

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. 9381

Давыдов Д.С.

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 содержать 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 `ICh` 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.

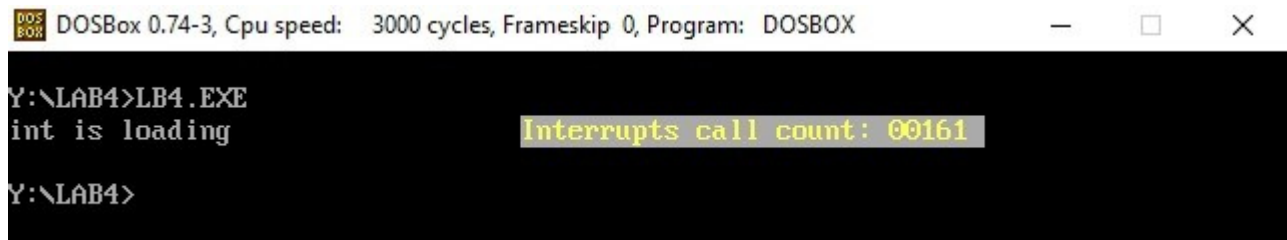
Шаг 4. Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для того также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

Шаг 5. Ответьте на контрольные вопросы.

Выполнение работы.

Был написан и отлажен EXE-модуль, который: проверяет установлено ли пользовательское прерывание с вектором 1Ch, устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено, и осуществляется выход по функции 4Ch прерывания int 21h. Если прерывание было установлено, то выводится соответствующее сообщение в консоль и происходит выход по функции 4Ch прерывания int 21h. Последняя функция — это выгрузка прерывания по соответствующему значению параметра при запуске программы /um. Операция выгрузки состоит в восстановлении стандартного вектора прерываний и освобождении памяти, занимаемой резидентом. Уже после осуществляется выход по функции 4Cp прерывания int 21h

- 1) Запущен модуль lab4.exe без параметров. В верхней части экрана расположился счетчик, который показывает количество сигналов таймера.



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
Y:\LAB4>LB4.EXE
int is loading
Interrupts call count: 00161
Y:\LAB4>
```

- 2) Был запущен модуль лабораторной 3, чтобы проверить наличие загруженного прерывания в память.

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
Y:\LAB4>LB4.EXE
int is loading Interrupts call count: 00478
Y:\LAB4>LB3_S1.COM
Amount of available memory: 648000 b
Size of extended memory: 15360 Kb
List of memory control blocks:
MCB type: 4Dh PSP address: 0008h Size: 16 b
MCB type: 4Dh PSP address: 0000h Size: 64 b
MCB type: 4Dh PSP address: 0040h Size: 256 b
MCB type: 4Dh PSP address: 0192h Size: 144 b
MCB type: 4Dh PSP address: 0192h Size: 736 b LB4
MCB type: 4Dh PSP address: 01CBh Size: 144 b
MCB type: 5Ah PSP address: 01CBh Size: 648000 b LB3_S1
Y:\LAB4>
```

3) Вновь запустили LB4.exe, тоже без аргументов. На экран вывелось изображение о том, что прерывание уже было загружено в память.

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
Y:\LAB4>LB4.EXE
int is loading Interrupts call count: 00703
Y:\LAB4>LB3_S1.COM
Amount of available memory: 648000 b
Size of extended memory: 15360 Kb
List of memory control blocks:
MCB type: 4Dh PSP address: 0008h Size: 16 b
MCB type: 4Dh PSP address: 0000h Size: 64 b
MCB type: 4Dh PSP address: 0040h Size: 256 b
MCB type: 4Dh PSP address: 0192h Size: 144 b
MCB type: 4Dh PSP address: 0192h Size: 736 b LB4
MCB type: 4Dh PSP address: 01CBh Size: 144 b
MCB type: 5Ah PSP address: 01CBh Size: 648000 b LB3_S1
Y:\LAB4>LB4.EXE
int is loaded
Y:\LAB4>
```

4) Теперь запускаем LB4.EXE с аргументом /um, чтобы восстановить стандартный обработчик прерывания.

