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.

Студентка гр. 0382 _____ Михайлова О.Д.

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 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.

Выполнение работы.

Для выполнения задания были использованы шаблоны из методических указаний, а также были добавлены следующие процедуры:

- PRINT_STRING – процедура вывода строки на экран;
- MY_INTERRUPT – собственный обработчик прерывания, выводит количество прерываний, которые были вызваны;
- CHECK_COMMAND – проверка наличия ключа /un при запуске программы;
- IS_INTERRUPT_LOAD – проверка, загружено ли пользовательское прерывание;
- LOAD_INTERRUPT – загрузка обработчика прерывания в память;
- INTERRUPT_UNLOAD – выгрузка пользовательского прерывания из памяти.

Шаг 1. Был написан и отлажен программный модуль .EXE, который выполняет все заданные в условии функции. В верхнем левом углу находиться счетчик, который показывает, сколько раз было вызвано прерывание.

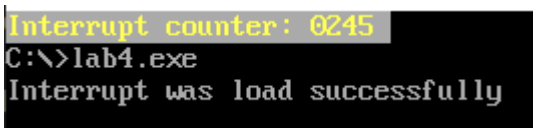

Рисунок 1 - результат запуска модуля lab4.exe

Шаг 2. Был запущен модуль .COM из лабораторной работы №3 для проверки того, что прерывание находиться в памяти.

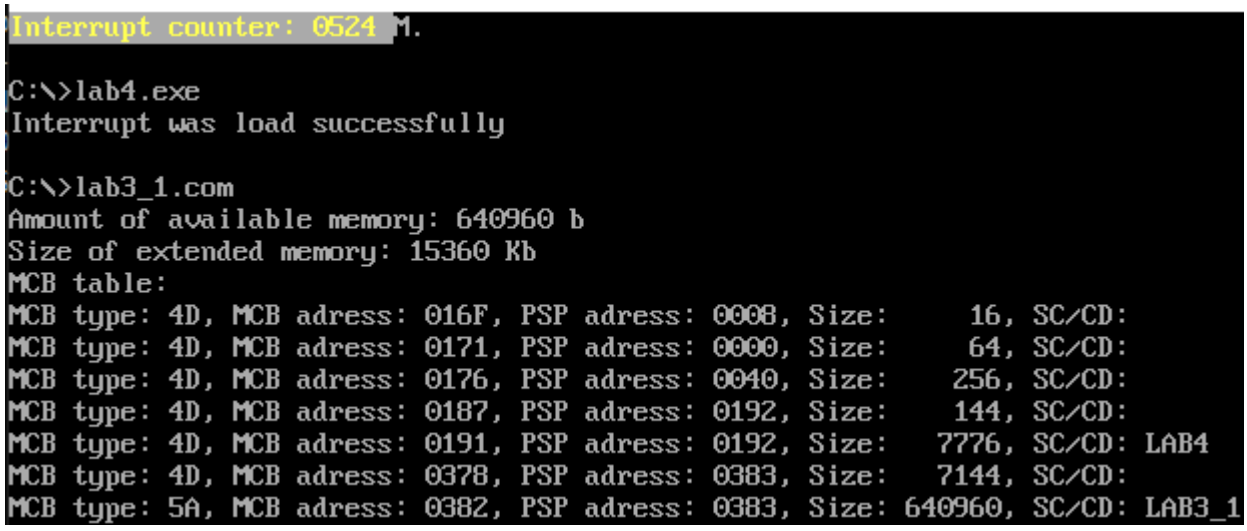

Рисунок 2 - Результат запуска модуля lab3_1.com

Шаг 3. Отлаженная программа была запущена еще раз, в результате чего на экран было выведено сообщение о том, что обработчик прерывания уже загружен в память.

Рисунок 3 - Результат повторного запуска модуля lab4.exe

Шаг 4. Была запущена программа с ключом выгрузки, в результате чего на экран было выведено сообщение о том, что обработчик прерывания был

выгружен из памяти. Для того, чтобы в этом убедиться, повторно был запущен модуль lab3_1.com.

