МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по практической работе № 4
по дисциплине «Операционные системы»
Тема: Обработка стандартных прерываний

Студент гр. 9383 _____ Рыбников Р.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

В архитектуре компьютера существуют стандартные прерывания, за которыми закреплены определённые вектора прерываний. Вектор прерываний хранит адрес подпрограммы обработчика прерываний. При возникновении прерывания, аппаратура компьютера передаёт управление по соответствующему адресу вектора прерывания . Обработчик прерываний получает управление и выполняет соответствующие действия.

В 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.

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 МСВ. 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.

Результаты исследования проблем.

Был написан и отлажен программный модуль типа .EXE, который выполняет, прописанные в задании функции.

Программа была отлажена и запущена. Проверено размещение прерывания в памяти.

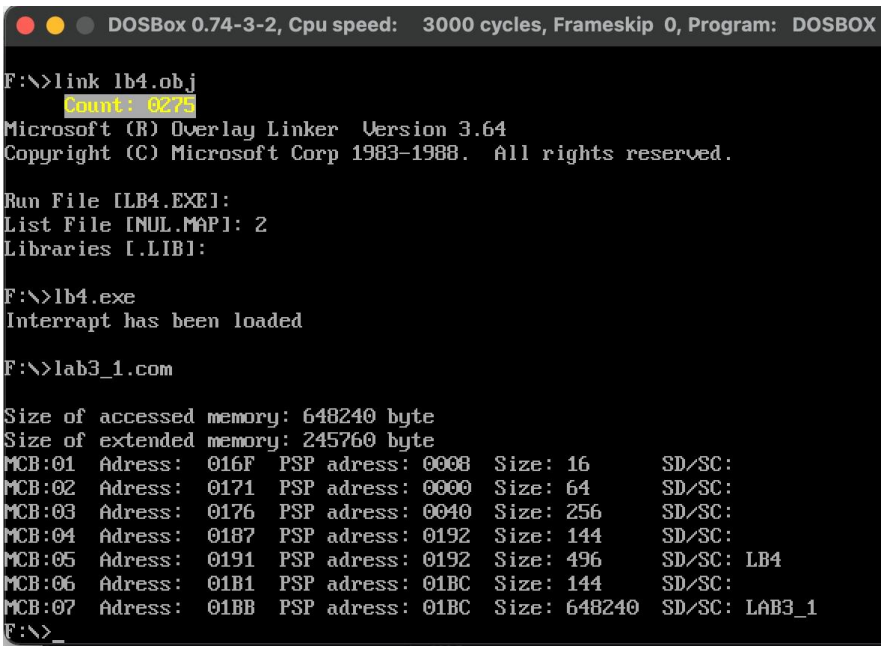

Рисунок 1

Программа была повторно запущена, чтобы удостовериться, что программа определяет установленный обработчик прерываний.

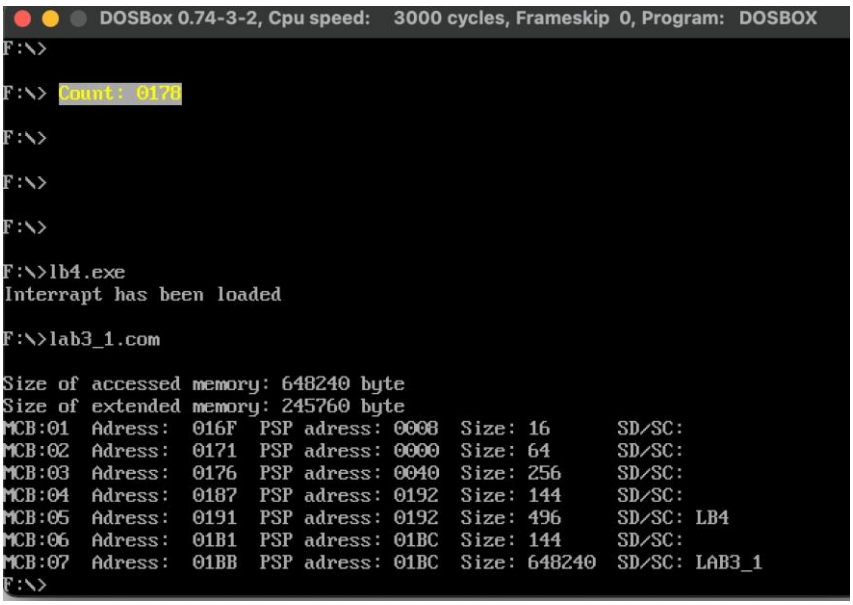

Рисунок 2

Программа была запущена с ключом выгрузки, чтобы удостовериться, что резидентный обработчик прерывания выгружен, и память, занятая резидентом освобождена.

