

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. 0382

Шангичев В. А.

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.

Шаг 1. Для создания требуемого в задании файла .EXE был написан исходный файл `main.asm`.

Описание процедур:

`get_curs` – процедура, которая с помощью прерывания `10h` дает позицию курсора.

`set_curs` – процедура, которая устанавливает позицию курсора с помощью прерывания `10h`.

`rout` – процедура, которая реализует прерывание. Внутри нее в кодовом сегменте задаются необходимые данные. Среди них сигнатура, по которой можно ее идентифицировать, строка, представляющая количество раз, которое прерывание было вызвано, стек и машинные слова для хранения некоторых сегментов. Процедура прибавляет 1 к количеству раз, которое было вызвано прерывание и печатает это количество в первой строке консоли.

`load_rout` – процедура для загрузки прерывания.

`unload_rout` – процедура для выгрузки прерывания.

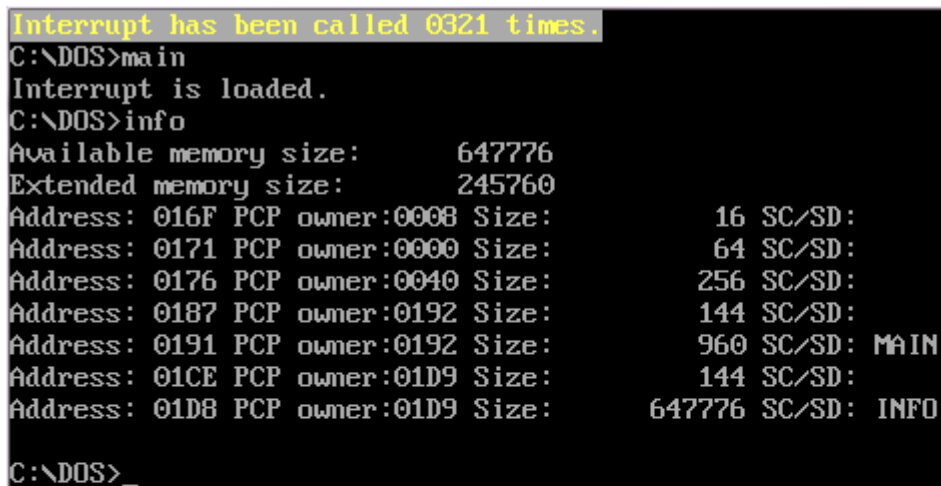
`load_check` – процедура, которая определяет, загружено ли пользовательское прерывание. Ответ возвращается в `al`.

`cmd_flag_check` – процедура, которая определяет, была ли программа запущена с флагом `/un`. Ответ возвращается в `al`.

print – печатает строку.

main – главная процедура. В ней выясняется задача программы и проверяется, корректна ли эта задача (например, если требуется выгрузить прерывание, то это прерывание должно быть загружено), и, если задача корректна, то происходит выполнение задачи.

Шаг 2.



```
Interrupt has been called 0321 times.  
C:\DOS>main  
Interrupt is loaded.  
C:\DOS>info  
Available memory size:      647776  
Extended memory size:      245760  
Address: 016F PCP owner:0008 Size:      16 SC/SD:  
Address: 0171 PCP owner:0000 Size:      64 SC/SD:  
Address: 0176 PCP owner:0040 Size:     256 SC/SD:  
Address: 0187 PCP owner:0192 Size:     144 SC/SD:  
Address: 0191 PCP owner:0192 Size:     960 SC/SD: MAIN  
Address: 01CE PCP owner:01D9 Size:     144 SC/SD:  
Address: 01D8 PCP owner:01D9 Size:  647776 SC/SD: INFO  
  
C:\DOS>_
```

Рисунок 1 - Прерывание было успешно размещено в памяти.

Шаг 3

При попытке загрузить прерывание повторно выводится соответствующее уведомление.

```

Interrupt has been called 1866 times.
C:\DOS>main
Interrupt is loaded.
C:\DOS>info
Available memory size:      647776
Extended memory size:      245760
Address: 016F PCP owner:0008 Size:      16 SC/SD:
Address: 0171 PCP owner:0000 Size:      64 SC/SD:
Address: 0176 PCP owner:0040 Size:     256 SC/SD:
Address: 0187 PCP owner:0192 Size:     144 SC/SD:
Address: 0191 PCP owner:0192 Size:     960 SC/SD: MAIN
Address: 01CE PCP owner:01D9 Size:     144 SC/SD:
Address: 01D8 PCP owner:01D9 Size:    647776 SC/SD: INFO

C:\DOS>main
Interrupt is already loaded.
C:\DOS>

