

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.

Студент гр. 9381

Николаев А.А.

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.

Ход работы.

Был написан программный модуль .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.

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)	“Interrupts call count:”
M_INT_NOT_SET (db)	“Interruption didnt load!”
M_INT_RESTORED (db)	“Interruption was restored!”
M_INT_ISLOADED (db)	“Interruption already load!”
M_INT_ISLOADED0 (db)	“Interuption is loading now!”

Названия функций	Описание
INTERRUPTION	Собственный обработчик прерывания. Выводит количество прерываний, которые были вызваны.
IS_INTERRUPT_SET	Проверка установлен ли разработанный вектор прерывания
CHECK_LOAD	Загрузка или выгрузка(проверка параметра un)
LOAD_INTERRUPT	Устанавливает новые обработчики прерывания, используя функцию 25h прерывания int 21h
UNLOAD_INTERRUPT	Восстанавливает сохранённые заранее обработчики прерываний и выгружает резидентную программу
PRINT_STRING	Вывод строки в консоль.

Тестирование программы.

Программа запущена, выводится строка отображающая количество вызовов прерывания.

```
Directory c:\MASM
Interrupts call count: 0450

C:\>lab4.EXE
Interruption is loading now!

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

C:\>lab4.EXE /un
Interruption did not load!
```

Была выведена строки о том, что прерывание загружено.

При запуске lab4.EXE с параметром /un, программа восстановила стандартный обработчик прерывания.

При повторном вызове предыдущей команды, программа говорит, что прерывание не загружено.

```
C:\>lab4.EXE
Interruption has already loaded!
```

При повторном запуске программы, говорится, что прерывание уже загружено.

```
Amount of available memory: 647952 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: 784 b LAB4
MCB type: 4Dh PSP address: 01CEh Size: 144 b
MCB type: 4Dh PSP address: 01CEh Size: 784 b LAB3_2
MCB type: 5Ah PSP address: 0000h Size: 647152 b Z[X|P| =
```

При выводе списка MCB блоков видно, что обработчик оказался в памяти.

Далее, программа была запущена с параметром /un, что означает, что мы выгружаем прерывание.

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

```
C:\>lab3_2
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: 4Dh   PSP address: 0192h   Size:   784 b
MCB type: 5Ah   PSP address: 0000h   Size:  648112 b
LAB3_2
C:\>
```

Теперь, как видно, обработчик прерываний выгружен.

Контрольные вопросы

1. Как реализован механизм прерывания от часов?

После поступления сигнала прерывания, который вызывается каждые 55 мс сохраняется содержимое регистров, затем 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 потом переходит возврат управления прерванной программе.

2. Какого типа прерывания использовались в работе?

Аппаратные прерывания 1Ch и программные int21h и int10h.

Вывод

Была исследована обработка прерываний и построен обработчик прерываний сигналов таймера.

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

```