

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. 0382

Охотникова Г.С.

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 4 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

Выполнение работы.

1. При выполнении данной лабораторной работы был написан модуль EXE, в который были добавлены следующие процедуры:

- INTERRUPT — печать на экран количества вызванных прерываний.
- IS_INTERRUPT_SET — проверка, установлен ли разработанный вектор прерывания.
- IS_COMMAND_PROMT — выгрузка прерывания или загрузка с /un.
- LOAD_INTERRUPT — установка нового обработчика прерываний.
- UNLOAD_INTERRUPT — установка сохраненных обработчиков прерываний и выгрузка резидентов функции.
- PRINT_STRING — вывод строки на экран.

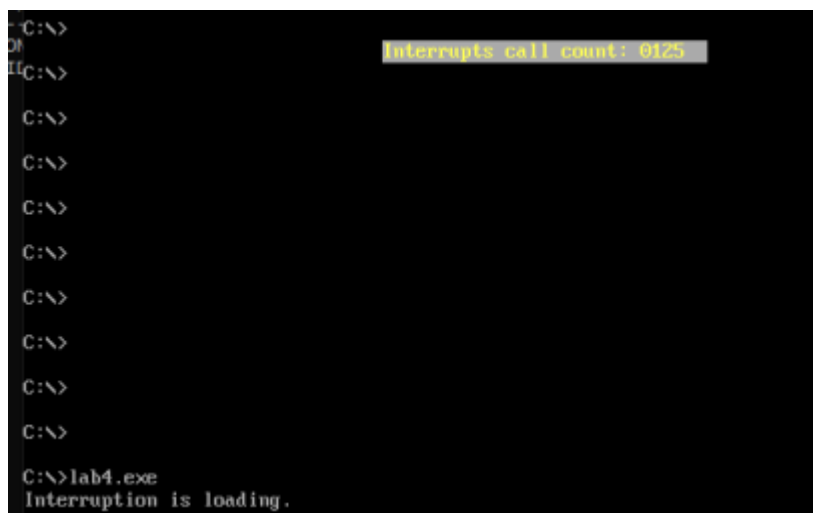


Рисунок 1 — Результат работы модуля lab4.exe

2. Был запущен модуль из лабораторной работы №3 lab3_1.com, который отображает карту памяти в виде списка блоков МСВ. Прерывание осталось в памяти, как видно из рисунка 2.

```

C:\>lab4.exe
Interruption is loading.

C:\>lab3_1.com
Amount of available memory: 647952 byte
Extended memory size: 245920 byte
MCB table:
Address: 016F PSP address: 0008 Size: 16 SC/SD:
Address: 0171 PSP address: 0000 Size: 64 SC/SD:
Address: 0176 PSP address: 0040 Size: 256 SC/SD:
Address: 0187 PSP address: 0192 Size: 144 SC/SD:
Address: 0191 PSP address: 0192 Size: 784 SC/SD: LAB4
Address: 01C3 PSP address: 01CE Size: 144 SC/SD:
Address: 01CD PSP address: 01CE Size: 647952 SC/SD: LAB3_1

```

Рисунок 2 — Результат работы модуля lab4.exe

3. Модуль lab4.exe был запущен еще раз. На экран вывелось сообщение о том, что прерывание уже загружено в память, то есть определяется установленный обработчик прерываний.

```

C:\>lab4.exe
Interruption is loaded.

```

Рисунок 3 — Результат вторичной работы модуля lab4.exe

4. Модуль lab4.exe был запущен с ключом выгрузки '/UN'. На экран вывелось сообщение о том, что стандартный обработчик прерываний был восстановлен.

```

C:\>lab4.exe /UN
Interruption was restored.

```

Рисунок 4 — Выгрузка пользовательского обработчика

Исходный программный код см. в приложении А.

Контрольные вопросы.

1. Как реализован механизм прерывания от часов?

Прерывание от часов (int 1ch) вызывается обработчиком аппаратного прерывания от таймера int 08h примерно 18 раз в секунду. Сохраняется содержимое регистров, определяется смещение и вызывается обработчик прерывания по сохраненному адресу. Управление передается прерванной программе.

2. Какого типа прерывания использовались в работе?

Программные (21h, 10h) и аппаратные (1ch).

Выводы.

При выполнении данной лабораторной работы была написана программа обработчика прерываний таймера, изучена обработка стандартных прерываний. Были изучены методы загрузки программы-резидента, а также его выгрузка из памяти.

ПРИЛОЖЕНИЕ А.

