

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ЛАБОРАТОРНАЯ РАБОТА № 4
по дисциплине «Операционные системы»
ТЕМА: Обработка стандартных прерываний.

Студентка гр. 9381

Андрух И.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 и выполняет соответствующие действия.

В данной 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.

Ход работы.

1) Написала и отладила программный модуль типа .EXE, который выполняет следующие функции:

- 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 о функции 4Ch прерывания int 21h.
- Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h.
- Выгрузка прерывания о 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.

2) Далее запустила отлаженную программу и убедилась, что резидентный обработчик прерывания 1Ch установлен. Сделала так, чтобы работа прерывания отображалась на экране, а также проверила размещение прерывания в памяти. Для этого запустила программу лабораторной работы №3, которая отображает карту памяти в виде списка блоков МСВ.

3) Запустила отлаженную программу еще раз и убедилась, что она определяет установленный обработчик прерываний.

4) Запустила отлаженную программу с ключом выгрузки и убедилась, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также запустила программу лабораторной работы №3.

Сведения о функциях и структурах данных управляющей программы:

Название переменных	Описание
PSP_ADDRESS_1 (dw)	Переменная для хранения старого значения ES до того, как программа была оставлена резидентной в памяти
KEEP_CS (dw)	Переменная для хранения сегмента прерывания
KEEP_IP (dw)	Переменная для хранения смещения прерывания
MY_INTERRUPT_SET (dw)	Переменная для хранения количества вызванных прерываний
INT_COUNT (db)	Хранит текст: Interrupts call count:
M_INT_NOT_SET (db)	Хранит текст: Interruption didnt load!
M_INT_RESTORED (db)	Хранит текст: Interruption was restored!
M_INT_ISLOADED (db)	Хранит текст: Interruption already load!
M_INT_ISLOADED0 (db)	Хранит текст: Interuption is loading now!

Названия функций	Описание
INTERRUPTION	Собственный обработчик прерывания. Выводит количество прерываний, которые были вызваны.
IS_INTERRUPTION_SET	Проверка установлен ли разработанный вектор прерывания
CHECK_LOAD	Загрузка или выгрузка(проверка параметра un)
LOAD_INTERRUPTION	Устанавливает новые обработчики прерывания, используя функцию 25h прерывания int 21h
UNLOAD_INTERRUPTION	Восстанавливает сохранённые заранее обработчики прерываний и выгружает резидентную программу
PRINT_STRING	Печать строки на экран

Результат работы программы.

1) При первом запуске lab4.exe, прерывание не было установлено. Восстановила прерывание, используя параметр /un. В результате появилось сообщение, так как нечего ещё восстанавливать.

```
C:\>lab4.exe
Interruption is loading now!
```

```
C:\>lab4.exe /un
Interruption was restored!
```

```
C:\>lab4.exe /un
Interruption didnt load!
```

2) Запустила lab4.exe без параметра. В верхней части окна расположен счётчик, в котором показано, сколько раз было вызвано прерывание. Счетчик продолжает работать и после завершения программы.

```

C:\>lab4.exe /un
Interrupt was restored!

C:\>lab4.exe /un
Interruption didnt load!

C:\>lab4.exe
Interuption is loading now!

C:\>lab3_2
Amount of available memory:      648128 b
Size of extended memory:      15360 Kb
List of memory control blocks:
MCB type: 4Dh   PSP adress: 0008h      Size:      16 b
MCB type: 4Dh   PSP adress: 0000h      Size:      64 b
MCB type: 4Dh   PSP adress: 0040h      Size:     256 b
MCB type: 4Dh   PSP adress: 0192h      Size:     144 b
MCB type: 4Dh   PSP adress: 0192h      Size:     608 b      LAB4
MCB type: 4Dh   PSP adress: 01C3h      Size:     144 b
MCB type: 4Dh   PSP adress: 01C3h      Size:     816 b      LAB3_2
MCB type: 5Ah   PSP adress: 0000h      Size:    647296 b      x ■< t+x

```

3) Запустила модуль лабораторной работы №3, для проверки, осталось ли прерывание в памяти:

```

C:\>lab3_2
Amount of available memory:      648128 b
Size of extended memory:      15360 Kb
List of memory control blocks:
MCB type: 4Dh   PSP adress: 0008h      Size:      16 b
MCB type: 4Dh   PSP adress: 0000h      Size:      64 b
MCB type: 4Dh   PSP adress: 0040h      Size:     256 b
MCB type: 4Dh   PSP adress: 0192h      Size:     144 b
MCB type: 4Dh   PSP adress: 0192h      Size:     608 b      LAB4
MCB type: 4Dh   PSP adress: 01C3h      Size:     144 b
MCB type: 4Dh   PSP adress: 01C3h      Size:     816 b      LAB3_2
MCB type: 5Ah   PSP adress: 0000h      Size:    647296 b      x ■< t+x

```

4) Повторила запуск lab4.exe. Высветилось сообщение о том, что прерывание наше уже находится в памяти.

```

C:\>lab4.exe
Interruption already load!

