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 _____ Камзолов Н.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2021

Постановка задачи.

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 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.

Результаты исследования проблем.

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

Шаг 2. Программа была отлажена и запущена. Проверено размещение прерывания в памяти.

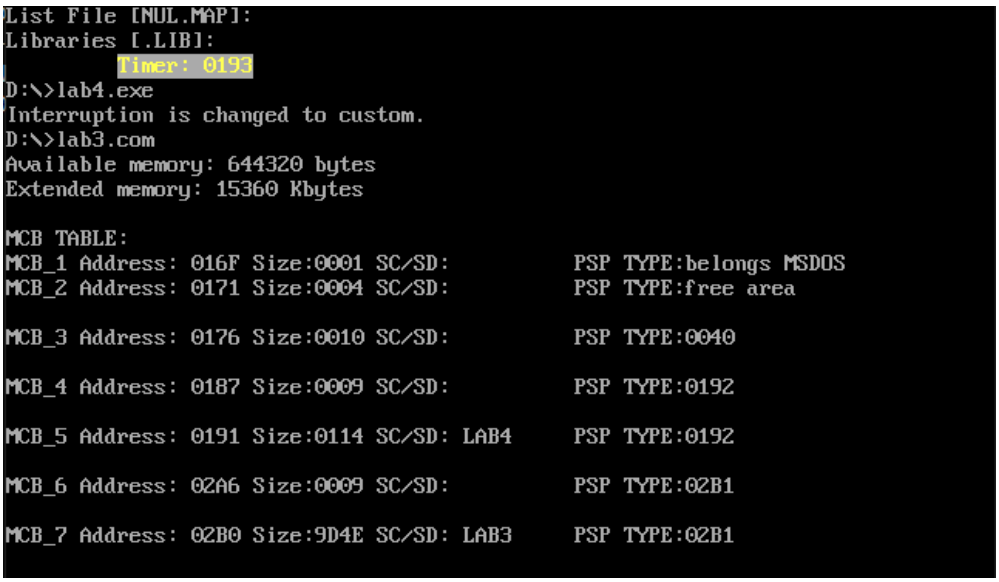

Рисунок 1 – Демонстрация корректной работы резидентного обработчика прерываний, а также его расположение в памяти, которое было получено с помощью программы, написанной в лабораторной работе номер 3 (сведения о прерывании находятся в 5-ой строке таблицы MCB).

Шаг 3. Программа была повторна запущена, чтобы удостовериться, что программа определяет установленный обработчик прерываний.

