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

Студентка гр. 9383 _____ Сергиенкова А.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 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.

Были написаны строки для вывода информации:

- STR_ISN_LOAD DB 'Iterrapt is not load', 0AH, 0DH,'$'
- STR_ALR_LOAD DB 'Iterrapt is already loaded', 0AH, 0DH,'$'
- STR_HB_LOAD DB 'Iterrapt has been loaded', 0AH, 0DH,'$'
- STR_IS_UNLOAD DB 'Iterrapt is unloaded', 0AH, 0DH,'$'

Переменные для хранения флагов:

- flag db 0 – флаг удаления;
- flag_load db 0 – флаг загрузки.

Переменные, хранящиеся в прерывании:

- PSP dw ? – сохранение адреса PSP;
- KEEP_IP dw 0 – сохранение данных исходного прерывания;
- KEEP_CS dw 0 – сохранение данных исходного прерывания;
- ITERRUPT_ID dw 8f17h – уникальный идентификатор прерывания;
- STR_COUNTER db 'Number of my iterrups: 0000$' – строка вывода кол-во прерываний;
- KEEP_SS dw ? – для работы стека прерывания;
- KEEP_SP dw ? – для работы стека прерывания;
- KEEP_AX dw ? – для работы стека прерывания;
- ITERRUPT_STACK dw 32 dup (?) –стек прерывания;
- END_IT_STACK dw ? – конец стека прерывания;

Были написаны функции:

- ITERRUPT – Прерывание, которое загружается в память и выполняет накопление и вывод числа накопленных прерываний на экран.
- WRITE_STRING – Вывод строки на экран.
- LOAD_FLAG – Проверка на наличия флага “/un”.
- IS_LOAD – Проверка на загрузку пользовательского прерывания в память.
- LOAD_ITERRAPT – Сохранение первоначального прерывания и загрузка пользовательского прерывания в память.
- UNLOAD_ITERRAPT – Выгрузка пользовательского прерывания из памяти, а также освобождение памяти и восстановление первоначальных прерываний.
- MAIN – Главная функция.

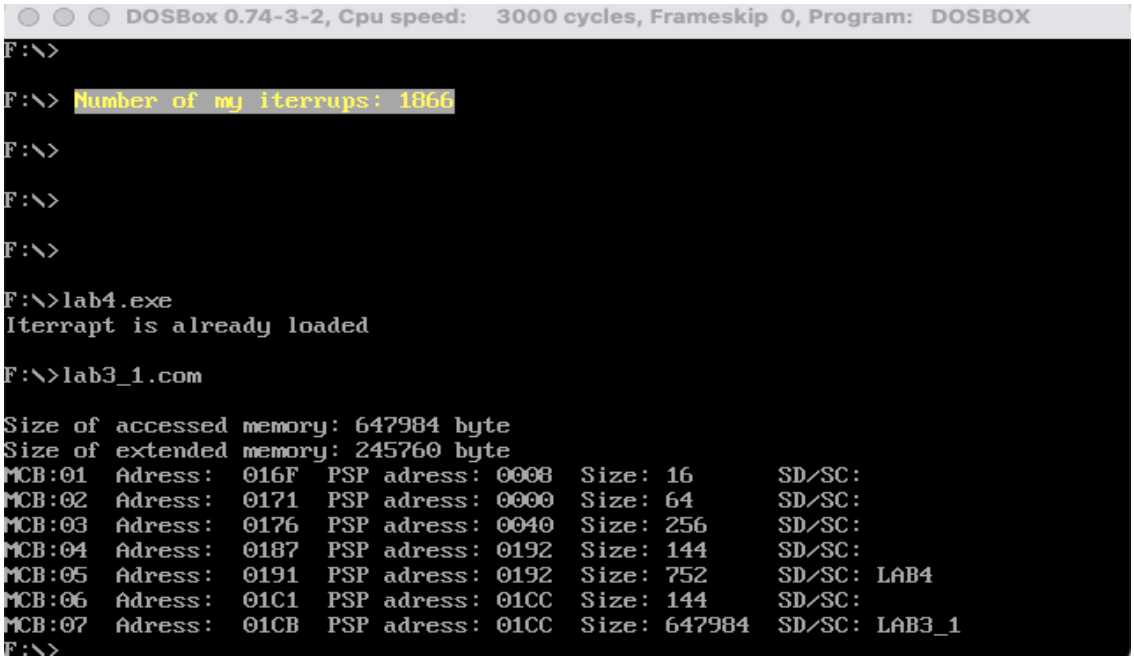
РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ ПРОБЛЕМ

Шаг 1.

Был написан и отлажен программный модуль типа .EXE.

