

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ по лабораторной
работе №4
по дисциплине «Операционные системы»
Тема: Обработка стандартных прерываний

Студент гр. 7381

Аженилок В.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2019

Цель работы.

В архитектуре компьютера существуют стандартные прерывания, за которыми закреплены определённые вектора прерываний. Вектор прерываний хранит адрес подпрограммы обработчика прерываний. При возникновении прерывания, аппаратура компьютера передаёт 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 определённые интервалы времени и, при возникновении такого сигнала, возникает прерывание с определённым значением вектора. Таким образом, управление будет передано функции, чья точка входа записана в соответствующий вектор прерывания.

Основные теоретические положения.

Резидентные обработчики прерываний - это программные модули, которые вызываются при возникновении прерываний определенного типа (сигнал таймера, нажатие клавиши и т.д.), которым соответствуют определенные вектора прерывания. Когда вызывается прерывание, 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. Обработчик прерывания должен заканчиваться инструкцией IRET (возврат из прерывания).

Вектор прерывания имеет длину 4 байта. В первом хранится значение IP, во втором - CS. Младшие 1024 байта памяти содержат 256 векторов. Вектор для прерывания 0 начинается с ячейки 0000:0000, для прерывания 1 - с ячейки 0000:0004 и т.д.

Обработчик прерывания - это отдельная процедура, имеющая следующую структуру:

ROUT PROC FAR

PUSH AX ; сохранение изменяемых регистров

.....

<действия по обработке прерывания>

POP AX ; восстановление регистров

MOV AL, 20H

OUT 20H,AL IRET

ROUT ENDP

Две последние строки необходимы для разрешения обработки прерываний с более низкими уровнями, чем только что обработанное. Для установки написанного прерывания в поле векторов прерываний используется функция 25H прерывания 21H, которая устанавливает вектор прерывания на указанный адрес.

PUSH DS

MOV DX, OFFSET ROUT ; смещение для процедуры в DX

MOV AX, SEG ROUT ; сегмент процедуры

MOV DS, AX ; помещаем в DS

MOV AH, 25H ; функция установки вектора

MOV AL, 1CH ; номер вектора

INT 21H ; меняем прерывание

POP DS

Программа, выгружающая обработчик прерываний должна восстанавливать оригинальные векторы прерываний. Функция 35 прерывания 21H позволяет восстановить значение вектора прерывания, помещая значение сегмента в ES, а смещение в BX. Программа должна содержать следующие инструкции:

; -- хранится в обработчике прерываний

KEEP_CS DW 0 ; для хранения сегмента

KEEP_IP DW 0 ; и смещения прерывания

; -- в программе при загрузке обработчика прерывания

```

MOV AH, 35H      ; функция получения вектора
MOV AL, 1CH      ; номер вектора ШТЕ 21P
MOV KEEP_IP, BX  ; запоминание смещения
MOV KEEP_CS, ES  ; и сегмента
; -- в программе при выгрузке обработчика прерываний CLI
PUSH DS
MOV DX, KEEP_IP
MOV AX, KEEP_CS
MOV DS, AX
MOV AH, 25H
MOV AL, 1CH
INT 21H          ; восстанавливаем вектор
POP DS
STI

```

Для того, чтобы оставить процедуру прерывания резидентной в памяти, следует воспользоваться функцией DOS 31h прерывания 21h. Эта функция оставляет память, размер которой указывается в качестве параметра, занятой, а остальную память освобождает и осуществляет выход в DOS. Функция 31h int 21h использует следующие параметры:

AH - номер функции 31h;
 AL - код завершения программы;
 DX - размер памяти в параграфах, требуемый резидентной программе.

Пример обращения к функции:

```

MOV DX, OFFSET LAST_BYTE ; размер в байтах от начала сегмента
MOV CL, 4 ; перевод в параграфы SHR
DX, CL
INC DX      ; размер в параграфах
MOV AH, 31h INT
21h

```

Описание функций и структур данных.

Таблица 1 – функции управляющей программы.

Название функции	Назначение
BYTE_TO_HEX	Переводит число AL в коды символов 16 с/с, записывая получившиеся в AL и AH.
TETR_TO_HEX	Вспомогательная функция для работы
	BYTE_TO_HEX
WRD_TO_HEX	Переводит число AX в строку в 16 с/с, записывая получившиеся в di, начиная с младшего разряда.
PRINT	Печатает строку на экран
outputBP	Функция вывода строки по адресу ES:BP
CHECK_ROUT	Функция, проверяющая установлен ли пользовательский обработчик прерываний.
SET_ROUT	Функция, устанавливающая пользовательской прерывание.
DELETE_ROUT	Функция, удаляющая пользовательское прерывание.
MAIN	Основная функция программы.
ROUT	Пользовательский обработчик прерываний, который считает и печатает на экран количество вызовов.

Таблица 2 – структуры данных управляющей программы.

Название	Тип	Назначение
LoadResident	db	Вывод строки ' Resident was loaded!'

