

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. 9383

Преподаватель

Нистратов Д.Г.

Ефремов М.А.

Санкт-Петербург

2021

Постановка задачи.

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 управления 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 прерывания с определенным значением вектора. Таким образом, управление будет передано функции, чья точка входа записана в соответствующий вектор прерывания.

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 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 прерывания 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.

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) Сохранять 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 прерывания `int 10h`, которое позволяет непосредственно выводить информацию на экран.

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.

Выполнение работы.

Шаг 1. Был написан модуль типа `.EXE`, осуществляющий установку резидента, а также выгрузку.

Шаг 2. Запустим программу и убедимся, что резидентный обработчик прерывания `1Ch` установлен. Примеры работы программы см. Изображение 1

```

Interruption counter: 01593

For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com

Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>mount D: D:\
Drive D is mounted as local directory D:\

Z:\>D:

D:\>cd LETI\OS\MASM

D:\LETI\OS\MASM>LAB4.EXE
Interruption is loaded successfully

D:\LETI\OS\MASM>_

```

Изображение 1

Шаг 3. Запустим отлаженную программу еще раз и убедитесь, что программа определяет установленный обработчик прерываний. Примеры работы программы см. Изображение 2

```

Interruption counter: 1088 H5 T6

Z:\>mount D: D:\
Drive D is mounted as local directory D:\

Z:\>D:

D:\>cd LETI\OS\MASM

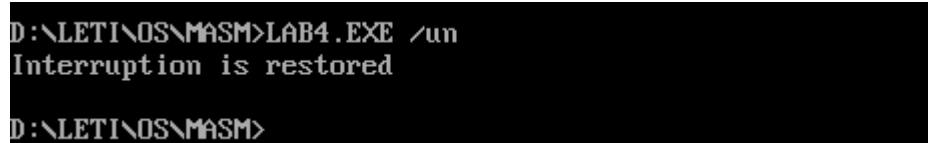
D:\LETI\OS\MASM>LAB4.EXE
Interruption is loaded successfully

D:\LETI\OS\MASM>lab3_1.com
Available memory: 644352
Extended memory: 15360
Table:
    4D      0008      16
    4D      0000      64
    4D      0040     256
    4D      0192     144
    4D      0192    4384   LAB4
    4D      02AF     4144
    5A      02AF   644352   LAB3_1

```

Изображение 2

Шаг 4. Запустим отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен. Примеры работы программы см. Изображение 3



```
D:\LETI\OS\MASM>LAB4.EXE /un
Interruption is restored
D:\LETI\OS\MASM>
```

Изображение 3

Ответы на вопросы:

1. Как реализован механизм прерывания от часов?

Каждые 55мс сохраняются содержимое регистров, после определяется смещение по номеру источника прерывания в таблице векторов. Первые 2 байта записываются в IP, вторые – в CS. Далее выполняется прерывание по сохранённому адресу и восстанавливается информация прерванного процесса, а также управление возвращается прерванной программе.

2. Какого типа прерывания использовались в работе?

1CH, 10H, 21H, где 1CH – аппаратные

10H, 21H – программные

Заключение.

В ходе лабораторной работы был написан обработчик прерываний таймера, а также была реализована загрузка и выгрузка резидента.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab4.asm

ASTACK SEGMENT STACK

 DW 200 DUP(?)

ASTACK ENDS

DATA SEGMENT

 ROUT_LOADED db 'Interruption is already loaded', 0DH, 0AH, '\$'

 ROUT_CHANGED db 'Interruption is loaded successfully', 0DH, 0AH, '\$'

 ROUT_IS_NOT_LOADED db 'Interruption is not loaded', 0DH, 0AH, '\$'

 ROUT_UNLOADED db 'Interruption is restored', 0DH, 0AH, '\$'

DATA ENDS

CODE SEGMENT

 ASSUME CS:CODE, DS:DATA, SS:ASTACK

WRITE PROC NEAR

 push ax

 mov ah, 9

 int 21h

 pop ax

 ret

WRITE ENDP

;функция вывода символа из AL

outputAL PROC

 push AX

 push BX

 push CX

```
mov AH, 09h
mov BH, 0
mov CX, 1
int 10h
pop CX
pop BX
pop AX
ret
outputAL ENDP
```

; Функция вывода строки по адресу ES:BP на экран

```
outputBP PROC NEAR
```

```
    push ax
    push bx
    push dx
    push cx
    mov ah,13h
    mov al, 0
    mov bh, 0
    mov dh, 22
    mov dl, 0
    int 10h
    pop cx
    pop dx
    pop bx
    pop ax
    ret
outputBP ENDP
```

; Установка позиции курсора

; установка на строку 25 делает курсор невидимым

setCurs PROC NEAR

mov ah, 02h

mov bh, 0h

mov dh, 0h

mov dl, 0h

int 10h

ret

setCurs ENDP

;

; 03H читать позицию и размер курсора

; вход: BH = видео страница

; выход: DH, DL = текущая строка, колонка курсора

; CH, CL = текущие начальная, конечная строки курсора

getCurs PROC NEAR

mov ah, 03h

mov bh, 0

int 10h

ret

getCurs ENDP

ROUT PROC FAR

jmp START

INT_COUNTER db 'Interruption counter: 0000\$'

