

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. 9382

Павлов Р.В.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Постановка задачи.

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.

Функции и структуры данных:

Название процедуры	Описание процедуры
main	Вызов реализованных процедур, загрузка и выгрузка прерываний
print_string	Вывод на экран указанной строки
inter	Реализация обработчика прерывания
is_loaded	Проверка на наличие прерывания в памяти
is_tounld	Проверка на флаг выгрузки
load	Загрузка прерывания в память
unload	Выгрузка прерывания из памяти
loaded	Флаг наличия прерывания в памяти
tounld	Флаг выгрузки
msg_loading	Сообщение об успешной загрузке
msg_loaded	Сообщение об уже загруженном прерывании

msg_unloading	Сообщение об успешной выгрузке
msg_unloaded	Сообщение об отсутствии прерывания в памяти

Последовательность действий:

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.

Ход работы.

1. Был реализован пользовательский обработчик прерываний от таймера, который хранится в памяти после завершения программы резидентно. На экран выводится количество вызовов прерывания. При запуске обработчик помещается в память и хранится там до указания выгрузки.

```
C:\>lab4.exe
Прерывание загружено в память
C:\>_
```

0053 вызовов

2. Осуществлена проверка памяти:

```
C:\>lab4.exe 1386 вызовов
Прерывание загружено в память

C:\>lab3
Доступный размер памяти: 644336 байт
Размер расширенной памяти: 246720 байт
Блок№ 001 Адрес: 016F; PSP: 0008; Размер: 16 байт; SC/CD:
Блок№ 002 Адрес: 0171; PSP: 0000; Размер: 64 байт; SC/CD:
Блок№ 003 Адрес: 0176; PSP: 0040; Размер: 256 байт; SC/CD:
Блок№ 004 Адрес: 0187; PSP: 0192; Размер: 144 байт; SC/CD:
Блок№ 005 Адрес: 0191; PSP: 0192; Размер: 4400 байт; SC/CD: LAB4
Блок№ 006 Адрес: 02A5; PSP: 02B0; Размер: 144 байт; SC/CD:
Блок№ 007 Адрес: 02AF; PSP: 02B0; Размер: 644336 байт; SC/CD: LAB3

C:\>
```

Видно, что обработчик помещён в память.

3. Повторный запуск программы привёл к выводу на экран сообщения о том, что обработчик уже находится в памяти.

```
C:\>lab4.exe 3247 вызовов
Прерывание загружено в память

C:\>lab3
Доступный размер памяти: 644336 байт
Размер расширенной памяти: 246720 байт
Блок№ 001 Адрес: 016F; PSP: 0008; Размер: 16 байт; SC/CD:
Блок№ 002 Адрес: 0171; PSP: 0000; Размер: 64 байт; SC/CD:
Блок№ 003 Адрес: 0176; PSP: 0040; Размер: 256 байт; SC/CD:
Блок№ 004 Адрес: 0187; PSP: 0192; Размер: 144 байт; SC/CD:
Блок№ 005 Адрес: 0191; PSP: 0192; Размер: 4400 байт; SC/CD: LAB4
Блок№ 006 Адрес: 02A5; PSP: 02B0; Размер: 144 байт; SC/CD:
Блок№ 007 Адрес: 02AF; PSP: 02B0; Размер: 644336 байт; SC/CD: LAB3

C:\>lab4.exe
Прерывание уже загружено

C:\>_
```

4. Программа запущена с параметром -u, обработчик удалён из памяти, что видно на скриншоте ниже.

```
Размер расширенной памяти: 246720 байт
Блок№ 001 Адрес: 016F; PSP: 0008; Размер: 16 байт; SC/CD:
Блок№ 002 Адрес: 0171; PSP: 0000; Размер: 64 байт; SC/CD:
Блок№ 003 Адрес: 0176; PSP: 0040; Размер: 256 байт; SC/CD:
Блок№ 004 Адрес: 0187; PSP: 0192; Размер: 144 байт; SC/CD:
Блок№ 005 Адрес: 0191; PSP: 0192; Размер: 4400 байт; SC/CD: LAB4
Блок№ 006 Адрес: 02A5; PSP: 02B0; Размер: 144 байт; SC/CD:
Блок№ 007 Адрес: 02AF; PSP: 02B0; Размер: 644336 байт; SC/CD: LAB3

C:\>lab4.exe
Прерывание уже загружено

C:\>lab4.exe -u
Прерывание выгружено из памяти

C:\>lab3
Доступный размер памяти: 648912 байт
Размер расширенной памяти: 246720 байт
Блок№ 001 Адрес: 016F; PSP: 0008; Размер: 16 байт; SC/CD:
Блок№ 002 Адрес: 0171; PSP: 0000; Размер: 64 байт; SC/CD:
Блок№ 003 Адрес: 0176; PSP: 0040; Размер: 256 байт; SC/CD:
Блок№ 004 Адрес: 0187; PSP: 0192; Размер: 144 байт; SC/CD:
Блок№ 005 Адрес: 0191; PSP: 0192; Размер: 648912 байт; SC/CD: LAB3

C:\>_
```

Ответы на контрольные вопросы.

Сегментный адрес недоступной памяти

1. На первый и последующие байты памяти, недоступные программе.
2. Она находится за блоком памяти, выделенным программе.
3. Туда можно что-то записать, но делать этого не стоит.

Среда, передаваемая программе

1. Среда — это область памяти, в которой хранится массив строк вида «имя = значение» (переменных среды), которые служат вызываемой программе параметрами.