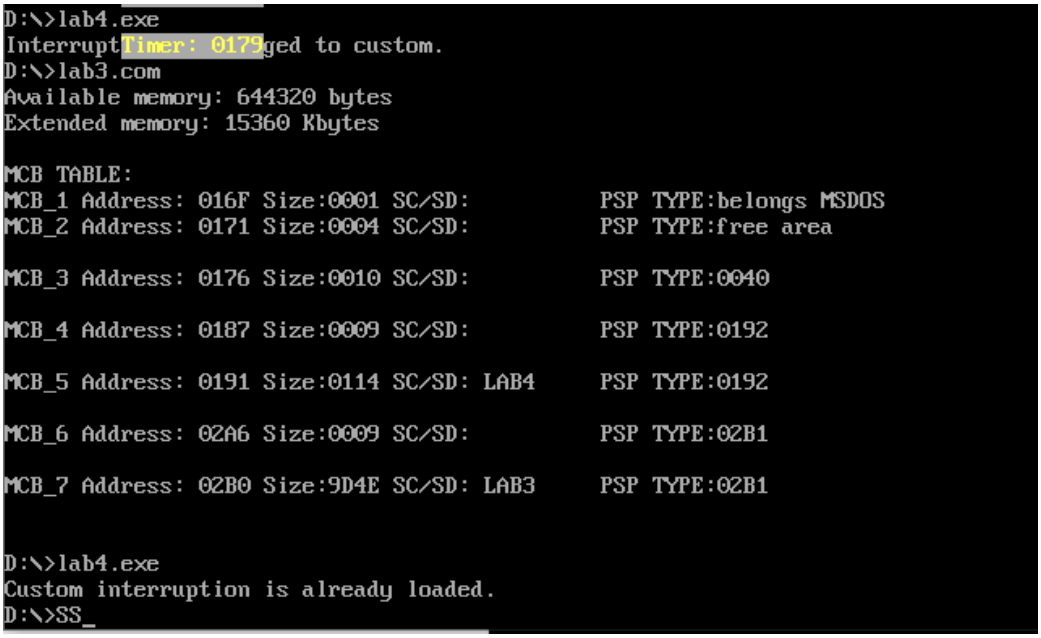

Рисунок 2 – Демонстрация корректного определения установленного обработчика прерывания при повторном запуске программы.

Шаг 4. Программа была запущена с ключом выгрузки, чтобы убедиться, что резидентный обработчик прерывания выгружен и память, занятая резидентом освобождена.

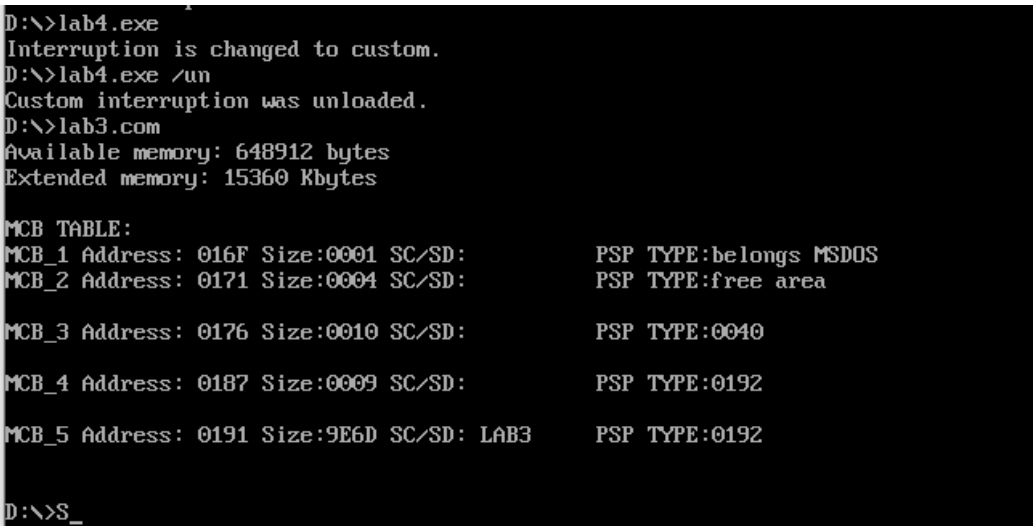

Рисунок 3 – Демонстрация корректной выгрузки резидентного обработчика прерываний.

По итогам выполнения работы можно ответить на контрольные вопросы:

1. Как реализован механизм прерывания от часов?

Ответ: Прерывание INT 1Ch вызывается обработчиком аппаратного прерывания от таймера INT 08h приблизительно 18,2 раза в секунду.

Сначала запоминаются значения регистров, определяется смещение по номеру источника прерывания в таблице векторов (2 байта в IP, два – в CS). Вызывается обработчик прерывания по сохраненному адресу.

В конце управление передается обратно от обработчика прерывания к прерванной программе.

2. Какого типа прерывания использовались в работе?

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

Выводы.

Построен собственный обработчик прерываний сигналов таймера. Получены дополнительные знания о работе с памятью (резидентный обработчик может быть загружен и выгружен из памяти).

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

lab4.asm:

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

DATA  SEGMENT
    ROUT_LOADED db "Custom interruption is already loaded.$"
    ROUT_IS_LOADING db "Interruption is changed to custom.$"
    ROUT_IS_NOT_LOADED db "Default interruption is set and can't be
unloaded.$"
    ROUT_IS_UNLOADED db "Custom interruption was unloaded.$"
DATA  ENDS

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

PRINT_BUF proc near
    push ax
    mov ah, 9h
    int 21h
    pop ax
    ret
PRINT_BUF endp


start_rout:
ROUT proc far
    jmp start_proc
    KEEP_PSP dw 0
    KEEP_IP dw 0
     KEEP_CS dw 0
    KEEP_SS DW 0
     KEEP_SP DW 0
     KEEP_AX DW 0