Название файла: lab4.asm

```
ASSUME CS:CODE, DS:DATA, SS:MY_STACK
```

```
MY_STACK SEGMENT STACK
```

```
    DW 64 DUP(?)
```

```
MY_STACK ENDS
```

```
CODE SEGMENT
```

```
INTERRUPT PROC far
```

```
    jmp START_FUNCTION
```

```
    PSP_ADDRESS_0 DW 0
```

```
    PSP_ADDRESS_1 DW 0
```

```
    KEEP_CS DW 0
```

```
    KEEP_IP DW 0
```

```
    KEEP_SP DW 0
```

```
    KEEP_SS DW 0
```

```
    KEEP_AX DW 0
```

```
    INTERRUPT_SET DW 0FEDCh
```

```
    INT_COUNT DB 'Interrupts call count: 0000  $'
```

```
    BStack DW 64 DUP(?)
```

```
START_FUNCTION:
```

```
    mov KEEP_SP, sp
```

```
    mov KEEP_AX, ax
```

```
    mov KEEP_SS, ss
```

```
    mov sp, offset START_FUNCTION
```

```
    mov ax, seg BStack
```

```
    mov ss, ax
```

```
    push ax
```

```
    push bx
```

```
    push cx
```

```
    push dx
```

```
    mov ah, 03h ;03h читает позицию и размер курсора
```

```
    mov bh, 00h ;bh - видео страница
```

```

int 10h ;выполнение
push dx
mov ah, 02h ;позиция курсора
mov bh, 00h ;bh - видео страница
mov dx, 0220h
int 10h ;выполнение

push si
push cx
push ds

mov ax, SEG INT_COUNT
mov ds, ax
mov si, offset INT_COUNT
add si, 1Ah
mov ah, [si]
inc ah
mov [si], ah
cmp ah, 3Ah
jne END_CLC
mov ah, 30h
mov [si], ah

mov bh, [si - 1]
inc bh
mov [si - 1], bh
cmp bh, 3Ah
jne END_CLC
mov bh, 30h
mov [si - 1], bh

mov ch, [si - 2]
inc ch
mov [si - 2], ch
cmp ch, 3Ah
jne END_CLC
mov ch, 30h
mov [si - 2], ch

```



```

    mov dh, [si - 3]
    inc dh
    mov [si - 3], dh
    cmp dh, 3Ah
    jne END_CLC
    mov dh, 30h
    mov [si - 3], dh

END_CLC:
    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, 1Dh
    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
    pop ax

```

```

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

        iret

INTERRUPT endp

MEMORY_AREA PROC
MEMORY_AREA endp
IS_INTERRUPT_SET PROC near
    push bx
    push dx
    push es

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

    mov dx, es:[bx + 17]
    cmp dx, 0FEDCh
    je IS_INTER_SET
    mov al, 00h
    jmp POP_REG

IS_INTER_SET:
    mov al, 01h
    jmp POP_REG

POP_REG:
    pop es
    pop dx
    pop bx

    ret
IS_INTERRUPT_SET endp

IS_COMMAND_PROMT PROC near
    push es

```

```

    mov ax, PSP_ADDRESS_0
    mov es, ax

    mov bx, 0082h

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

    mov al, es:[bx]
    inc bx
    cmp al, 'U'
    jne NULL_CMD

    mov al, es:[bx]
    inc bx
    cmp al, 'N'
    jne NULL_CMD

    mov al, 0001h
NULL_CMD:
    pop es
    ret
IS_COMMAND_PROMT endp

LOAD_INTERRUPT PROC near
    push ax
    push bx
    push dx
    push es

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

```

```

    push ds
    mov dx, offset INTERRUPT
    mov ax, seg INTERRUPT
    mov ds, ax
    mov ah, 25h
    mov al, 1Ch
    int 21h
    pop ds

    mov dx, offset INTERRUPT_LOADING
    call PRINT_STRING

    pop es
    pop dx
    pop bx
    pop ax

    ret
LOAD_INTERRUPT endp

UNLOAD_INTERRUPT 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 INTERRUPT_RESTORED
    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

    ret
UNLOAD_INTERRUPT endp

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

MAIN PROC FAR
    mov bx, 02Ch
    mov ax, [bx]
    mov PSP_address_1, ax
    mov PSP_address_0, ds
    sub ax, ax

```

```

    sub bx, bx

    mov ax, DATA
    mov ds, ax

    call IS_COMMAND_PROMT
    cmp al, 01h
    je START_UNLOAD

    call IS_INTERRUPT_SET
    cmp al, 01h
    jne INTERRUPT_NOT_LOADED

    mov dx, offset INTERRUPT_LOADED
    call PRINT_STRING
    jmp EXIT_PR

    mov ah, 4Ch
    int 21h

INTERRUPT_NOT_LOADED:
    call LOAD_INTERRUPT

    mov dx, offset MEMORY_AREA
    mov cl, 04h
    shr dx, cl
    add dx, 1Bh

    mov ax, 3100h
    int 21h

START_UNLOAD:
    call IS_INTERRUPT_SET
    cmp al, 00h
    je INTERRUPT_NOT_SET
    call UNLOAD_INTERRUPT
    jmp EXIT_PR

INTERRUPT_NOT_SET:

```

```

        mov dx, offset INT_NOT_SET
        call PRINT_STRING
        jmp EXIT_PR

EXIT_PR:
        mov ah, 4Ch
        int 21h

MAIN endp
CODE ENDS

DATA SEGMENT
        INT_NOT_SET DB 'Interruption did not load.', 0dh, 0ah, '$'
        INTERRUPT_RESTORED DB 'Interruption was restored.', 0dh, 0ah,
'$'
        INTERRUPT_LOADED DB 'Interruption is loaded.', 0dh, 0ah, '$'
        INTERRUPT_LOADING DB 'Interruption is loading.', 0dh, 0ah, '$'
DATA ENDS

END MAIN

```