2. Среда создаётся при загрузке MS DOS, копируется при загрузке программы в память до передачи управления этой программе.

3. Переменные среды берутся из командного файла **autoexec.bat**, который также выполняется при запуске ОС и отвечает за настройку параметров среды.

Выводы.

В результате выполнения лабораторной работы был исследован интерфейс управляющей программы и загрузочных модулей, изучены некоторые фрагменты структуры PSP, исследована среда программы.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД

- имя файла : lab4.asm

```
Astack segment stack
```

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

```
Astack ends
```

```
data segment
```

```
    loaded db 0
```

```
    tounld db 0
```

```
    msg_loading db "Прерывание загружено в память",10,13,"$"
```

```
    msg_loaded db "Прерывание уже загружено",10,13,"$"
```

```
    msg_unloading db "Прерывание выгружено из памяти",10,13,"$"
```

```
    msg_unloaded db "Нет загруженного прерывания",10,13,"$"
```

```
data ends
```

```
code segment
```

```
    assume cs:code, ds:data, ss:Astack
```

```
inter proc far
```

```
    jmp init
```

```
    routdata:
```

```
    counter db "0000 вызовов"
```

```
    id dw 2228h
```

```
    save_cs dw 0
```

```
    save_ip dw 0
```

```
    save_psp dw 0
```

```
    save_ax dw 0
```

```
    save_ss dw 0
```

```
    save_sp dw 0
```

```
    substack dw 64 dup(?)
```

```
init:
```

```
    mov save_ax, ax
```

```
    mov save_sp, sp
```

```
    mov save_ss, ss
```



```

    mov ax, seg substack
    mov ss, ax
    mov ax, offset substack
    add ax, 128
    mov sp, ax

    push ax
    push bx
    push cx
    push dx
    push si
    push es
    push ds
    mov ax, seg counter
    mov ds, ax

    mov ah, 03h
    mov bh, 0h
    int 10h
    push dx

    mov ah, 02h
    mov bh, 0h
    mov dx, 0141h
    int 10h

    mov si, offset counter
    add si, 3
    mov cx, 4

increase:
    mov ah, [si]
    cmp ah, '9'
    jnl increased
    sub ah, 9
    mov [si], ah
    dec si
    loop increase

```

```

increaseD:
    cmp cx, 0
    je reset
    inc ah
    mov [si], ah
reset:

push es
push bp
mov ax, seg counter
mov es, ax
mov bp, offset counter
mov ah, 13h
mov al, 1h
mov bl, 2h
mov bh, 0
mov cx, 12
int 10h

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

pop ds
pop es
pop si
pop dx
pop cx
pop bx
pop ax

mov sp, save_sp
mov ax, save_ss
mov ss, ax
mov ax, save_ax

```

```

        mov al, 20h
        out 20h, al
        iret
inter endp

inter_end:

is_loaded proc
    push ax
    push bx
    push si

    mov ax, 351ch
    int 21h
    mov si, offset id - offset inter
    mov ax, es:[bx + si]
    cmp ax, id
        jne is_loaded_ret
    mov loaded, 1

    is_loaded_ret:
        pop si
        pop bx
        pop ax
        ret
is_loaded endp

is_tounld proc
    push ax
    push es

    mov ax, save_psp
    mov es, ax
    cmp byte ptr es:[82h], '-'
        jne is_tounld_ret
    cmp byte ptr es:[83h], 'u'
        jne is_tounld_ret

```

```

        mov tounld, 1

is_tounld_ret:
        pop es
        pop ax
        ret
is_tounld endp

load proc
        push ax
        push bx
        push cx
        push dx
        push es
        push ds

        mov ah, 35h
        mov al, 1ch
        int 21h
        mov save_cs, es
        mov save_ip, bx
        mov ax, seg inter
        mov ds, ax
        mov dx, offset inter
        mov ah, 25h
        mov al, 1ch
        int 21h
        pop ds

        mov dx, offset inter_end
        mov cl, 4h
        shr dx, cl
        add dx, 100h
        xor ax, ax
        mov ah, 31h
        int 21h

        pop es

```

```

        pop dx
        pop cx
        pop bx
        pop ax
        ret
load endp

unload 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 save_cs
        sub si, offset inter
        mov ax, es:[bx + si]
        mov dx, es:[bx + si + 2]

        push ds
        mov ds, ax
        mov ax, 251ch
        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

        sti

        pop si
        pop es
        pop ds
        pop dx
        pop bx
        pop ax

        ret
unload endp

print_string proc near
        push ax
        mov ah, 09h
        int 21h
        pop ax
        retn
print_string endp

main proc
        push ds
        xor ax, ax
        push ax
        mov ax, data
        mov ds, ax
        mov save_psp, es

        call is_loaded
        call is_tounld
        cmp tounld, 1
            je to_unload
        cmp loaded, 1
            jne to_load
        mov dx, offset msg_loaded

```

```

    call print_string
    jmp main_ret

to_load:
    mov dx, offset msg_loading
    call print_string
    call load
    jmp main_ret

to_unload:
    cmp loaded, 1
    jne not_to_unload
    mov dx, offset msg_unloading
    call print_string
    call unload
    jmp main_ret

not_to_unload:
    mov dx, offset msg_unloaded
    call print_string

main_ret:
    mov     ax, 4c00h
    int     21h

main endp
code ends
end main

```