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

Студент гр. 9382 _____ Павлов Р.В.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2021

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 lCh.

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 солержать код устанавливаемого прерывания в виде удаленной процедуры. Этот код будет работать после установки при возникновении прерывания. Он должен выполнять следующие функции:

1) Сохранить значения регистров в стеке при входе и восстановить их при выходе.

2) 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.

Шаг 2. Запустите отлаженную программу и убедитесь, что резидентный обработчик прерывания lCh 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.

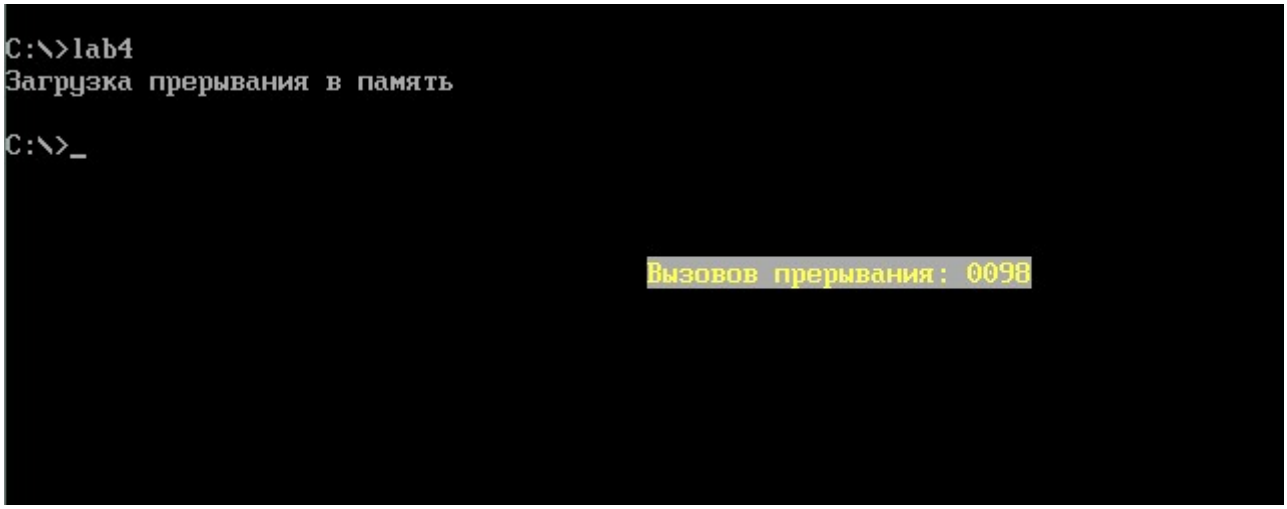
Шаг 3. Запустите отлаженную программу еще раз и убедитесь, что программа определяет установленный обработчик прерываний. Полученные результаты поместите в отчет.

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 того также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

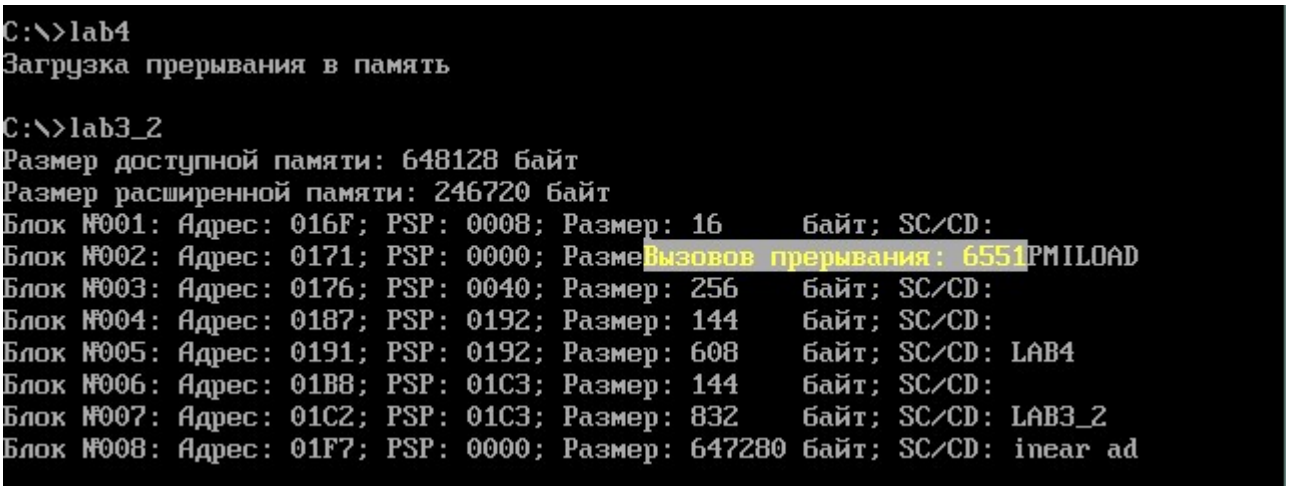
Шаг 5. Ответьте на контрольные вопросы.