```

5) Запустила lab4.exe с параметром /un, для восстановления стандартного обработчика прерывания.

```

C:\>lab4.exe /un
Interruption was restored!

```

6) Запустила 3_2.com для проверки того, что память была освобождена.

```
C:\>lab3_2
Amount of available memory:      648912 b
Size of extended memory:        15360 Kb
List of memory control blocks:
MCB type: 4Dh   PSP adress: 0008h   Size:      16 b
MCB type: 4Dh   PSP adress: 0000h   Size:       64 b
MCB type: 4Dh   PSP adress: 0040h   Size:     256 b
MCB type: 4Dh   PSP adress: 0192h   Size:     144 b
MCB type: 4Dh   PSP adress: 0192h   Size:     816 b
MCB type: 5Ah   PSP adress: 0000h   Size:   648080 b
```

LAB3_2
||@h

Выводы.

В ходе работы был построен обработчик прерывания от сигналов таймера. Изучены дополнительные функции работы с памятью: установка программы-резидента и его выгрузка из памяти.

Ответы на контрольные вопросы:

1. Как реализован механизм прерывания от часов?

Сначала сохраняется содержимое регистров, потом определяется источник прерывания, по номеру которого определяется смещение в таблице векторов прерывания, сохраняется в CS:IP, передаётся управление по адресу CS:IP и происходит выполнение обработчика, и в конце происходит возврат управления прерванной программе. Аппаратное прерывание от таймера происходит каждые 55 мс.

2. Какого типа прерывания использовались в работе?

- 1) аппаратные прерывания
- 2) прерывания функций DOS(21h)
- 3) прерывания функций BIOS(10h)

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

```
LAB4 SEGMENT
    ASSUME CS:LAB4, DS:DATA, SS:STACK

INTERRUPTION PROC FAR
    jmp FSTART

    PSP_ADDRESS_0 dw 0
    PSP_ADDRESS_1 dw 0
    KEEP_CS dw 0
    KEEP_IP dw 0
    MY_INTERRUPT_SET dw 0FEDCh
    INT_COUNT db 'Interrupts call count: 0000 $'

    KEEP_SS dw ?
    KEEP_SP dw ?
    KEEP_AX dw ?
    INT_STACK dw 64 dup (?)
    END_INT_STACK dw ?

FSTART:

    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 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 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_CALC
    mov ah, 30h
    mov [si], ah

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

    mov ch, [si - 2]
```



```

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

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

END_CALC:
        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
        mov AL, 20H
        out 20H, AL

        iret
INTERRUPTION ENDP

NEED_MEM_AREA PROC
NEED_MEM_AREA ENDP

IS_INTERRUPTION_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 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_INTERRUPTION_SET ENDP

CHECK_LOAD 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
CHECK_LOAD ENDP

LOAD_INTERRUPTION 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 INTERRUPTION
        mov ax, seg INTERRUPTION
        mov ds, ax

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

    mov dx, offset M_INT_ISLOADING
    call PRINT_STRING

    pop es
    pop dx
    pop bx
    pop ax

    ret
LOAD_INTERRUPTION ENDP

UNLOAD_INTERRUPTION 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 M_INT_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_INTERRUPTION ENDP

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

MAIN_PROGRAM PROC FAR
    mov bx, 02Ch
    mov ax, [bx]
    mov PSP_ADDRESS_1, ax
    mov PSP_ADDRESS_0, ds
    sub ax, ax
    xor bx, bx

    mov ax, DATA
    mov ds, ax

    call CHECK_LOAD
    cmp al, 01h
    je UNLOAD_START

    call IS_INTERRUPTION_SET
    cmp al, 01h
    jne INTERRUPTION_IS_NOT_LOADED

    mov dx, offset M_INT_ISLOADED
    call PRINT_STRING
    jmp EXIT_PROGRAM

    mov ah, 4Ch
    int 21h

INTERRUPTION_IS_NOT_LOADED:
    call LOAD_INTERRUPTION

```

```

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

        mov ax, 3100h
        int 21h

UNLOAD_START:
        call IS_INTERRUPTION_SET
        cmp al, 00h
        je INT_IS_NOT_SET
        call UNLOAD_INTERRUPTION
        jmp EXIT_PROGRAM

INT_IS_NOT_SET:
        mov dx, offset M_INT_NOT_SET
        call PRINT_STRING
        jmp EXIT_PROGRAM

EXIT_PROGRAM:
        mov ah, 4Ch
        int 21h
MAIN_PROGRAM ENDP

LAB4 ENDS
STACK SEGMENT STACK
        db 64 DUP(?)
STACK ENDS

DATA SEGMENT
        M_INT_NOT_SET db "Interruption did not load!", 0dh, 0ah, '$'
        M_INT_RESTORED db "Interruption was restored!", 0dh, 0ah, '$'
        M_INT_ISLOADED db "Interruption has already loaded!", 0dh, 0ah, '$'
        M_INT_ISLOADING db "Interruption is loading now!", 0dh, 0ah, '$'
DATA ENDS

END MAIN_PROGRAM

```