INT_SIG dw 7777h

KEEP_IP dw 0

KEEP_CS dw 0

KEEP_PSP dw ?


```
KEEP_SS dw 0
KEEP_SP dw 0
KEEP_AX dw 0
INTSEG dw 16 dup(?)
```

START:

```
mov KEEP_SP, sp
mov KEEP_AX, ax
mov ax, ss
mov KEEP_SS, ax
```

```
mov ax, KEEP_AX
```

```
mov sp, OFFSET START
mov ax, seg INTSEG
mov ss, ax
```

; Сохранение изменяемого регистра

```
push ax
push cx
push dx
```

```
call getCurs
push dx
call setCurs
```

```
push si
push cx
push ds
push bp
```

```
mov ax, SEG INT_COUNTER
mov ds, ax
mov si, offset INT_COUNTER
add si, 21
mov cx, 4
```

LOOP_ELEM:

```
mov bp, cx
mov ah, [si+bp]
inc ah
mov [si+bp], ah
cmp ah, 3ah
jne UPDATE_RES
mov ah, 30h
mov [si+bp], ah

loop LOOP_ELEM
```

UPDATE_RES:

```
pop bp
pop ds
pop cx
pop si

push es
push bp

mov ax, SEG INT_COUNTER
mov es, ax
```

mov ax, offset INT_COUNTER

mov bp,ax

mov ah, 13h

mov al, 0

mov bh, 0

mov cx, 26

int 10h

pop bp

pop es

pop dx

mov ah, 2

mov bh, 0

int 10h

; Восстановление регистра

pop dx

pop cx

pop ax

mov KEEP_AX, ax

mov sp, KEEP_SP

mov ax, KEEP_SS

mov ss, ax

mov ax, KEEP_AX

mov al, 20h

out 20h, al

iret

END_ROUT_P:

ROUT ENDP

CHECK_UNLOAD PROC near

push ax

push es

mov cl, 0h

mov al,es:[81h+1]

cmp al,'/'

jne WRONG_ARG

mov al,es:[81h+2]

cmp al,'u'

jne WRONG_ARG

mov al,es:[81h+3]

cmp al,'n'

jne WRONG_ARG

mov cl,1h

WRONG_ARG:

pop es

pop ax

ret

CHECK_UNLOAD ENDP

CHECK_LOADED PROC NEAR

push ax

push dx

push es

push si

mov cl, 0h

mov ah, 35h

mov al, 1ch

int 21h

mov si, offset INT_SIG

sub si, offset ROUT

mov dx, es:[bx + si]

cmp dx, INT_SIG

jne NOT_LOADED

mov cl, 1h

NOT_LOADED:

pop si

pop es

pop dx

pop ax

ret

CHECK_LOADED ENDP

LOAD_ROUT PROC near

push ax

push cx

push dx

mov KEEP_PSP, es

mov ah, 35h

mov al, 1ch

int 21h

mov KEEP_CS, es

mov KEEP_IP, bx

push es

push bx

push ds

lea dx, ROUT

mov ax, SEG ROUT

mov ds, ax

mov ah, 25h

mov al, 1ch

int 21h

pop ds

pop bx

pop es

```
mov dx, offset ROUT_CHANGED  
call WRITE
```

```
lea dx, END_ROUT_P  
mov cl, 4h  
shr dx, cl  
inc dx
```

```
add dx, 100h
```

```
xor ax,ax
```

```
mov ah, 31h  
int 21h
```

```
pop dx  
pop cx  
pop ax  
ret
```

```
LOAD_ROUT ENDP
```

```
UNLOAD_ROUT PROC near
```

```
push ax  
push si
```

```
call CHECK_LOADED  
cmp cl, 1h  
jne ROUT_ISNOT_LOADED
```

```
cli
```

push ds

push es

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]

mov ds, ax

mov ah, 25h

mov al, 1ch

int 21h

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 es

pop ds

sti

mov dx, offset ROUT_UNLOADED

call WRITE

jmp UNLOAD_END

ROUT_ISNOT_LOADED:

mov dx, offset ROUT_IS_NOT_LOADED

call WRITE

UNLOAD_END:

pop si

pop ax

ret

UNLOAD_ROUT ENDP

MAIN PROC FAR

mov ax, DATA

mov ds, ax

call CHECK_UNLOAD

cmp cl, 1h

je START_UNLOAD

```
call CHECK_LOADED
cmp ch, 1h
je ALREADY_LOADED
```

```
call LOAD_ROUT
jmp EXIT
```

```
START_UNLOAD:
call UNLOAD_ROUT
jmp EXIT
```

```
ALREADY_LOADED:
mov dx, offset ROUT_LOADED
call WRITE
jmp EXIT
```

```
EXIT:
xor al, al
mov ah, 4ch
int 21h
```

```
MAIN ENDP
```

```
CODE ENDS
```

```
END MAIN
```