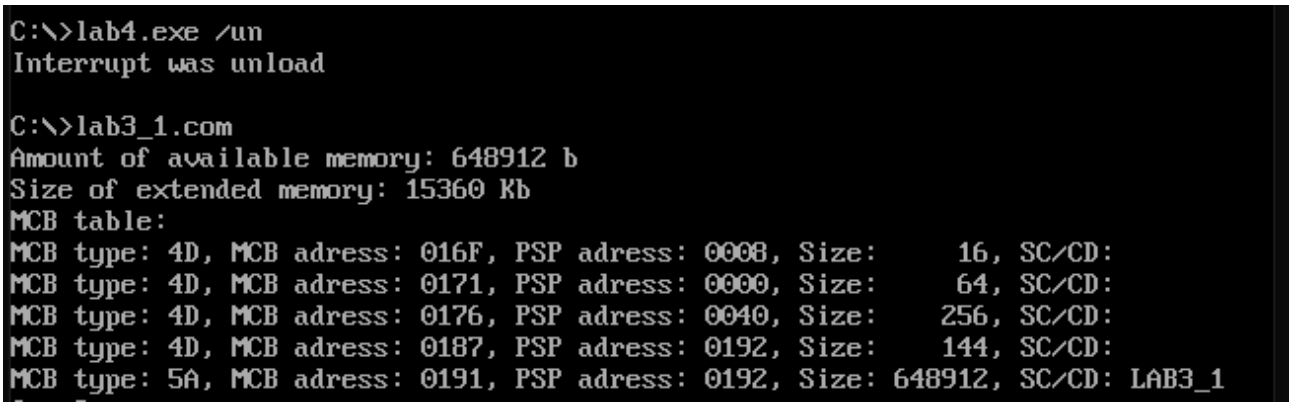

Рисунок 4 - Запуск модуля lab4.exe с ключом /un и результат повторного запуска модуля lab3_1.com

Исходный код программы см. в приложении А.

Ответы на контрольные вопросы.

1. Как реализован механизм прерывания от часов?

Прерывание от часов int 1Ch вызывается обработчиком аппаратного прерывания от системного таймера int 08h.

Механизм: принимается сигнал прерывания от часов и запоминаются содержимое регистров. Далее по номеру источника прерывания определяется смещение вызываемого вектора, запоминается адрес в IP и CS. После этого выполняется прерывание по сохраненному адресу. В конце управление возвращается прерванной программе.

2. Какого типа прерывания использовались в работе?

Программное прерывание (21h и 10h) и аппаратное прерывание (1Ch)

Выводы.