    ROUT_INDEX dw 1337h
    TIMER_COUNTER db 'Timer: 0000$'
    BStack DW 64 DUP(?)
start_proc:
    mov KEEP_SP, sp
    mov KEEP_AX, ax
    mov ax, ss
    mov KEEP_SS, ss

    mov ax, KEEP_AX

    mov sp, offset start_proc

    mov ax, seg BStack
    mov ss, ax

    push bx
     push cx
```

```
      push dx

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


    push si
    push cx
    push ds
    push ax
    push bp


    mov ax, SEG TIMER_COUNTER
    mov ds,ax
    mov si, offset TIMER_COUNTER

    add si, 6
    mov cx, 4

timer_inc:
    mov bp, cx
    mov ah, [si+bp]
    inc ah
    cmp ah, 3ah
    jl timer_inc_end
    mov ah, 30h
    mov [si+bp], ah

    loop timer_inc

timer_inc_end:
    mov [si+bp], ah

    pop bp
    pop ax
    pop ds
    pop cx
    pop si

    push es
     push bp

    mov ax, SEG TIMER_COUNTER
     mov es,ax
     mov ax, offset TIMER_COUNTER
     mov bp,ax
     mov ah,13h
     mov al,00h
    mov dh,02h
     mov dl,09h
     mov cx,11
     mov bh,0
     int 10h
```

```
      pop bp
      pop es

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

      pop dx
      pop cx
      pop bx

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

    mov al, 20H
    out 20H, al

    iret
end_rout:
ROUT endp


IF_NEED_UNLOAD proc near
      push ax
    push es

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

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

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

    mov cl,1h

end_if_need_unload:
    pop es
      pop ax
      ret
IF_NEED_UNLOAD endp


LOAD_ROUT PROC near
      push ax
      push dx

    mov KEEP_PSP, es
```

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

      push ds
      lea dx, ROUT
      mov ax, SEG ROUT
      mov ds,ax
      mov ah,25h
      mov al,1ch
      int 21h
      pop ds

      lea dx, end_rout
      mov cl,4h
      shr dx,cl
      inc dx
      add dx,100h
    xor ax, ax
      mov ah,31h
      int 21h

      pop dx
      pop ax
      ret
LOAD_ROUT endp

UNLOAD_ROUT PROC near
      push ax
      push si

    cli
      push ds
      mov ah,35h
      mov al,1ch
    int 21h

    mov si,offset KEEP_IP
    sub si,offset ROUT
    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

    pop si
    pop ax
    ret
UNLOAD_ROUT endp

IF_LOADED proc near
     push ax
     push si

    push es
    push dx

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

     mov si, offset ROUT_INDEX
     sub si, offset ROUT
     mov dx,es:[bx+si]
     cmp dx, ROUT_INDEX
     jne end_if_loaded
     mov ch,1h

end_if_loaded:
    pop dx
    pop es
     pop si
     pop ax
     ret
IF_LOADED ENDP

MAIN proc far
    push  DS
    push  AX
    mov   AX,DATA
    mov   DS,AX

    call IF_NEED_UNLOAD
    cmp cl, 1h
    je need_unload

    call IF_LOADED
    cmp ch, 1h
    je print_rout_is_already_set
    mov dx, offset ROUT_IS_LOADING
    call PRINT_BUF
    call LOAD_ROUT
    jmp exit

need_unload:
```

```
    call IF_LOADED
    cmp ch, 1h
    jne print_rout_cant_be_unloaded
    call UNLOAD_ROUT
    mov dx, offset ROUT_IS_UNLOADED
    call PRINT_BUF
    jmp exit

print_rout_cant_be_unloaded:
    mov dx, offset ROUT_IS_NOT_LOADED
    call PRINT_BUF
    jmp exit
print_rout_is_already_set:
    mov dx, offset ROUT_LOADED
    call PRINT_BUF
    jmp exit

exit:
    mov ah, 4ch
    int 21h
MAIN endp
CODE ends
END Main
```