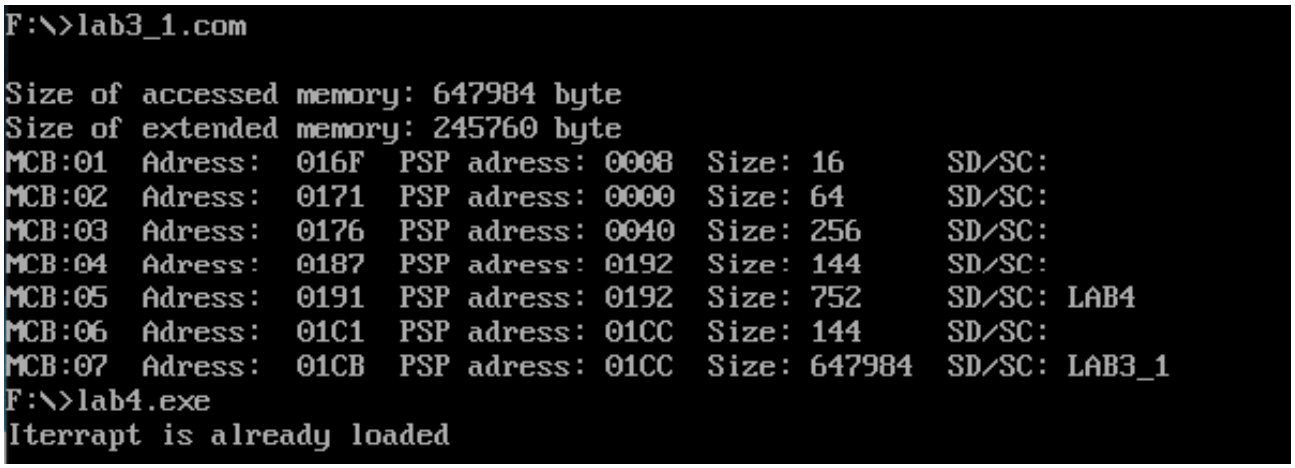
Шаг 2.

Написанный модуль был отлажен и запущен. Резидентный обработчик был установлен и размещён в памяти.



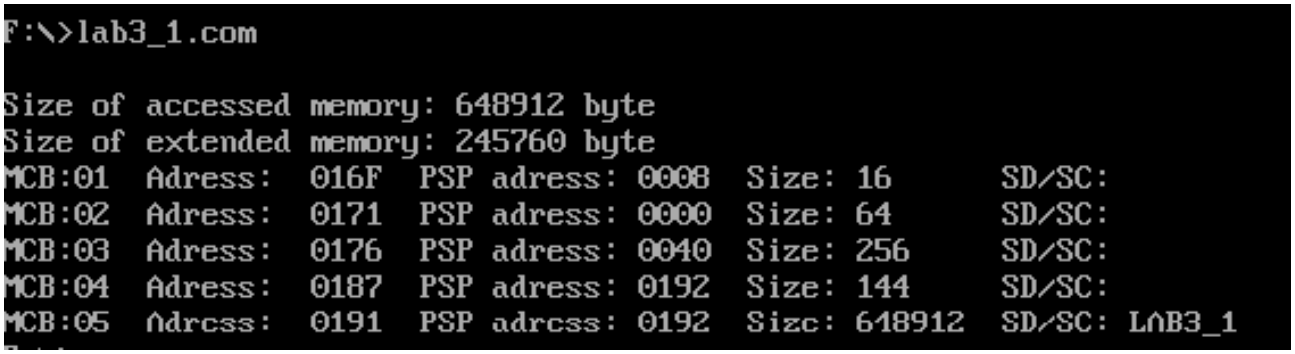
Шаг 3.

Программа корректно определяет установленный обработчик прерываний.



Шаг 4.

Была запущена отлаженная программа с ключом выгрузки.



Шаг 5.

Была произведена оценка результатов и были отвечены контрольные вопросы.

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как реализован механизм прерывания от часов?

Принимается сигнал прерывания (приходит примерно каждые 54 мс), запоминаются содержимые регистров, по номеру источника прерывания в таблице векторов определяется смещение, запоминается адрес 2 байта в IP и 2 байта в CS. Дальше выполняется прерывание по сохранённому адресу и далее восстанавливается информация прерванного процесса и управление возвращается прерванной программе.

2. Какие прерывания использовались в работе?

- Int 10h – видео сервис BIOS
- Int 21h – сервисы DOS
- Пользовательское прерывание с вектором 1ch int 21h

Выводы.

