символ с 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 ; функция
    mov al, 0 ; sub function code
    ; 1 = use attribute in BL; leave cursor at end of string
    mov bh,0 ; видео страница
    mov dh,22 ; DH,DL = строка, колонка (считая от 0)
    mov dl,0
    int 10h
    pop CX
```

```
    pop dx
    pop bx
    pop ax
    ret
OUTPUTBP ENDP

    ; Установка позиции курсора
; установка на строку 25 делает курсор невидимым
SETCURSORFORINT PROC NEAR
    mov ah, 02h
    mov bh, 0h
    mov dh, 0h ; DH,DL = строка, колонка (считая от 0)
    mov dl, 0h
    int 10h ; выполнение.

    ret
SETCURSORFORINT ENDP

GETCURSOR PROC NEAR
    mov ah, 03h
    mov bh, 0
    int 10h
    ret
GETCURSOR ENDP

MY_INTERRUPTION PROC FAR
    jmp start

    int_counter_string db 'Interruption counter: 0000$'
    interruption_signature dw 7777h

    int_keep_ip dw 0
    int_keep_cs dw 0
    psp_address dw ?
    int_keep_ss dw 0
    int_keep_sp dw 0
    int_keep_ax dw 0
    IntStack dw 16 dup(?)

start:
    mov int_keep_sp, sp
    mov int_keep_ax, ax
```

```
    mov ax, ss
    mov int_keep_ss, ax

    mov ax, int_keep_ax

    mov sp, OFFSET start
    mov ax, seg IntStack
    mov ss, ax

    push ax ;сохранение изменяемого регистра
    push cx ;сохранение изменяемого регистра
    push dx ;сохранение изменяемого регистра

    ;Само прерывание

    call GETCURSOR ;DX = (ROW, COLUMN)

    push dx

    call SETCURSORFORINT

    push si
push cx
push ds
      push bp

    mov ax, SEG int_counter_string
  mov ds, ax
  mov si, offset int_counter_string
  add si, 21

        mov cx, 4

interruption_counter_loop:
    mov bp, cx
    mov ah, [si+bp]
    inc ah
    mov [si+bp], ah
    cmp ah, 3ah
    jne print_msg
    mov ah, 30h
    mov [si+bp], ah
```

```
        loop interruption_counter_loop

print_msg:

        pop bp
        pop ds
        pop cx
        pop si

  push es
  push bp

  mov ax, SEG int_counter_string
  mov es,ax
  mov ax, offset int_counter_string
  mov bp,ax
  mov ah, 13h ;Write Character String in any display page
  mov al, 00h ;do not update cursor
  mov cx, 26 ;length
  mov bh,0 ;page number
  int 10h

  pop bp
  pop es

  ;return cursor
  pop dx
  mov ah,02h
  mov bh,0h
  int 10h


    ;Конец прерывания

    pop dx ;восстановление регистра
    pop cx ;восстановление регистра
    pop ax ;восстановление регистра

    mov int_keep_ax, ax
    mov sp, int_keep_sp
    mov ax, int_keep_ss
```

```
    mov ss, ax
    mov ax, int_keep_ax

    mov al, 20h     ;разрешаем обработку прерываний
    out 20h, al     ;с более низкими уровнями
    iret ;конец прерывания

interruption_last_byte:
MY_INTERRUPTION ENDP

CHECK_CLI_OPT PROC near
        push ax
    push bp

    mov cl, 0h

    mov bp, 81h

        mov al,es:[bp + 1]
        cmp al,'/'
        jne lafin

        mov al,es:[bp + 2]
        cmp al,'u'
        jne lafin

        mov al,es:[bp + 3]
        cmp al,'n'
        jne lafin

        mov cl, 1h

lafin:
    pop bp
        pop ax
        ret
CHECK_CLI_OPT ENDP

CHECK_LOADED PROC NEAR
    push ax
    push dx
    push es
```

```
    push si

    mov cl, 0h

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

    mov si, offset interruption_signature
    sub si, offset MY_INTERRUPTION
    mov dx, es:[bx + si]
    cmp dx, interruption_signature
    jne checked

    mov cl, 1h ;already loaded

checked:
    pop si
    pop es
    pop dx
    pop ax
    ret
CHECK_LOADED ENDP

LOAD_INTERRUPTION PROC near
        push ax
    push cx
        push dx


        call CHECK_LOADED
        cmp cl, 1h
        je int_already_loaded

    mov psp_address, es

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

    mov int_keep_cs, es
mov int_keep_ip, bx
```

```
  push es
    push bx
        push ds

        lea dx, MY_INTERRUPTION
        mov ax, SEG MY_INTERRUPTION
        mov ds, ax

        mov ah, 25h
        mov al, 1ch
        int 21h

        pop ds
    pop bx
    pop es

    mov dx, offset interruption_loaded_successfully_string
        call WRITEWRD

        lea dx, interruption_last_byte
        mov cl, 4h
        shr dx, cl
        inc dx ;dx - size in paragraphs

        add dx, 100h

        xor ax,ax

        mov ah, 31h
        int 21h

    jmp fin_load_interruption

int_already_loaded:
  mov dx, offset interruption_already_loaded_string
    call WRITEWRD

fin_load_interruption:
        pop dx
    pop cx
        pop ax
```

```
        ret
LOAD_INTERRUPTION ENDP

UNLOAD_INTERRUPTION PROC near
        push ax
        push si

        call CHECK_LOADED
        cmp cl, 1h
        jne interruption_is_not_loaded

    cli

    push ds
    push es

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

    mov si, offset int_keep_ip
sub si, offset MY_INTERRUPTION
mov dx, es:[bx + si]
mov ax, es:[bx + si + 2]
      mov ds, ax

    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

    sti

    mov dx, offset interruption_restored_string
        call WRITEWRD



    jmp int_unloaded

interruption_is_not_loaded:
    mov dx, offset interruption_not_loaded_string
    call WRITEWRD

int_unloaded:
        pop si
        pop ax
        ret
UNLOAD_INTERRUPTION ENDP

MAIN PROC FAR
        mov   ax, DATA
        mov   ds, ax

    call CHECK_CLI_OPT
    cmp cl, 0h
    jne opt_unload

    call LOAD_INTERRUPTION
    jmp main_end

opt_unload:

    call UNLOAD_INTERRUPTION

main_end:
    xor al, al
    mov ah, 4ch
    int 21h
```

```
    MAIN ENDP


CODE ENDS

END MAIN
```