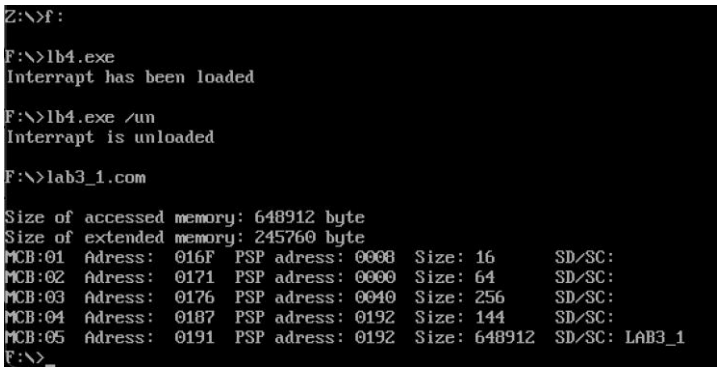

Рисунок 3

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как реализован механизм прерывания от часов?
   Ответ:

Принимается сигнал прерывания, запоминаются содержимые регистров, по номеру источника прерывания в таблице векторов определяется смещение (2 байта в IP и 2 байта в CS). Затем вызывается обработчик прерывания по сохраненному адресу. После управление передается обратно прерванной программе.

2. Какие прерывания использовались в работе?

Ответ:

1Ch – аппаратное прерывание, 10h и 21h – программное прерывание.

Выводы.

Был построено свой обработчик прерываний сигналов таймера.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

lb4.asm

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

;-------------------------------

MY_INTERRUPT PROC far
   jmp start_interrupt

   KEEP_PSP dw ?
   KEEP_IP dw 0
   KEEP_CS dw 0
   INTERRUPT_ID dw 6666h


   COUNTER db 'Count: 0000$' ; 6

   KEEP_AX dw ?
      KEEP_SS dw ?
      KEEP_SP dw ?
      INTERRUPT_STACK dw 32 dup (?)
      END_IT_STACK dw ?

start_interrupt:
   mov KEEP_SS,ss
   mov KEEP_SP,sp
   mov KEEP_AX,ax
```

```
mov ax,cs
mov ss,ax
mov sp,offset END_IT_STACK

push bx
push cx
push dx


   mov ah,3h
mov bh,0h
   int 10h
   push dx


   mov ah,02h
   mov bh,0h
mov dh,02h
mov dl,05h
   int 10h


push si
   push cx
push ds
push bp

   mov ax,SEG COUNTER
   mov ds,ax
   mov si,offset COUNTER
   add si,6 ; и вот тут колво чаров включая двоеточие
```

```
   mov cx,4
interrapt_loop:
   mov bp,cx
   mov ah,[si+bp]
      inc ah
      mov [si+bp],ah
      cmp ah,3Ah
      jl m_number
   mov ah,30h
   mov [si+bp],ah

   loop interrapt_loop

m_number:
   pop bp
   pop ds
   pop cx
   pop si


   push es
      push bp

      mov ax,SEG COUNTER
      mov es,ax
   mov ax,offset COUNTER
      mov bp,ax
      mov ah,13h
      mov al,00h
      mov cx,11 ; и вот тут прямо до конца строчки(вкл или не вкл $)
```