UnloadResident	db	Вывод строки 'Resident was unloaded!'
AlreadyLoaded	db	Вывод строки 'Resident is already loaded!'
NotYetLoad	db	Вывод строки 'Resident not yet loaded!'

Описание работы утилиты.

Программа проверяет, установлено ли пользовательское прерывание с вектором 1Ch. Устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено. Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход через функцию 2Ch прерывания 21h. Выгружает прерывание по соответствующему значению параметра в командной строке /un. Выгрузка прерывания состоит в восстановлении стандартного вектора прерываний и освобождения памяти, занимаемой резидентом. Осуществляется выход через функцию 4Ch прерывания 21h.

Результат работы программы представлен на рис. 1.

```

Assembling file:  lab4.ASM
Error messages:  None
Warning messages: None
Passes:         1
Remaining memory: 466k

Turbo Link  Version 5.1 Copyright (c) 1992 Borland International
End of processing!
C:\>LAB4.EXE
Resident was loaded!
                                Number of calls: 004E
C:\>

```

Рисунок 1 – результат работы программы lab4.exe.

Для проверки размещения прерывания в памяти была запущена программа из лабораторной работы №3, отображающей карту памяти в виде блоков MSB

(рис. 2).

```
C:\>lab3
Available memory: 648080 B
Extended memory: 15360 KB
MCB Address | MCB Type | Owner | Size | Name
016F        | 4D        | 0008   | 16   |
0171        | 4D        | 0000   | 64   | DPMILOAD
0176        | 4D        | 0040   | 256  |
0187        | 4D        | 0192   | 144  |
0191        | 4D        | 0192   | 672  | LAB4
01BC        | 4D        | 01C7   | 144  |
01C6        | 5A        | 01C7   | 648064 | LAB3
                                     Number of calls: 0237
C:\>!!
```

Рисунок 2 – состояние памяти после загрузки собственного прерывания.

После повторного запуска программы было выведено сообщение о том, что резидентная программа уже загружена. Результат повторного запуска работы представлен на рис. 3.

```
C:\>LAB4.EXE
Resident is already loaded!
                                     Number of calls: 0477
C:\>_
```

Рисунок 3 – повторный запуск программы lab4.exe.

Была запущена программа с ключом выгрузки. Для того чтобы проверить, что память, занятая резидентом, освобождена, был выполнен запуск программы лабораторной работы №3.

```
C:\>LAB4.EXE /un
Resident was unloaded!
C:\>_
```

Рисунок 4 – Результат запуска программы с ключом выгрузки.

```
C:\>lab3
Available memory: 648928 B
Extended memory: 15360 KB
MCB Address | MCB Type | Owner | Size | Name
016F        | 4D        | 0008   | 16   |
0171        | 4D        | 0000   | 64   | DPMILOAD
0176        | 4D        | 0040   | 256  |
0187        | 4D        | 0192   | 144  |
0191        | 5A        | 0192   | 648912 | LAB3
C:\>_
```

Рисунок 5 – Состояние памяти после выгрузки резидентной программы.

Выводы.

В ходе выполнения лабораторной работы был изучен и построен обработчик прерываний сигналов таймера.

Ответы на контрольные вопросы.

1. Как реализован механизм прерывания от часов?

Ответ: Прерывание по таймеру вызывается каждые 55 мс – 18 раз в секунду. После вызова сохраняется содержимое регистров и определяется источник прерывания, по номеру которого определяется смещение в таблице векторов прерываний. Полученный адрес сохраняется в регистр CS:IP. После этого управление передаётся по этому адресу, т. е. выполняется запуск обработчика прерываний и происходит его выполнение. После выполнения происходит возврат управления прерванной программе.

2. Какого типа прерывания использовались в работе?

Ответ: В данной лабораторной работе использовались аппаратные прерывания (1Ch), прерывания функций MS DOS (int 21h) и прерывания функций BIOS (int 10h).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

`_CODE SEGMENT`

`ASSUME CS:_CODE, DS:_DATA, ES:NOTHING, SS:_STACK`

`ROUT PROC FAR`

`jmp start`

`SIGNATURE dw 01984h`

`KEEP_PSP dw 0`

`KEEP_IP dw 0`

`KEEP_CS dw 0`

`INT_STACK dw 100 dup (?)`

`COUNT dw 0`

`KEEP_SS dw 0`

`KEEP_AX dw ?`

`KEEP_SP dw 0`

`MESSAGE db 'Number of calls: $'`

`start:`

`mov KEEP_SS, SS mov`
`KEEP_SP, SP mov KEEP_AX,`
`AX mov AX, seg INT_STACK`
`mov SS, AX mov SP, 0 mov`
`AX, KEEP_AX`

`push ax`

`push bp`

`push es`

`push ds`

`push dx push di`

```

        mov ax, cs    mov ds, ax    mov
es, ax    mov ax, CS:COUNT    add
ax, 1    mov CS:COUNT, ax    mov di,
offset MESSAGE + 20    call
WRD_TO_HEX    mov bp, offset MESSAGE
call outputBP