```

Рисунок 2 – программа обнаруживает прерывание в памяти

Шаг 4.

```

C:\DOS>info
Available memory size:      647776
Extended memory size:      245760
Address: 016F PCP owner:0008 Size:      16 SC/SD:
Address: 0171 PCP owner:0000 Size:      64 SC/SD:
Address: 0176 PCP owner:0040 Size:     256 SC/SD:
Address: 0187 PCP owner:0192 Size:     144 SC/SD:
Address: 0191 PCP owner:0192 Size:     960 SC/SD: MAIN
Address: 01CE PCP owner:01D9 Size:     144 SC/SD:
Address: 01D8 PCP owner:01D9 Size:    647776 SC/SD: INFO

C:\DOS>main
Interrupt is already loaded.
C:\DOS>main /un
Interrupt is unloaded.
C:\DOS>info
Available memory size:      648912
Extended memory size:      245760
Address: 016F PCP owner:0008 Size:      16 SC/SD:
Address: 0171 PCP owner:0000 Size:      64 SC/SD:
Address: 0176 PCP owner:0040 Size:     256 SC/SD:
Address: 0187 PCP owner:0192 Size:     144 SC/SD:
Address: 0191 PCP owner:0192 Size:    648912 SC/SD: INFO

C:\DOS>_

```

Рисунок 3 – прерывание было выгружено

Как можно видеть, сообщения больше не выводятся, и прерывание было выгружено из памяти.

Ответы на вопросы.

1. Как реализован механизм прерывания от часов?

18 раз в секунду по тикку аппаратных часов вызывается прерывание 1Ch, которое по умолчанию ничего не делает (содержит только команду `iret`). Для этого прерывания можно создать пользовательский обработчик.

2. Какого типа прерывания использовались в работе?

Аппаратное прерывание – 1Ch.

Программные прерывания – `int10h` и `int21h`.

Также был создан пользовательский обработчик прерываний.

Выводы.

Был построен обработчик прерываний сигналов таймера. Для этого была написана процедура, выводящая в консоль количество раз, которое оно было вызвано. Также было изучено размещение обработчика прерывания в памяти средствами лабораторной работы 3.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.asm

```
AStack    SEGMENT STACK
            DB 256 dup (?)
AStack    ENDS

DATA SEGMENT
    not_loaded_msg db 'Interrupt is not loaded.$', 0dh, 0ah
    loaded_msg db 'Interrupt is loaded.$', 0dh, 0ah
    unloaded_msg db 'Interrupt is unloaded.$', 0dh, 0ah
    already_loaded db 'Interrupt is already loaded.$', 0dh, 0ah

DATA ENDS

CODE    SEGMENT
ASSUME  CS:CODE, DS:DATA, SS:AStack

get_curs proc near

    mov AH, 03h
    mov BH, 0
    int 10h

    ret
get_curs endp

set_curs proc near

    mov AH, 02h
    mov BH, 0
    int 10h

    ret
set_curs endp
```



```

rout proc far
    jmp start
    signature dw 1234h
    number db 'Interrupt has been called 0000 times.$'
    keep_psp dw 0
    keep_cs dw 0
    keep_ip dw 0
    keep_ss dw 0
    keep_sp dw 0
    keep_ax dw 0
    IStack db 128 dup(?)

start:
    mov keep_ax, ax
    mov ax, ss
    mov keep_ss, ax
    mov keep_sp, sp
    mov ax, seg IStack
    mov ss, ax
    mov sp, offset start

    push cx
    push dx

    call GET_CURS
    push dx

    mov dh, 0
    mov dl, 0
    call SET_CURS
    push si

    push cx
    push ds
    push bp

    mov ax, seg number
    mov ds, ax
    mov si, offset number

```

```

        add si, 25
        mov cx, 4

loop_int:
        mov bp, cx
        mov ah, [si+bp]
        inc ah
        mov [si+bp], ah
        cmp ah, 3Ah
        jne print_msg
        mov ah, 30h
        mov [si+bp], ah
        loop loop_int

print_msg:
        pop bp
        pop ds
        pop cx
        pop si

        push es
        push bp

        mov ax, seg number
        mov es, ax
        mov ax, offset number
        mov bp, ax
        mov ah, 13h
        mov al, 0
        mov cx, 37
        mov bh, 0
        int 10h

        pop bp
        pop es

        pop dx
        call SET_CURS