```

Amount of available memory: 648000 b
Size of extended memory: 15360 Kb
List of memory control blocks:
MCB type: 4Dh PSP address: 0008h Size: 16 b
MCB type: 4Dh PSP address: 0000h Size: 64 b
MCB type: 4Dh PSP address: 0040h Size: 256 b
MCB type: 4Dh PSP address: 0192h Size: 144 b
MCB type: 4Dh PSP address: 0192h Size: 736 b LB4
MCB type: 4Dh PSP address: 01CBh Size: 144 b
MCB type: 5Ah PSP address: 01CBh Size: 648000 b LB3_S1

L:\LAB4>LB4.EXE
int is loaded

L:\LAB4>LB4.EXE /un
int was restored

```

- 5) Снова запустили модуль лабораторной 3, чтобы удостовериться, что память была освобождена.

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

```

MCB type: 4Dh PSP address: 0008h Size: 16 b
MCB type: 4Dh PSP address: 0000h Size: 64 b
MCB type: 4Dh PSP address: 0040h Size: 256 b
MCB type: 4Dh PSP address: 0192h Size: 144 b
MCB type: 4Dh PSP address: 0192h Size: 736 b LB4
MCB type: 4Dh PSP address: 01CBh Size: 144 b
MCB type: 5Ah PSP address: 01CBh Size: 648000 b LB3_S1

L:\LAB4>LB4.EXE
int is loaded

L:\LAB4>LB4.EXE /un
int was restored

L:\LAB4>LB3_S1.COM
Amount of available memory: 648912 b
Size of extended memory: 15360 Kb
List of memory control blocks:
MCB type: 4Dh PSP address: 0008h Size: 16 b
MCB type: 4Dh PSP address: 0000h Size: 64 b
MCB type: 4Dh PSP address: 0040h Size: 256 b
MCB type: 4Dh PSP address: 0192h Size: 144 b
MCB type: 5Ah PSP address: 0192h Size: 648912 b LB3_S1

L:\LAB4>

```

Разработанный программный код смотреть в приложении А.

Ответы на контрольные вопросы

Контрольные вопросы по лабораторной работе №4

1) Как реализован механизм прерывания от часов?

Ответ: Как только принимается сигнал прерывания (происходит это примерно каждые 54 мс.), запоминаются значения регистров. Затем определяется смещение по номеру источника прерывания в таблице векторов,

запоминается адрес 2 байта в IP и 2 байта в CS. После этого выполняется вызов обработчика прерывания по сохраненному адресу. По итогу восстанавливается информация прерванного процесса и происходит возвращение управления прерванной программе.

2) Какого типа прерывания использовались в работе?

Ответ: аппаратные (1Ch – прерывание от часов) и программные (int 10h - сервис BIOS, int 21h – сервисы DOS).

Выводы.

В ходе выполнения лабораторной работы была написана программа обработчика прерывания от сигналов таймера, была изучена установка резидентной программы в память и её выгрузка.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: LB4.ASM

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

```
LAB_STACK SEGMENT STACK
```

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

```
LAB_STACK ENDS
```

```
CODE SEGMENT
```

```
INTERRUPTION_TIMER PROC FAR
```

```
    jmp START
```

```
    pspAddres0 dw 0
```

```
    pspAddres1 dw 0
```

```
    keepCs dw 0
```

```
    keepIp dw 0
```

```
    keepSs dw 0
```

```
    keepSp dw 0
```

```
    keepAx dw 0
```

```
    interruptionTimerSet dw 0FEDCh
```

```
    intCount db 'Interrupts call count: 0000 $'
```

```
    newstack dw 64 dup(?)
```

```
START:
```

```
    mov keepSp, sp
```

```
    mov keepAx, ax
```

```
    mov keepSs, ss
```

```
    mov sp, offset START
```

```
    mov ax, seg newstack
```

```
    mov ss, ax
```

```
    push ax
```

```
    push bx
```



```
push cx
push dx
```

```
mov ah, 03h
mov bh, 00h
int 10h
push dx
```

```
mov ah, 02h
mov bh, 00h
mov dx, 0220h
int 10h
```

```
push si
push cx
push ds
mov ax, seg intCount
mov ds, ax
mov si, offset intCount
add si, 27
mov cx, 4
```

