

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

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

Студент гр. 8381

Муковский Д.В.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

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 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 определённые 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
    INT 21H
    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 прерывания 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

Выполнение работы.

Написан текст исходного EXE модуля, который выполняет следующие функции: проверяет, установлено ли пользовательское прерывание с вектором 1Ch; устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено, и осуществляется выход о функции 4Ch прерывания int 21h; если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h; 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.

Полученный исходный модуль был отлажен. Результаты выполнения программы представлены на рис. 1.



```
S:\>lr4.exe
Resident program has been loaded          Number of interrupts: 069
```

Рисунок 1 – Результат выполнения LR4.EXE

После чего было проверено размещение прерывания в памяти. Для этого была запущена программа LR3_1.COM, которая отображает карту памяти в виде списка блоков MCB. Результат выполнения программы представлен на рис. 2.

```
S:\>lr3.com
Available memory: 640 kbytes
Expanded memory: 15360 kbytes
MCB: 1
Block is MSDOS      Area size: 16

MCB: 2
Block is free       Area size: 64

MCB: 3
Block is 0040       Area size: 256

MCB: 4
Block is 0192       Area size: 144

MCB: 5
Block is 0192       Area size: 4464
LR4
MCB: 6
Block is 02B4       Area size: 144

MCB: 7
Block is 02B4       Area size: 644272
LR3
```

Рисунок 2 – Состояние памяти после загрузки прерывания

На рисунке видно, что процедура прерывания осталась резидентной в памяти и располагается в блоке 5.

После повторного запуска программа определила установленный обработчик прерываний. Результат выполнения программы представлен на рис. 3.

```
S:\>lr4.exe
Resident program has been loaded      Number of interrupts: 861

S:\>lr4.exe
Resident program is already loaded    Number of interrupts: 910
```

Рисунок 3 – Повторный запуск программы

Далее программа была запущена с ключом выгрузки, чтобы убедиться, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также была запущена программа LR3_1.COM. Результаты выполнения программы представлен на рис. 4.

```

S:\>lr4.exe /un
Resident program has been unloaded

S:\>lr3_1.com
Available memory: 640 kbytes
Expanded memory: 15360 kbytes
MCB number 1
Block is MSDOS   Area size: 16

MCB number 2
Block is free    Area size: 64

MCB number 3
Block is 0040    Area size: 256

MCB number 4
Block is 0192    Area size: 144

MCB number 5
Block is 0192    Area size: 648912
LR3_1

```

Рисунок 4 – Состояние памяти после выгрузки прерывания

Из рисунка видно, что память резидентного обработчика была освобождена.

Контрольные вопросы

■ Как реализован механизм прерывания от часов?

1. Системный таймер вырабатывает прерывание INT 8h 1193180/65536 раз в секунду.
2. При инициализации BIOS устанавливает свой обработчик для прерывания таймера. Этот обработчик каждый раз увеличивает на 1 текущее значение счетчика таймера.
3. Последнее действие, которое выполняет обработчик прерывания таймера - вызов прерывания INT 1Ch. После инициализации системы вектор INT 1Ch указывает на команду IRET, т.е. ничего не выполняется. Программа может установить собственный обработчик этого прерывания, для того чтобы выполнять какие-либо периодические действия.
4. Прерывание INT 1Ch вызывается обработчиком прерывания INT 8h до сброса контроллера прерывания, поэтому во время выполнения прерывания INT 1Ch все аппаратные прерывания запрещены

5. Обработчик прерывания INT 1Ch должен заканчиваться командой IRET, которая приводит к возврату в основную программу в ту самую точку, где она была прервана.

▪ **Какого типа прерывания использовались в работе?**

- Аппаратные прерывания (INT 8H)
- Прерывания функций BIOS для видеосервиса (INT 10H)
- Прерывания функций DOS (INT 21H)

Вывод.

В ходе выполнения лабораторной работы был реализован обработчик прерывания от сигналов таймера. Изучены дополнительные функции работы с памятью: установка программы-резидента и его выгрузка из памяти.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. OS4.ASM

```
ASTACK      SEGMENT  STACK
```

```
DW 128 DUP(0)
```

```
ASTACK  ENDS
```

```
DATA        SEGMENT
```

```
INT_LOADED_NOW      db 'Resident program has been loaded', 0dh, 0ah,  
'$'
```

```
INT_UNLOAD          db 'Resident program has been unloaded', 0dh, 0ah,  
'$'
```

```
INT_ALREADY_LOADdb 'Resident program is already loaded', 0dh, 0ah, '$'
```

```
NOT_LOADED          db 'Resident program is not loaded', 0dh, 0ah,  
'$'
```

```
IS_LOADED            DB 0
```

```
CHECK                DB 0
```

```
DATA          ENDS
```

```
CODE  SEGMENT
```

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

```
INTERRUPT proc far
```

```
    jmp  INTERRUPT_START
```

```
        COUNTER_INT db 'Number of interrupts: 000'
```

```
        INT_ID      dw 6666h
```

```
        SAVE_IP     dw 0
```

```
        SAVE_CS     dw 0
```

```
        SAVE_PSP    dw 0
```

```
INTERRUPT_START:
```

```
    push ax
```

```
    push bx
```

```
    push cx
```

```
    push dx
```

```
    push si
```

```
    push es
```

```
    push ds
```

```
mov ax, seg COUNTER_INT
mov ds, ax
```

```
call GET_CURS ;dh, dl - текущая строка, колонка курсора
                ;ch, cl - текущая начальная, конечная строка
```

