

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. 8383

Мололкин К.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

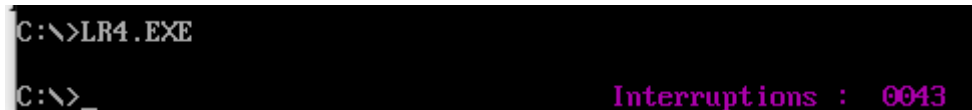
2020

Цель работы

Построить обработчик прерываний сигналов таймера, которые генерируются аппаратурой через определенные интервалы времени. При возникновении такого сигнала, возникает прерывание с определенным значением вектора.

Ход работы

Был написан и отлажен программный модуль .EXE, который проверяет если параметр /up был передан через командную строку, и записывает соответствующий результат в переменную-флаг IS_UN_FLAG, затем проверяет установлено ли пользовательское прерывание с вектором 1Ch, если установлено, то проверяется значение IS_UN_FLAG, если оно 1, то прерывание выгружается, если 0, то выводится сообщение о попытке повторно загрузить прерывание, если прерывание не установлено и значение флага 1, то выводится сообщение о попытке выгрузить неустановленное прерывание, если 0, то устанавливает резидентную функцию по обработке прерывания и настраивает вектор прерывания. На рис 1. представлен пример работы программы



```
C:\>LR4.EXE
C:\>_ Interruptions : 0043
```

Рисунок 1 – пример работы программы

На следующем шаге было проверено размещение прерывания в памяти с помощью программы LR3_1.COM, для этого сначала была запущена программа LR4.EXE, а затем LR3_1.COM, результат работы двух программ представлен на рис. 2.

```
Available memory: 648320
Extended memory: 245760

MS DOS part
Size: 16

Free part
Size: 64

0040
Size: 256

0192
Size: 144

0192
Size: 416 LR4

01B7
Size: 144

01B7
Size: 648320 LR3_1
C:\>_

Interruptions : 0202
```

Рисунок 2 – проверка размещения прерывания в памяти

Как видно из рисунка, программа LR4.EXE, размещается в 5 по счету блоке.

Для проверки того, что программа определяет установленный обработчик прерывания, программа LR4.EXE еще раз. Результат повторного запуска представлен на рис. 3.

```
C:\>lr4.exe
Trying to load interruption again

Interruptions : 0311
```

Рисунок 3 – проверка программы на определение запущенного прерывания

Затем для проверки выгрузки резидентного обработчика прерывания, программа была запущена с ключом выгрузки “/un”. На рис.4 представлен результат запуска программы с ключом, а на рис. 5 представлен результат запуска программы LR3_1.COM, для проверки выгрузки программы из памяти. Код программы LR4.ASM представлен в приложении А.

```
C:\>lr4.exe /un

Interruptions : 0816

C:\>
```

Рисунок 4 – результат выгрузки программы

```
Available memory: 648912
Extended memory: 245760

MS DOS part
Size: 16

Free part
Size: 64

0040
Size: 256

0192
Size: 144

0192
Size: 648912 LR3_1
```

Рисунок 5 – результат выгрузки программы из памяти

Контрольные вопросы

1. Как реализован механизм прерывания от часов?

Аппаратное прерывание 1Ch, возникающее каждые 55мс, вызывает резидентный обработчик, процессор переключается на выполнение кода обработчика, а затем возвращается на выполнение прерванной программы. Адрес возврата в прерванную программу (CS:IP) запоминается в стеке вместе с регистром флагов. Затем в CS:IP загружается адрес точки входа программы обработки прерывания и начинает выполняться его код.

2. Какого типа прерывания использовались в работе?

В программе используются аппаратное (1Ch) и программные (10h, 21h) прерывания.

Вывод

Во время выполнения лабораторной работы был написан обработчик прерываний сигналов таймера. Исследована загрузка обработчика прерывания в память.