```

```

        pop dx
        pop cx

        mov sp, keep_sp
        mov ax, keep_ss
        mov ss, ax
        mov ax, keep_ax
        mov al, 20h
        out 20h, al
        iret

route_end:
rout endp

load_rout proc near
    push dx
    push ax
    push cx

    mov ax, 351Ch
    int 21h
    mov keep_ip, bx
    mov keep_cs, es

    push ds
    mov dx, offset rout
    mov ax, seg rout
    mov ds, ax
    mov ax, 251Ch
    int 21h
    pop ds

    mov dx, offset route_end
    mov cl, 4
    shr dx, cl
    inc dx
    mov ax, cs
    sub ax, keep_psp
    add dx, ax

```

```

        xor ax, ax
        mov ah, 31h
        int 21h

        pop cx
        pop ax
        pop dx

        ret
load_rout endp

unload_rout proc near
        push ax
        push bx

        mov AH, 35h
        mov AL, 1Ch
        int 21h

        cli
        push ds
        mov ax, es:[keep_cs]
        mov ds, ax
        mov dx, es:[keep_ip]
        mov ah, 25h
        mov al, 1Ch
        int 21h
        pop ds
        sti

        mov ax, es:[keep_psp]
        mov es, ax
        push es
        mov ax, es:[2Ch]
        mov es, ax
        mov ah, 49h
        int 21h
        pop es
        int 21h

```

```

        pop bx
        pop ax
        ret
unload_rout endp

```

```

load_check proc near
    ; return value:
    ; al - nonzero if interrupt is set.
    push si
    push dx
    push bx
    push ax

```

```

    mov ax, 351Ch
    int 21h
    mov si, offset signature
    sub si, offset rout
    mov dx, es:[bx + si]
    mov al, 1
    cmp dx, 1234h
    je restore
    mov al, 0

```

```

restore:
    mov bl, al
    pop ax
    mov al, bl
    pop bx
    pop dx
    pop si
    ret

```

```

load_check endp

```

```

cmd_flag_check proc near
    ; return value:
    ; al - nonzero if cmd tail contains flag

```

```

    push bx

    mov al, 0
    mov bh, es:[82h]
    cmp bh, '/'
    jne end_
    mov bh, es:[83h]
    cmp bh, 'u'
    jne end_
    mov bh, es:[84h]
    cmp bh, 'n'
    jne end_
    mov al, 1

end_:
    pop bx
    ret

cmd_flag_check endp

print proc near
    push ax
    mov ah, 09
    int 21h
    pop ax
    ret
print endp

MAIN proc far
    mov ax, data
    mov ds, ax
    mov keep_psp, es

    call cmd_flag_check
    mov ah, al
    call load_check

```

```

; ah - is flag setted
; al - is interrupt loaded

cmp ah, 1
je flag_setted

flag_not_setted:
    cmp al, 1
    je print_already_loaded
    mov dx, offset loaded_msg
    call print
    call load_rout
    jmp finish_program

print_already_loaded:
    mov dx, offset already_loaded
    call print
    jmp finish_program

flag_setted:
    cmp al, 1
    jne print_not_loaded
    call unload_rout
    mov dx, offset unloaded_msg
    call print
    jmp finish_program

print_not_loaded:
    mov dx, offset not_loaded_msg
    call print

finish_program:
    xor ax, ax
    mov ah, 4Ch
    int 21h

main endp
code ends

```

```
end main
```

Файл info.asm

```
AStack    SEGMENT STACK
```

```
            DB 256 dup (?)
```

```
AStack    ENDS
```

```
DATA SEGMENT
```

```
    not_loaded_msg db 'Interrupt is not loaded.$', 0dh, 0ah
```

```
    loaded_msg db 'Interrupt is loaded.$', 0dh, 0ah
```

```
    unloaded_msg db 'Interrupt is unloaded.$', 0dh, 0ah
```

```
    already_loaded db 'Interrupt is already loaded.$', 0dh, 0ah
```

```
DATA ENDS
```

```
CODE    SEGMENT
```

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

```
get_curs proc near
```

```
    mov AH, 03h
```

```
    mov BH, 0
```

```
    int 10h
```

```
    ret
```

```
get_curs endp
```

```
set_curs proc near
```

```
    mov AH, 02h
```

```
    mov BH, 0
```

```
    int 10h
```

```
    ret
```

```
set_curs endp
```



```

rout proc far
    jmp start
    signature dw 1234h
    number db 'Interrupt has been called 0000 times.$'
    keep_psp dw 0
    keep_cs dw 0
    keep_ip dw 0
    keep_ss dw 0
    keep_sp dw 0
    keep_ax dw 0
    IStack db 128 dup(?)

start:
    mov keep_ax, ax
    mov ax, ss
    mov keep_ss, ax
    mov keep_sp, sp
    mov ax, seg IStack
    mov ss, ax
    mov sp, offset start

    push cx
    push dx

    call GET_CURS
    push dx

    mov dh, 0
    mov dl, 0
    call SET_CURS
    push si

    push cx
    push ds
    push bp

    mov ax, seg number
    mov ds, ax
    mov si, offset number