курсора

```
push dx
call SET_CURS
```

```
mov ax, seg COUNTER_INT
push ds
mov ds, ax
mov si, offset COUNTER_INT
add si, 24
mov cx, 3
```

```
CYCLE:
    mov ah, [si]
    inc ah
    mov [si], ah
    cmp ah, ':'
    jne END_CYCLE
    mov ah, '0'
    mov [si], ah
    dec si
    loop CYCLE
```

```
END_CYCLE:
    pop ds
```

```
PRINTING:
    push es
    push bp
    mov ax, seg COUNTER_INT
    mov es, ax
    mov bp, offset COUNTER_INT
    call OUTPUT_BP
    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 al, 20h
        out 20h, al

        iret
INTERRUPT ENDP

        LAST_BYTE:

OUTPUT_BP proc near ;вывод строки по адресу es:bp на экран
        push ax
        push bx
        push dx
        push cx
        mov ah, 13h ;функция вывода строки в bp
        mov al, 1h ;использовать атрибут в bl и не трогать курсор
        mov bl, 0Bh ;цвет
        mov bh, 0 ;номер видео страницы
        mov cx, 25 ;длина строки
        int 10h

        pop cx
        pop dx
        pop bx
        pop ax
        ret
OUTPUT_BP ENDP

```

GET_CURS proc

push ax

push bx

mov ah, 03h

mov bh, 0h

int 10h

pop bx

pop ax

ret

GET_CURS ENDP

SET_CURS proc

push ax

push bx

mov ah, 02h

mov bh, 0h

mov dh, 22 ;строка начала вывода

mov dl, 42 ;колонка начала вывода

int 10h

pop bx

pop ax

ret

SET_CURS ENDP

PRINT proc near

push ax

mov ah, 09h

int 21h

pop ax

ret

PRINT endp

LOAD_INTERRUPT PROC near

```

push AX
push BX
push CX
push DX
push DS
push ES

mov AH, 35H ; функция получения вектора
mov AL, 1CH ; номер вектора
int 21H
mov SAVE_IP, BX ; запоминание смещения
mov SAVE_CS, ES ; и сегмента

CLI
push DS
mov DX, offset INTERRUPT
mov AX, seg INTERRUPT
mov DS, AX
mov AH, 25H
mov AL, 1CH
int 21H ; восстанавливаем вектор
pop DS
STI

mov DX, offset LAST_BYTE
mov cl, 4h
shr dx, cl
add dx, 10fh
inc DX ; размер в параграфах
xor AX, AX
mov AH, 31h
int 21h

pop ES
pop DS
pop DX
pop CX
pop BX
pop AX

```

```

        ret
LOAD_INTERRUPT ENDP

UNLOAD_INTERRUPT PROC near
    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_IP
    sub SI, offset INTERRUPT
    mov DX, ES:[BX+SI]
    mov AX, ES:[BX+SI+2]

    push DS
    mov DS, AX
    mov AH, 25h
    mov AL, 1Ch
    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_INTERRUPT ENDP

CHECK_UN PROC near
    push AX
    push ES

    mov AX, SAVE_PSP
    mov ES, AX
    cmp byte ptr ES:[82h], '/'
    jne END_OF_CHECK
    cmp byte ptr ES:[83h], 'u'
    jne END_OF_CHECK
    cmp byte ptr ES:[84h], 'n'
    jne END_OF_CHECK
    mov CHECK , 1

    END_OF_CHECK:
        pop ES
        pop AX
        ret
CHECK_UN ENDP

CHECK_1CH PROC near
    push AX
    push BX
    push SI

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

```

```
mov SI, offset INT_ID
sub SI, offset INTERRUPT
mov AX, ES:[BX+SI]
cmp AX, 6666h
jne END_OF_CHECK_1CH
mov IS_LOADED, 1
```

```
END_OF_CHECK_1CH:
    pop SI
    pop BX
    pop AX
    ret
```

```
CHECK_1CH ENDP
```

```
MAIN PROC FAR
```

```
    push DS
    xor ax, ax
    push ax
    mov AX, DATA
    mov DS, AX
    mov SAVE_PSP, ES
```

```
    call CHECK_1CH
    call CHECK_UN
```

```
    cmp CHECK, 1
    je UNLOAD
```

```
    mov al, IS_LOADED
    cmp al, 1
    jne LOAD
```

```
    mov DX, offset INT_ALREADY_LOAD
    call PRINT
    jmp ENDD
```

```
LOAD:
```



```
    mov DX, offset INT_LOADED_NOW
    call PRINT
    call LOAD_INTERRUPT
    jmp ENDD
```

UNLOAD:

```
    cmp IS_LOADED, 1
    jne IF_1CH_NOT_SET
    call UNLOAD_INTERRUPT
    mov DX, offset INT_UNLOAD
    call PRINT
    jmp ENDD
```

IF_1CH_NOT_SET:

```
    mov DX, offset NOT_LOADED
    call PRINT
```

ENDD:

```
    xor AL, AL
    mov AH, 4Ch
    int 21h
```

MAIN ENDP

CODE ENDS

END MAIN