В ходе лабораторной работы была исследована обработка стандартных прерываний, а также построен обработчик прерываний сигналов таймера, которые генерируются аппаратурой через определённые интервалы времени.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Lab4.asm

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

DATA      SEGMENT
  flag db 0
  flag_load db 0

  STR_ISN_LOAD  DB 'Iterrapt is not load', 0AH, 0DH,'$'
  STR_ALR_LOAD  DB 'Iterrapt is already loaded', 0AH, 0DH,'$'
  STR_HB_LOAD  DB 'Iterrapt has been loaded', 0AH, 0DH,'$'
  STR_IS_UNLOAD  DB 'Iterrapt is unloaded', 0AH, 0DH,'$'
DATA      ENDS

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

WRITE_STRING PROC near
   push AX
   mov AH,09h
   int 21h
   pop AX
   ret
WRITE_STRING ENDP

ITERRUPT PROC far
   jmp start_iterrupt

   PSP dw ?
   KEEP_IP dw 0
   KEEP_CS dw 0
   ITERRUPT_ID dw 8f17h
```

```
 STR_COUNTER db 'Number of my iterrups: 0000$'

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

start_iterrupt:
  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

  ;get cursor
        mov ah,3h
        mov bh,0h
        int 10h
        push dx

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

  ;number of times
  push si
        push cx
        push ds
  push bp
```

```
        mov ax,SEG STR_COUNTER
        mov ds,ax
        mov si,offset STR_COUNTER
        add si,22 ;26

  mov cx,4
iterrapt_loop:
  mov bp,cx
  mov ah,[si+bp]
        inc ah
        mov [si+bp],ah
        cmp ah,3Ah
        jne m_number
        mov ah,30h
        mov [si+bp],ah

  loop iterrapt_loop

m_number:
  pop bp
  pop ds
  pop cx
  pop si

   ;write string
        push es
        push bp

        mov ax,SEG STR_COUNTER
        mov es,ax
        mov ax,offset STR_COUNTER
        mov bp,ax
        mov ah,13h
        mov al,00h
        mov cx,27 ; number of chars
        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 ax, KEEP_SS
        mov ss, ax
        mov ax, KEEP_AX
        mov sp, KEEP_SP

   iret
iterrapt_end:
ITERRUPT ENDP

LOAD_FLAG PROC near
  push ax

  mov PSP,es
  mov al,es:[81h+1]
  cmp al,'/'
  jne load_flag_end
  mov al,es:[81h+2]
  cmp al, 'u'
  jne load_flag_end
  mov al,es:[81h+3]
  cmp al, 'n'
  jne load_flag_end
  mov flag,1h

load_flag_end:
  pop ax
```

```
  ret
LOAD_FLAG ENDP

IS_LOAD PROC near
  push ax
  push si

  mov ah,35h
  mov al,1Ch
  int 21h
  mov si,offset ITERRUPT_ID
  sub si,offset ITERRUPT
  mov dx,es:[bx+si]
  cmp dx, 8f17h
  jne is_load_end
  mov flag_load,1h
is_load_end:
  pop si
  pop ax
  ret
IS_LOAD ENDP

LOAD_ITERRAPT 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, ITERRUPT
  mov ax, seg ITERRUPT
  mov ds,ax
  mov ah,25h
  mov al, 1Ch
  int 21h
  pop ds
  lea dx, STR_ALR_LOAD
  call WRITE_STRING

   lea dx, iterrapt_end
   mov CL, 4h
   shr DX,CL
   inc DX
   mov ax,cs
   sub ax,PSP
   add dx,ax
   xor ax,ax
   mov AH,31h
   int 21h

end_load:
   pop dx
   pop ax
   ret
LOAD_ITERRAPT ENDP

UNLOAD_ITERRAPT PROC near
   push ax
   push si

   call IS_LOAD
   cmp flag_load,1h
   jne cant_unload
```

```
  jmp start_unload

cant_unload:
  lea dx,STR_ISN_LOAD
  call WRITE_STRING
  jmp m_unload_end


start_unload:
  CLI ;восстановим оригинальный вектор
  PUSH DS
  mov ah,35h
  mov al,1Ch
        int 21h

  mov si,offset KEEP_IP
        sub si,offset ITERRUPT
        mov dx,es:[bx+si]
        mov ax,es:[bx+si+2]
  MOV DS,AX
  MOV AH,25H
  MOV AL, 1CH
  INT 21H
  POP DS

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

  mov ax,es:[2ch] ; очистка данных из префикса
  mov es,ax
  mov ah,49h
  int 21h

  pop es
  mov ah,49h
  int 21h
  STI
```

```
    lea dx,STR_IS_UNLOAD
    call WRITE_STRING
  m_unload_end:
    pop si
    pop ax
    ret
  UNLOAD_ITERRAPT ENDP

  ; Головная процедура
  Main      PROC  FAR
    push  DS
    xor   AX,AX
    push  AX
    mov   AX,DATA
    mov   DS,AX

    call LOAD_FLAG
    cmp flag, 1h
    je m_unload_iterrapt
    call LOAD_ITERRAPT
    jmp m_end

  m_unload_iterrapt:
    call UNLOAD_ITERRAPT

  m_end:
    mov ah,4ch
    int 21h
  Main      ENDP
  CODE      ENDS
END Main
```