```

```

        pop di    pop
dx    pop ds
        pop es    pop
bp    pop ax
        mov al, 20h
        out 20h, al

```

```

        mov AX, KEEP_SS
        mov SS, AX    mov
AX, KEEP_AX    mov
SP, KEEP_SP

```

```

        iret
ROUT ENDP

```

```

TETR_TO_HEX PROC near
    and al, 0fh    cmp
    al, 09 jbe     NEXT
        add al, 07 NEXT: add
    al, 30h
        ret
TETR_TO_HEX ENDP

```

```

BYTE_TO_HEX PROC near

```

```

push cx    mov  ah,al
call TETR_TO_HEX
xchg al,ah
mov  cl,4    shr
al,cl

        call TETR_TO_HEX
        pop    cx
        ret
BYTE_TO_HEX ENDP

```

```

WRD_TO_HEX PROC near

```

```

        push bx    mov  bh,ah
        call BYTE_TO_HEX
        mov  [di],ah    dec  di
        mov  [di],al    dec
di    mov  al,bh
        xor  ah,ah    call
BYTE_TO_HEX mov  [di],ah
        dec  di    mov  [di],al
        pop  bx

```

```

        ret
WRD_TO_HEX    ENDP
outputBP PROC near
    push ax    push bx
    push dx    push cx
    mov ah, 13h    mov
al, 0    mov bl, 03h
    mov bh, 0    mov dh,
23    mov dl, 22    mov
cx, 21    int 10h
    pop cx    pop dx
    pop bx    pop ax
    ret
outputBP ENDP
END_ROUT:

PRINT PROC near
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
PRINT ENDP

CHECK_ROUT PROC ah,
        35h

        mov al, 1ch int 21h mov
si, offset SIGNATURE sub si,
offset ROUT    mov ax, 01984h
    cmp ax, ES:[BX+SI]    je
ROUT_IS_LOADED    call SET_ROUT
ROUT_IS_LOADED:    call
DELETE_ROUT

        mov

```

```

        ret
CHECK_ROUT ENDP

SET_ROUT PROC
        mov ax, KEEP_PSP      mov
es, ax      cmp byte ptr
es:[80h], 0
        je      LOAD
        cmp byte ptr es:[82h], '/'
        jne LOAD
        cmp byte ptr es:[83h], 'u'
        jne      LOAD      cmp byte ptr
es:[84h], 'n'      jne LOAD

        lea dx, NotYetLoad      call
PRINT
        jmp      EXIT

LOAD:
        mov ah, 35h mov al,
        1ch int 21h mov
        KEEP_CS,      ES
        KEEP_IP, BX
        lea      dx, LoadResident
        call PRINT ;interrupt
        vector loading push ds
        mov dx, offset ROUT
        mov ax, seg ROUT
        mov ds, ax      mov
ah, 25h      mov al, 1ch

```

```

mov

```

```

        int 21h      pop
ds
    ;memory allocation    mov dx,
offset END_ROUT    mov cl, 4    shr
dx, cl    inc dx    add dx,
_CODE    sub dx, KEEP_PSP    sub
al, al    mov ah, 31h    int 21h
EXIT:
    sub al, al
    mov ah, 4ch
    int 21h
SET_ROUT ENDP

DELETE_ROUT PROC
    push dx
    push ax push
ds push es
    ax, KEEP_PSP

```

mov

```

        mov es, ax  cmp  byte
        ptr es:[80h], 0 je
            END_DELETE
        cmp  byte ptr es:[82h], '/'
jne  END_DELETE  cmp  byte ptr
es:[83h], 'u'  jne  END_DELETE  cmp
byte ptr es:[84h], 'n'  jne
END_DELETE

        lea dx, UnloadResident  call PRINT

        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-2]  mov
es, ax  mov ax, es:[2ch]
push es
        mov es, ax
        mov ah, 49h
        int 21h  pop
es

```

```

        mov  ah, 49h int
        21h

        jmp  END_DELETE2

        END_DELETE:
        mov  dx, offset AlreadyLoaded    call
PRINT
        END_DELETE2:

        pop  es
        pop          ds
        pop  ax    pop
dx      ret
DELETE_ROUT ENDP

MAIN  PROC NEAR  mov  ax,
_DATA  mov  ds, ax  mov
KEEP_PSP,  es      call
CHECK_ROUT      mov  ax,
4C00h  int  21h  ret
MAIN  ENDP
_CODE  ENDS
_STACK SEGMENT STACK
        db 512 dup(0)
_STACK ENDS
_DATA SEGMENT
        LoadResident      db      'Resident was loaded!', 0dh,
0ah, '$'
        UnloadResident    db      'Resident was unloaded!', 0dh,
0ah, '$'
        AlreadyLoaded      db      'Resident is already loaded!',
0dh, 0ah, '$'

```



```
                NotYetLoad                db        'Resident    not    yet
loaded!', 0DH, 0AH, '$'
_DATA    ENDS
                END    MAIN
```