Выполнение работы.

1. Был реализован пользовательский обработчик прерываний от таймера, который хранится в памяти после завершения программы резидентно. На экран выводится количество вызовов прерывания. При запуске обработчик помещается в память и хранится там до указания выгрузки.



2. Осуществлена проверка памяти:



Видно, что обработчик помещён в память.

3. Повторный запуск программы привёл к выводу на экран сообщения о том, что обработчик уже находится в памяти.

```
C:\>lab4
Загрузка прерывания в память

C:\>lab3_2
Размер доступной памяти: 648128 байт
Размер расширенной памяти: 246720 байт
Блок №001: Адрес: 016F; PSP: 0008; Размер: 16     байт; SC/CD:
Блок №002: Адрес: 0171; PSP: 0000; Разме                PMILOAD
Блок №003: Адрес: 0176; PSP: 0040; Размер: 256    байт; SC/CD:
Блок №004: Адрес: 0187; PSP: 0192; Размер: 144    байт; SC/CD:
Блок №005: Адрес: 0191; PSP: 0192; Размер: 608    байт; SC/CD: LAB4
Блок №006: Адрес: 01B8; PSP: 01C3; Размер: 144    байт; SC/CD:
Блок №007: Адрес: 01C2; PSP: 01C3; Размер: 832    байт; SC/CD: LAB3_2
Блок №008: Адрес: 01F7; PSP: 0000; Размер: 647280 байт; SC/CD: inear ad

C:\>lab4
Прерывание уже загружено в память
```

4. Программа запущена с параметром /un, обработчик удалён из памяти, что видно на скриншоте ниже.

```
Блок №004: Адрес: 0187; PSP: 0192; Размер: 144    байт; SC/CD:
Блок №005: Адрес: 0191; PSP: 0192; Размер: 608    байт; SC/CD: LAB4
Блок №006: Адрес: 01B8; PSP: 01C3; Размер: 144    байт; SC/CD:
Блок №007: Адрес: 01C2; PSP: 01C3; Размер: 832    байт; SC/CD: LAB3_2
Блок №008: Адрес: 01F7; PSP: 0000; Размер: 647280 байт; SC/CD: inear ad

C:\>lab4
Прерывание уже загружено в память

C:\>lab4 /un
Прерывание выгружено из памяти

C:\>lab3_2
Размер доступной памяти: 648912 байт
Размер расширенной памяти: 246720 байт
Блок №001: Адрес: 016F; PSP: 0008; Размер: 16     байт; SC/CD:
Блок №002: Адрес: 0171; PSP: 0000; Размер: 64     байт; SC/CD: DPMILOAD
Блок №003: Адрес: 0176; PSP: 0040; Размер: 256    байт; SC/CD:
Блок №004: Адрес: 0187; PSP: 0192; Размер: 144    байт; SC/CD:
Блок №005: Адрес: 0191; PSP: 0192; Размер: 832    байт; SC/CD: LAB3_2
Блок №006: Адрес: 01C6; PSP: 0000; Размер: 648064 байт; SC/CD:

C:\>
```

Ответы на контрольные вопросы

Контрольные вопросы по лабораторной работе №4

1) При вызове прерывания текущие значения CS и IP, а также регистр флагов, запоминаются. Затем значения CS и IP меняются на указанные в векторе, осуществляется переход к функции-обработчику прерывания. После завершения работы этой функции исходные значения изменённых регистров восстанавливаются, и основная программа продолжает работу.

2) В работе использовались как аппаратные, так и пользовательские прерывания.

Выводы.