В ходе работы был построен собственный обработчик прерывания от сигналов таймера. Были изучены дополнительные функции работы с памятью: загрузка в память обработчика прерывания и его выгрузка из нее.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab4.asm

```
AStack SEGMENT STACK
     DW 256 DUP(?)
AStack ENDS

DATA SEGMENT
     INT_LOAD db "Interrupt was load successfully", 0Dh, 0Ah, '$'
     INT_NOT_LOAD db "Interrupt is not load", 0Dh, 0Ah, '$'
     INT_UNLOAD db "Interrupt was unload", 0Dh, 0Ah, '$'
     INT_ALREADY_LOAD db "Interrupt has already been loaded", 0Dh,
0Ah, '$'

     flag_cmd db 0
     flag_load db 0

DATA ENDS

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


PRINT_STRING PROC near
     push ax
     mov ah, 09h
     int 21h
     pop ax
     ret
PRINT_STRING ENDP

MY_INTERRUPT PROC far
     jmp start_func

     KEEP_PSP dw ?
     KEEP_IP dw 0
     KEEP_CS dw 0
     INT_ID dw 5555h
     INT_COUNT db 'Interrupt counter: 0000 $'

     KEEP_SS dw ?
     KEEP_SP dw ?
     KEEP_AX dw ?
     INTERRUPT_STACK dw 128 dup (?)
     END_INT_STACK dw ?

start_func:
     mov KEEP_SS, ss
     mov KEEP_SP, sp
     mov KEEP_AX, ax
     mov ax, cs
     mov ss, ax
     mov sp, offset END_INT_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, 0h
        mov dl, 0h
        int 10h


        push si
        push cx
        push ds
        push bp

        mov ax, SEG INT_COUNT
        mov ds, ax
        mov si, offset INT_COUNT
        add si, 18
        mov cx, 4

l_loop:
        mov bp, cx
        mov ah, [si+bp]
        inc ah
        mov [si+bp], ah
        cmp ah, 3Ah
        jl metka
        mov ah, 30h
        mov [si+bp], ah
        loop l_loop

metka:
        pop bp
        pop ds
        pop cx
        pop si

        push es
        push bp

        mov ax, SEG INT_COUNT
        mov es, ax
        mov ax, offset INT_COUNT
        mov bp, ax
        mov ah, 13h
        mov al, 00h
```

```
      mov cx, 24
      mov bh, 0h
      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
int_end:
MY_INTERRUPT ENDP

CHECK_COMMAND PROC NEAR
      push es
      mov ax, KEEP_PSP
      mov es, ax
      mov bx, 82h
      mov al, es:[bx]
      inc bx
      cmp al, '/'
      jne check_end

      mov al, es:[bx]
      inc bx
      cmp al, 'u'
      jne check_end

      mov al, es:[bx]
      inc bx
      cmp al, 'n'
      jne check_end
      mov flag_cmd, 1h

check_end:
      pop es
      ret
CHECK_COMMAND ENDP



IS_INTERRUPT_LOAD PROC NEAR
      push ax
```

```
      push bx
      push si

      mov ah, 35h
      mov al, 1Ch
      int 21h
      mov si, offset INT_ID
      sub si, offset MY_INTERRUPT
      mov dx, es:[bx + si]
      cmp dx, 5555h
      jne is_load_end
      mov flag_load, 1h

is_load_end:
      pop si
      pop bx
      pop ax

      ret
IS_INTERRUPT_LOAD ENDP



LOAD_INTERRUPT PROC NEAR
      push ax
      push cx
      push dx
      push es
      push ds

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

      mov KEEP_IP, bx
      mov KEEP_CS, es

      mov dx, offset MY_INTERRUPT
      mov ax, seg MY_INTERRUPT
      mov ds, ax

      mov ah, 25h
      mov al, 1Ch
      int 21h
      pop ds

      mov dx, offset INT_LOAD
      call PRINT_STRING

      mov dx, offset int_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

      pop es
      pop dx
      pop cx
      pop ax

      ret
LOAD_INTERRUPT ENDP

INTERRUPT_UNLOAD PROC NEAR
      push ax
      push bx
      push dx
      push si
      push es

      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

      mov dx, offset INT_UNLOAD
      call PRINT_STRING

      pop es
```

```
      pop si
      pop dx
      pop bx
      pop ax

      ret
INTERRUPT_UNLOAD ENDP



Main PROC FAR
      mov ax, DATA
      mov ds, ax
      mov KEEP_PSP, es

      call CHECK_COMMAND
         cmp flag_cmd, 1
         je unload_int

         call IS_INTERRUPT_LOAD
         cmp flag_load, 0
         je not_load
         mov DX, OFFSET INT_ALREADY_LOAD
         call PRINT_STRING
         jmp final

not_load:
         call LOAD_INTERRUPT
         jmp final

unload_int:
         call IS_INTERRUPT_LOAD
         cmp flag_load, 0
         jne already_load
         mov DX, OFFSET INT_NOT_LOAD
         call PRINT_STRING
         jmp final
already_load:
         call INTERRUPT_UNLOAD

final:
      mov ah, 4Ch
      int 21h

Main ENDP
CODE ENDS
END Main
```