```

```

        add si, 25
        mov cx, 4

loop_int:
        mov bp, cx
        mov ah, [si+bp]
        inc ah
        mov [si+bp], ah
        cmp ah, 3Ah
        jne print_msg
        mov ah, 30h
        mov [si+bp], ah
        loop loop_int

print_msg:
        pop bp
        pop ds
        pop cx
        pop si

        push es
        push bp

        mov ax, seg number
        mov es, ax
        mov ax, offset number
        mov bp, ax
        mov ah, 13h
        mov al, 0
        mov cx, 37
        mov bh, 0
        int 10h

        pop bp
        pop es

        pop dx
        call SET_CURS

```

```

        pop dx
        pop cx

        mov sp, keep_sp
        mov ax, keep_ss
        mov ss, ax
        mov ax, keep_ax
        mov al, 20h
        out 20h, al
        iret

route_end:
rout endp

load_rout proc near
    push dx
    push ax
    push cx

    mov ax, 351Ch
    int 21h
    mov keep_ip, bx
    mov keep_cs, es

    push ds
    mov dx, offset rout
    mov ax, seg rout
    mov ds, ax
    mov ax, 251Ch
    int 21h
    pop ds

    mov dx, offset route_end
    mov cl, 4
    shr dx, cl
    inc dx
    mov ax, cs
    sub ax, keep_psp
    add dx, ax

```

```

        xor ax, ax
        mov ah, 31h
        int 21h

        pop cx
        pop ax
        pop dx

        ret
load_rout endp

unload_rout proc near
        push ax
        push bx

        mov AH, 35h
        mov AL, 1Ch
        int 21h

        cli
        push ds
        mov ax, es:[keep_cs]
        mov ds, ax
        mov dx, es:[keep_ip]
        mov ah, 25h
        mov al, 1Ch
        int 21h
        pop ds
        sti

        mov ax, es:[keep_psp]
        mov es, ax
        push es
        mov ax, es:[2Ch]
        mov es, ax
        mov ah, 49h
        int 21h
        pop es
        int 21h

```

```

        pop bx
        pop ax
        ret
unload_rout endp

```

```

load_check proc near
    ; return value:
    ; al - nonzero if interrupt is set.
    push si
    push dx
    push bx
    push ax

```

```

    mov ax, 351Ch
    int 21h
    mov si, offset signature
    sub si, offset rout
    mov dx, es:[bx + si]
    mov al, 1
    cmp dx, 1234h
    je restore
    mov al, 0

```

```

restore:
    mov bl, al
    pop ax
    mov al, bl
    pop bx
    pop dx
    pop si
    ret

```

```

load_check endp

```

```

cmd_flag_check proc near
    ; return value:
    ; al - nonzero if cmd tail contains flag

```

```

    push bx

    mov al, 0
    mov bh, es:[82h]
    cmp bh, '/'
    jne end_
    mov bh, es:[83h]
    cmp bh, 'u'
    jne end_
    mov bh, es:[84h]
    cmp bh, 'n'
    jne end_
    mov al, 1

end_:
    pop bx
    ret

cmd_flag_check endp

print proc near
    push ax
    mov ah, 09
    int 21h
    pop ax
    ret
print endp

MAIN proc far
    mov ax, data
    mov ds, ax
    mov keep_psp, es

    call cmd_flag_check
    mov ah, al
    call load_check

```

```

; ah - is flag setted
; al - is interrupt loaded

cmp ah, 1
je flag_setted

flag_not_setted:
    cmp al, 1
    je print_already_loaded
    mov dx, offset loaded_msg
    call print
    call load_rout
    jmp finish_program

print_already_loaded:
    mov dx, offset already_loaded
    call print
    jmp finish_program

flag_setted:
    cmp al, 1
    jne print_not_loaded
    call unload_rout
    mov dx, offset unloaded_msg
    call print
    jmp finish_program

print_not_loaded:
    mov dx, offset not_loaded_msg
    call print

finish_program:
    xor ax, ax
    mov ah, 4Ch
    int 21h

main endp
code ends

```

end main