Была исследована обработка прерываний, реализована пользовательская функция-обработчик прерывания, заменившая стандартную.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Имя файла: lab4.asm

```
assume cs:code, ds:data, ss:Astack

Astack segment stack
      dw 32 dup(?)
Astack ends

code segment

inter_1Ch proc far
      jmp init_counter

      psp1 dw 0
      psp2 dw 0
      save_cs dw 0
      save_ip dw 0
      is_timer_set dw 0fedch
      counter db 'Вызовов прерывания: 0000  $'

init_counter:
      push ax
      push bx
      push cx
      push dx

      mov ah, 3
      xor bh, bh
      int 10h
      push dx
      mov ah, 2
      xor bh, bh
      mov dx, 828h
      int 10h

      push si
      push cx
      push ds
      mov ax, seg counter
      mov ds, ax
      mov si, offset counter
      add si, 23

      mov ah,[si]
      inc ah
      mov [si], ah
      cmp ah, 3ah
      jne print_timer
      mov ah, 30h
      mov [si], ah

      mov bh, [si - 1]
      inc bh
      mov [si - 1], bh
      cmp bh, 3ah
      jne print_timer
      mov bh, 30h
      mov [si - 1], bh
```

```
      mov ch, [si - 2]
      inc ch
      mov [si - 2], ch
      cmp ch, 3ah
      jne print_timer
      mov ch, 30h
      mov [si - 2], ch

      mov dh, [si - 3]
      inc dh
      mov [si - 3], dh
      cmp dh, 3ah
      jne print_timer
      mov dh, 30h
      mov [si - 3],dh

print_timer:
    pop ds
    pop cx
      pop si

      push es
    push bp
    mov ax, seg counter
    mov es, ax
    mov bp, offset counter
    mov ax, 1300h
    mov cx, 24
    mov bh, 0
    int 10h
    pop bp
      pop es
      pop dx
      mov ah, 2
      xor bh, bh
      int 10h

      pop dx
      pop cx
      pop bx
      pop ax
      iret
inter_1Ch endp

print_string proc near
      push ax
      mov ah, 9h
      int   21h
      pop ax
      retn
print_string endp

reserve proc
reserve endp

is_set proc near
      push bx
      push dx
      push es

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

```
      mov dx, es:[bx + 11]
      cmp dx, 0fedch
      je is
      xor al, al
      jmp isnt

      is:
            mov al, 1
      isnt:
            pop es
            pop dx
            pop bx
      retn
is_set endp

param proc near
      push es

      mov ax, psp1
      mov es, ax
      mov bx, 82h

      mov al, es:[bx]
      inc bx
      cmp al, '/'
      jne unknown_param

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

      mov al, es:[bx]
      inc bx
      cmp al, 'n'
      jne unknown_param

      mov al, 1
      unknown_param:
            pop es
            retn
param endp

load proc near
      push ax
      push bx
      push dx
      push es

      mov ah, 35h
      mov al, 1ch
      int 21h
      mov save_ip, bx
      mov save_cs, es

      push ds
    mov ax, seg inter_1Ch
    mov ds, ax
      mov dx, offset inter_1Ch
    mov ah, 25h
    mov al, 1ch
    int 21h
      pop ds
```

```
      mov dx, offset ld_in_process
      call print_string

      pop es
      pop dx
      pop bx
      pop ax
      retn
load endp

unload proc near
      push ax
      push bx
      push dx
      push es

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

      cli
      push ds
    mov dx, es:[bx + 9]
    mov ax, es:[bx + 7]
    mov ds, ax
    mov ah, 25h
    mov al, 1ch
    int 21h
      pop ds
      sti

      mov dx, offset unloaded
      call print_string

      push es
    mov cx, es:[bx + 3]
    mov es, cx
    mov ah, 49h
    int 21h
      pop es

      mov cx, es:[bx + 5]
      mov es, cx
      int 21h

      pop es
      pop dx
      pop bx
      pop ax

      retn
unload endp

main proc far
      mov bx, 2ch
      mov ax, [bx]
      mov psp2, ax
      mov psp1, ds
      sub ax, ax
      xor bx, bx

      mov ax, data
      mov ds, ax
```

```
      call param
      cmp al, 1
      je tounld

      call is_set
      cmp al, 1
      jne nload

      mov dx, offset loaded
      call print_string
      jmp exit

      mov ah,4ch
      int 21h

      nload:
            call load

            mov dx, offset reserve
            mov cl, 4
            shr dx, cl
            add dx, 1bh

            mov ax, 3100h
            int 21h

      tounld:
            call is_set
            cmp al, 0
            je is_missing
            call unload
            jmp exit

      is_missing:
            mov dx, offset missing
            call print_string

      exit:
            mov ah, 4ch
            int 21h
main endp

code ends

data segment
      missing db "Не найдено прерывание в памяти", 13, 10, '$'
      unloaded db "Прерывание выгружено из памяти", 13, 10, '$'
      loaded db "Прерывание уже загружено в память", 13, 10, '$'
      ld_in_process db "Загрузка прерывания в память", 13, 10, '$'
data ends

end main
```