

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

Студентка гр. 0382

Здобнова К.Д.

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 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.

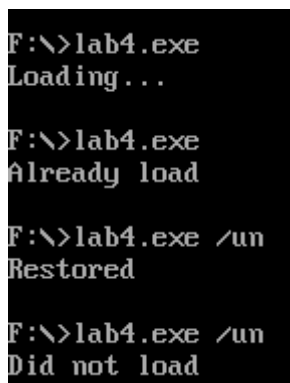
Шаг 4. Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

Шаг 5. Ответьте на контрольные вопросы.

Выполнение работы.

Был написан и отлажен программный модуль .EXE, проверяющий установлено ли собственное прерывание с вектором 1Ch, устанавливающий резидентную функцию, если не установлено, иначе – вывод соответствующего сообщения. Также, реализована выгрузка прерывания, при передаче в командной строке параметра /un – восстанавливает стандартный вектор прерывания.

Убедимся в работе нашей программы, результат выполнения:



```
F:\>lab4.exe
Loading...

F:\>lab4.exe
Already load

F:\>lab4.exe /un
Restored

F:\>lab4.exe /un
Did not load
```

Рисунок 1.

После чего, убедимся на примере лабораторной работы, что прерывание используется (на экране отобразим значение interrupts call count), используя ранее написанный код под лабораторную работу №3. Для этого запустим снова собственный обработчик прерываний, после чего проверим на примере лабораторной работы №3, что он определяется. Далее выгрузим обработчик и снова запустим модуль лабораторной работы №3.

Результат выполнения:

```

F:\>lab4.exe
Loading...
Interrupts call count: 0137

F:\>lab3_2.com
Available memory: 648112 b
Extended memory: 15360 Kb
Memory control blocks:
MCB type: 4Dh PSP address: 0008h Size: 16 b
MCB type: 4Dh PSP address: 0000h Size: 64 b DPMILOAD
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: 624 b LAB4
MCB type: 4Dh PSP address: 01C4h Size: 144 b
MCB type: 4Dh PSP address: 01C4h Size: 784 b LAB3_2
MCB type: 5Ah PSP address: 0000h Size: 647312 b te fret

```

Рисунок 2.

```

F:\>lab4.exe /un
Restored

F:\>lab3_2.com
Available memory: 648912 b
Extended memory: 15360 Kb
Memory control blocks:
MCB type: 4Dh PSP address: 0008h Size: 16 b
MCB type: 4Dh PSP address: 0000h Size: 64 b DPMILOAD
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 LAB3_2
MCB type: 5Ah PSP address: 0000h Size: 648112 b LAB4

```

Рисунок 3.

Ответы на контрольные вопросы

1) Как реализован механизм прерывания от часов?

Когда происходит сигнал часов, то сохраняется состояние регистров для того, чтобы вернуться в текущую программу, затем определяется источник прерывания, из вектора прерывания считываются CS и IP обработчика прерывания, прерывание обрабатывается, затем управление возвращается прерванной программе.

1) Какого типа прерывания использовались в работе?

Аппаратные: 1Ch; программные: 10h, 21h

Вывод

Был изучен механизм обработки прерываний в DOS, написана программа, которая заменяет текущий обработчик прерываний от таймера на пользовательский.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab4.asm

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

```
MY_STACK SEGMENT STACK
    DW 64 DUP(?)
MY_STACK ENDS
```

```
CODE SEGMENT
```

```
print PROC NEAR
```

```
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
```

```
print ENDP
```

```
interr PROC FAR
```

```
    jmp start
    PSP_ADDRESS_0 dw 0 ;3
    PSP_ADDRESS_1 dw 0 ;5
    KEEP_CS dw 0 ;7
    KEEP_IP dw 0 ;9
    interr_set dw 0FEDCh ;11
    count db 'Interrupts call count: 0000 '$' ;13
```

```
start:
```

```
    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 count
    mov ds, ax
    mov si, OFFSET count
    add si, 1Ah
```

```
    mov ah,[si]
    inc ah
    mov [si], ah
    cmp ah, 3Ah
    jne endFunc
    mov ah, 30h
```

```

        mov [si], ah

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

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

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

endFunc:
        pop ds
        pop cx
        pop si

        push es
        push bp

        mov ax, SEG count
        mov es, ax
        mov ax, OFFSET 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

        iret

interr ENDP

mem_check PROC
mem_check ENDP

isSet 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 isSetCheck
        mov al, 00h
        jmp ignoreIsSetCheck

isSetCheck:
        mov al, 01h
        jmp ignoreIsSetCheck

ignoreIsSetCheck:
        pop es
        pop dx
        pop bx

        ret
isSet ENDP

checkUN PROC NEAR
        push es

        mov ax, PSP_ADDRESS_0
        mov es, ax

        mov bx, 0082h

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

        mov al, es:[bx]
        inc bx
        cmp al, 'u'
        jne UNend

        mov al, es:[bx]
        inc bx
        cmp al, 'n'
        jne UNend

        mov al, 0001h
UNend:
        pop es

        ret
checkUN ENDP

loadInt 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 interr
    mov ax, seg interr
    mov ds, ax

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

    pop ds

    mov dx, OFFSET Loading
    call print

    pop es
    pop dx
    pop bx
    pop ax

    ret
loadInt ENDP

UNloadInt 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 IsRestored
    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
UnloadInt ENDP

main 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 checkUN
    cmp al, 01h
    je UnloadCheck

    call isSet
    cmp al, 01h
    jne NotLoadedCheck

    mov dx, OFFSET IsLoaded
    call print
    jmp ExitCheck

    mov ah, 4Ch
    int 21h

NotLoadedCheck:
    call loadInt

    mov dx, OFFSET mem_check
    mov cl, 04h
    shr dx, cl
    add dx, 1Bh

    mov ax, 3100h
    int 21h

UnloadCheck:
    call isSet
    cmp al, 00h
    je NotSetCheck
    call UNloadInt
    jmp ExitCheck

NotSetCheck:
    mov dx, OFFSET NotSet
    call print
    jmp ExitCheck

ExitCheck:
    mov ah, 4Ch
    int 21h
main ENDP

```

```
CODE ENDS
DATA SEGMENT
    NotSet db "Did not load", 0dh, 0ah, '$'
    IsRestored db "Restored", 0dh, 0ah, '$'
    IsLoaded db "Already load", 0dh, 0ah, '$'
    Loading db "Loading...", 0dh, 0ah, '$'
DATA ENDS
END main
```