Приложение А

```
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:ASTACK, ES:NOTHING

ROUT PROC FAR
    jmp START_ROUT
    INTER_NUMB db 'Interruptions : 0000'
    INTER_ID dw 7373h
    KEEP_IP dw 0
    KEEP_CS dw 0
    KEEP_SS dw 0
    KEEP_SP dw 0
    KEEP_AX dw 0
    KEEP_PSP dw 0
    ROUT_STACK dw 128 DUP(0)

START_ROUT:
    mov KEEP_SS, ss
    mov KEEP_SP, sp
    mov KEEP_AX, ax
    mov ax, seg ROUT_STACK
    mov ss, ax
    mov ax, offset ROUT_STACK
    add ax, 256
    mov sp, ax

    push ax
    push bx
    push ds
    push dx
    push si
    push cx
    push es

    call getCurs
    push dx
    call setCurs
    mov ax, seg INTER_NUMB
    push ds
    mov ds, ax
    mov si, offset INTER_NUMB
    add si, 20
    mov cx, 4

COUNT_INTER_LOOP:
    mov ah, [si]
    inc ah
    mov [si], ah
    cmp ah, ':'
    jne END_LOOP
    mov ah, '0'
```

```

        mov [si], ah
        dec si
        loop COUNT_INTER_LOOP

END_LOOP:
        pop ds
        push es
        push bp
        mov ax, seg INTER_NUMB
        mov es, ax
        mov bp, offset INTER_NUMB

        mov ah, 13h
        mov al, 1h
        mov bl, 5h
        mov cx, 21
        mov bh, 0
        int 10h
        pop bp
        pop es

        pop dx
        mov ah, 02h
        mov bh, 0h
        int 10h

        pop es
        pop cx
        pop si
        pop dx
        pop ds
        pop bx
        pop ax
        mov sp, KEEP_SP
        mov ax, KEEP_AX
        mov ss, KEEP_SS
        mov al, 20h
        out 20h, al
        IRET
ROUT ENDP
ROUT_END:

getCurs proc
        mov ah, 03h
        mov bh, 0
        int 10h
        ret
getCurs ENDP

setCurs proc
        mov ah, 02h
        mov bh, 00h
        mov dl, 25h

```

```

        mov dh, 18h
        int 10h
        ret
setCurs ENDP

START_OR_STOP_INTER PROC
    push ax
    push bx
    push si
    push es
    mov ax, KEEP_PSP
    mov es, ax
    cmp byte ptr es:[82h], '/'
    jne IS_LOAD
    cmp byte ptr es:[83h], 'u'
    jne IS_LOAD
    cmp byte ptr es:[84h], 'n'
    jne IS_LOAD
    mov IS_UN_FLAG, 1

IS_LOAD:
    mov ah, 35h
    mov al, 1Ch
    int 21h
    mov si, offset INTER_ID
    sub si, offset ROUT
    mov ax, es:[bx+si]
    cmp ax, 7373h
    jne NOT_LOAD
    cmp IS_UN_FLAG, 1
    jne LOAD_AGAIN
    call STOP_INTER
    jmp END_SS

LOAD_AGAIN:
    mov dx, offset TRY_LOAD_AGAIN
    call PRINT_STRING
    jmp END_SS

NOT_LOAD:
    cmp IS_UN_FLAG, 1
    jne BEGIN_ROUT
    mov dx, offset TRY_TO_STOP
    call PRINT_STRING
    jmp END_SS

BEGIN_ROUT:
    call LOAD_INTER

END_SS:

    pop es
    pop si

```

```

        pop bx
        pop ax
        ret
START_OR_STOP_INTER ENDP

LOAD_INTER PROC
        push ax
        push bx
        push cx
        push dx
        push ds
        push es

        mov ah, 35h
        mov al, 1Ch
        int 21h
        mov KEEP_CS, es
        mov KEEP_IP, bx
        push ds
        mov dx, offset ROUT
        mov ax, SEG ROUT
        mov ds, ax
        mov ah, 25h
        mov al, 1Ch
        int 21h
        pop ds
        mov dx, offset ROUT_END
        add dx, 10Fh
        mov cl, 4h
        shr dx, cl
        inc dx
        xor ax, ax
        mov ah, 31h
        int 21h

        pop es
        pop ds
        pop dx
        pop cx
        pop bx
        pop ax
        ret
LOAD_INTER ENDP

STOP_INTER PROC
        CLI
        push ax
        push bx
        push dx
        push ds
        push es
        push si

```



```

    mov ah, 35h
    mov al, 1Ch
    int 21h
    mov si, offset KEEP_IP
    sub si, offset ROUT
    mov dx, es:[bx + si]
    mov ax, es:[bx + si + 2]
    push ds
    mov ds, ax
    mov ah, 25h
    mov al, 1Ch
    int 21h
    pop ds
    mov ax, es:[bx + si + 4]
    mov es, ax
    push es
    mov ax, es:[2Ch]
    mov es, ax
    mov ah, 49h
    int 21h
    pop es
    mov ah, 49h
    int 21h

    pop si
    pop es
    pop ds
    pop dx
    pop bx
    pop ax
    STI
    ret
STOP_INTER ENDP

PRINT_STRING PROC near
    push AX
    mov ah, 09h
    int 21h
    pop AX
    ret
PRINT_STRING ENDP

MAIN PROC
    xor ax, ax
    mov ax, DATA
    mov ds, ax
    mov KEEP_PSP, es
    call START_OR_STOP_INTER
    xor al, al
    mov ah, 4Ch
    int 21h
MAIN ENDP

```

```
CODE ENDS

ASTACK SEGMENT STACK
    dw 128 DUP(0)
ASTACK ENDS

DATA SEGMENT
    IS_UN_FLAG db 0
    INTER_LOAD_FLAG db 0
    TRY_TO_STOP db "Trying to stop not loaded interruption$"
    TRY_LOAD_AGAIN db "Trying to load interruption again$"
DATA ENDS
END MAIN
```