```
loop_m:
    mov ah, [si]
    inc ah
    mov [si], ah
    cmp ah, 3ah
    jne PRINT_TIMER
    mov ah, 30h
    mov [si], ah
    dec si
    loop loop_m
```

```
PRINT_TIMER:
    pop ds
    pop cx
    pop si

    push es
```

```
push bp
mov ax, SEG intCount
mov es, ax
mov ax, offset intCount
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, keepSs
mov ax, keepAx
mov sp, keepSp
```

```
iret
```

```
INTERRUPTION_TIMER ENDP
```

```
MEMORY_AREA:
```

```
IS_INT_SETTED 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 INT_IS_SET
        mov al, 00h
        jmp POP_REG

INT_IS_SET:
        mov al, 01h
        jmp POP_REG

POP_REG:
        pop es
        pop dx
        pop bx

        ret
IS_INT_SETTED ENDP

IS_THERE_COMMAND PROC NEAR
        push es

        mov ax, pspAddress0
        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_THERE_COMMAND ENDP


LOAD_INT PROC NEAR
        push ax
        push bx
        push dx
        push es

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

        mov keepIp, bx
        mov keepCs, es

        push ds
        mov dx, offset INTERRUPTION_TIMER
        mov ax, seg INTERRUPTION_TIMER
        mov ds, ax

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

        mov dx, offset INT_IS_LOADING
        call PRINT

```

```
    pop es
    pop dx
    pop bx
    pop ax

    ret
LOAD_INT ENDP
```

```
DELETE_INT 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 INT_WAS_RESTORED
    call PRINT

    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
DELETE_INT ENDP

PRINT PROC NEAR
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
PRINT ENDP

MAIN PROC FAR
    mov bx, 02Ch
    mov ax, [bx]
    mov pspAddres1, ax
    mov pspAddres0, ds
    sub ax, ax
    xor bx, bx

    mov ax, DATA
    mov ds, ax

    call IS_THERE_COMMAND
    cmp al, 01h
    je CALL_DEL_INT

    call IS_INT_SETTED

```

```

    cmp al, 01h
    jne INTERRUPTION_IS_NOT_LOADED

    mov dx, offset INT_IS_LOADED
    call PRINT
    jmp EXIT

    mov ah, 4Ch
    int 21h

INTERRUPTION_IS_NOT_LOADED:
    call LOAD_INT

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

    mov ax, 3100h
    int 21h

CALL_DEL_INT:
    call IS_INT_SETTED
    cmp al, 00h
    je INT_IS_NOT_SETTED
    call DELETE_INT
    jmp EXIT

INT_IS_NOT_SETTED:
    mov dx, offset INT_NOT_SETTED
    call PRINT
    jmp EXIT

EXIT:
    mov ah, 4Ch
    int 21h

MAIN ENDP

```

```
CODE ENDS
```

```
DATA SEGMENT
```

```
    INT_NOT_SETTED      db      "int not loaded",      0dh, 0ah,  
'$'  
    INT_WAS_RESTORED    db      "int was restored",     0dh, 0ah,  
'$'  
    INT_IS_LOADED       db      "int is loaded",        0dh, 0ah,  
'$'  
    INT_IS_LOADING      db      "int is loading",       0dh, 0ah,  
'$'
```

```
DATA ENDS
```

```
END MAIN
```