```
      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

      mov ax, KEEP_SS
      mov ss, ax
   mov ax, KEEP_AX
   mov sp, KEEP_SP

   iret
interrapt_end:
MY_INTERRUPT ENDP

;-------------------------------

WRITE_STRING PROC near
   push ax
   mov ah, 09h
   int 21h
```

```
   pop ax
   ret
WRITE_STRING ENDP
;-------------------------------

LOAD_UN PROC near
   push ax
   mov KEEP_PSP,es
   mov al,es:[81h+1]
   cmp al,'/'
   jne load_un_end
   mov al,es:[81h+2]
   cmp al, 'u'
   jne load_un_end
   mov al,es:[81h+3]
   cmp al, 'n'
   jne load_un_end
   mov flag,1h

load_un_end:
   pop ax
   ret
LOAD_UN ENDP

;-------------------------------

IS_LOAD PROC near
   push ax
   push si

   mov ah,35h
```

```
   mov al,1Ch
   int 21h
   mov si,offset INTERRUPT_ID
   sub si,offset MY_INTERRUPT
   mov dx,es:[bx+si]
   cmp dx, 6666h
   jne is_load_end
   mov flag_load,1h
is_load_end:
   pop si
   pop ax
   ret
IS_LOAD ENDP

;--------------------------------

LOAD_INTERRAPT PROC near
   push ax
   push dx

   call IS_LOAD
   cmp flag_load,1h
   je already_load
   jmp start_load

already_load:
   lea dx,STR_ALR_LOAD
   call WRITE_STRING
   jmp end_load

start_load:
```

```
    mov ah,35h
    mov al,1Ch
    int 21h
    mov KEEP_CS, es
       mov KEEP_IP, bx

    push ds
    lea dx, MY_INTERRUPT
    mov ax, seg MY_INTERRUPT
    mov ds,ax
    mov ah,25h
    mov al, 1Ch
    int 21h
    pop ds
    lea dx, STR_SUC_LOAD
    call WRITE_STRING

    lea dx, interrapt_end
    mov cl, 4h
    shr dx,cl
    inc dx
    mov ax,cs
    sub ax,KEEP_PSP
    add dx,ax
    xor ax,ax
    mov ah,31h
    int 21h

end_load:
    pop dx
    pop ax
```

```
   ret
LOAD_INTERRAPT ENDP

;-------------------------------

UNLOAD_INTERRAPT PROC near
   push ax
   push si

   call IS_LOAD
   cmp flag_load,1h
   jne cant_unload
   jmp start_unload

cant_unload:
   lea dx, STR_ISNT_LOAD
   call WRITE_STRING
   jmp unload_end


start_unload:
   CLI
   push ds
   mov ah,35h
      mov al,1Ch
      int 21h

   mov si,offset KEEP_IP
      sub si,offset MY_INTERRUPT
   mov dx,es:[bx+si]
   mov ax,es:[bx+si+2]
```

```
   MOV ds, ax
   MOV ah,25h
   MOV al, 1Ch
   INT 21h
   POP ds


   mov ax,es:[bx+si-2]
   mov es,ax
   push es

   mov ax,es:[2ch]
   mov es,ax
   mov ah,49h
   int 21h

   pop es
   mov ah,49h
   int 21h
   STI

   lea dx, STR_IS_UNLOAD
   call WRITE_STRING
unload_end:
   pop si
   pop ax
   ret
UNLOAD_INTERRAPT ENDP

;-------------------------------
; Головная процедура
```

```
Main      PROC  FAR
   push  ds
   xor   ax, ax
   push  ax
   mov   ax,DATA
   mov   ds, ax

   call LOAD_UN
   cmp flag, 1h
   je unload_interrapt_1
   call LOAD_INTERRAPT
   jmp end_1

unload_interrapt_1:
   call UNLOAD_INTERRAPT

end_1:
   mov ah, 4ch
   int 21h

Main      ENDP
CODE      ENDS

ASTACK    SEGMENT  STACK
   DW 64 DUP(?)
ASTACK    ENDS

DATA      SEGMENT
   flag db 0
   flag_load db 0
```

```
    STR_ISNT_LOAD  DB 'Interrapt is not load', 0AH, 0DH,'$'
    STR_ALR_LOAD  DB 'Interrapt is already loaded', 0AH, 0DH,'$'
    STR_SUC_LOAD  DB 'Interrapt has been loaded', 0AH, 0DH,'$'
    STR_IS_UNLOAD  DB 'Interrapt is unloaded', 0AH, 0DH,'$'
DATA      ENDS
          END Main
```