

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

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

Студент гр. 8381

Преподаватель

Почаев Н.А.

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Построить обработчик прерываний сигналов таймера. Эти сигналы генерируются аппаратурой через определённые интервалы времени и, при возникновении такого сигнала, возникает прерывание с определённым значением вектора.

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 прерывание - это отдельная процедура, имеющая следующую структуру:

```
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.

OFFSET	смещение для процедуры в
SEG	сегмент процедуры
AX	помещаем в DS
25H	функция установки вектора
1CH	номер вектора
	меняем прерывание

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 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
```

Выполнение работы.

Выполнение работы производилось на базе операционной системы Windows XP (32 bit), запускаемой в системе виртуализации VMware Workstation, в редакторе Notepad++. Сборка и отладка модулей производились с помощью компилятора MASM и отладчика AFD. Также в работе был использован консольный файловый менеджер Far Manager и HEX-редактор HxD. Для дополнительного тестирования и проверки функциональности программы использовался DOSBox.

1. Состояние памяти до загрузки резидента (используем модуль, разработанный в третьей лабораторной работе):

```
E:\Labs\lr4>LR3_1.COM
Availible memory: 633536 B
Extended memory : 1024 KB
Address ! MCB Type ! PSP Address ! Size
020A      4D      0008
0414      4D      0415      2896
04CA      4D      0000      2112
04D2      4D      0415      1088
0517      4D      0553      1928
0552      5A      0553      633536
```

2. Загрузка резидента в память.

```
C:\WINDOWS\system32\cmd.exe - MAIN.EXE
E:\Labs\lr4>MAIN.EXE
Interrupts call count: 0406
```

3. Попытка повторной загрузки резидента в память.

```
E:\Labs\lr4>MAIN.EXE
E:\LABS\LR4>MAIN.EXE
Interruption already load!
E:\LABS\LR4>
```

4. Состояние памяти при загрузке в неё резидента

```
E:\LABS\LR4>LR3_1.COM
Availible memory: 631760 B
Extended memory : 1024 KB
```

Address	MCB Type	PSP	Address	Size
020A	4D		0008	8336
0414	4D		0415	2896
04CA	4D		0000	2112
04D2	4D		0415	1088
0517	4D		0553	1928
0552	4D		0553	1864
0589	4D		05C2	1880
05C1	5A		05C2	631760

5. Запускаем отложенную программу с ключом /un, тем самым выгружаем резидент и смотрим состояние памяти после выгрузки резидента.

```
E:\LABS\LR4>MAIN.EXE /un  
Interruption was restored!
```

Выводы.

В ходе выполнения лабораторной лабораторной работы была реализована программа загружающая и выгружающая пользовательское прерывание от системного таймера в память.

Ответы на контрольные вопросы.

1. Как реализован механизм прерывания от часов?

Сначала сохраняется содержимое регистров, потом определяется источник прерывания, по номеру которого определяется смещение в таблице векторов прерывания, сохраняется в CS:IP, передаётся управление по адресу CS:IP и происходит выполнение обработчика, и в конце происходит возврат управления прерванной программе. Аппаратное прерывание от таймера (int 8h) происходит каждые 1193180/65536 раз в мс.

При инициализации BIOS устанавливает свой обработчик для прерывания таймера. Этот обработчик каждый раз увеличивает на 1 текущее значение счетчика тиков таймера.

В конце этот обработчик прерывания вызывает прерывание int 1Ch пользовательское прерывание по таймеру (по соответствующему адресу в таблице векторов прерываний). После инициализации системы вектор INT 1Ch указывает на команду IRET, однако в реализованной в данной работе программе вектор указывает на пользовательский обработчик, который выполняет вывод на экран счетчика вызовов прерывания системного таймера.

2. Какого типа прерывания использовались в работе?

- Аппаратные прерывания – асинхронное прерывание от таймера;
- Прерывания функций DOS(21h);
- Прерывания функций BIOS(10h).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ MAIN.ASM.

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

```
MY_STACK SEGMENT STACK
```

```
    DW 64 DUP(?)
```

```
MY_STACK ENDS
```

```
;-----
```

```
CODE SEGMENT
```

```
MY_INTERRUPTION PROC FAR
```

```
    jmp START_FUNC
```

```
    ; TMP DATA
```

```
    PSP_ADDRESS_0 dw 0 ; offset - 3
```

```
    PSP_ADDRESS_1 dw 0 ; offset - 5
```

```
    KEEP_CS dw 0 ; offset - 7
```

```
    KEEP_IP dw 0 ; offset - 9
```

```
    MY_INTERRUPTION_SET dw 0FEDCh ; offset - 11
```

```
    INT_COUNT db 'Interrupts call count: 0000 $' ; offset - 13
```

```
    INT_STACK DW 100 dup (?)
```

```
    KEEP_SS DW 0
```

```
    KEEP_AX DW ?
```

```
    KEEP_SP DW 0
```

```
START_FUNC:
```

```
    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 bx
```

```
    push cx
```

```
    push dx
```

```
    mov ah, 03h
```

```
    mov bh, 00h
```

```
    int 10h
```

```
    push dx
```



```

mov ah, 02h
mov bh, 00h
mov dx, 0220h
int 10h

push si
push cx
push ds
mov ax, SEG INT_COUNT
mov ds, ax
mov si, offset INT_COUNT
add si, 1Ah

mov ah,[si]
inc ah
mov [si], ah
cmp ah, 3Ah
jne END_CALC
mov ah, 30h
mov [si], ah

mov bh, [si - 1]
inc bh
mov [si - 1], bh
cmp bh, 3Ah
jne END_CALC
mov bh, 30h
mov [si - 1], bh

mov ch, [si - 2]
inc ch
mov [si - 2], ch
cmp ch, 3Ah
jne END_CALC
mov ch, 30h
mov [si - 2], ch

mov dh, [si - 3]
inc dh
mov [si - 3], dh
cmp dh, 3Ah
jne END_CALC
mov dh, 30h
mov [si - 3],dh

```

END_CALC:

```

pop ds
pop cx
pop si

push es
    push bp
        mov ax, SEG INT_COUNT
        mov es, ax
        mov ax, offset INT_COUNT
        mov bp, ax
        mov ah, 13h
        mov al, 00h
        mov cx, 1Dh
        mov bh, 0
        int 10h
    pop bp
pop es

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

pop dx
pop cx
pop bx
pop ax

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

    iret
MY_INTERRUPTION ENDP
;-----
NEED_MEM_AREA PROC
NEED_MEM_AREA ENDP

IS_INTERRUPTION_SET PROC NEAR
    push bx
    push dx
    push es

    mov ah, 35h
    mov al, 1Ch
    int 21h

```

```

        mov dx, es:[bx + 11]
        cmp dx, 0FEDCh
        je INT_IS_SET
        mov al, 00h
        jmp POP_REG

INT_IS_SET:
        mov al, 01h
        jmp POP_REG

POP_REG:
        pop es
        pop dx
        pop bx

        ret
IS_INTERRUPTION_SET ENDP
;-----
CHECK_COMMAND_PROMT PROC NEAR
        push es

        mov ax, PSP_ADDRESS_0
        mov es, ax

        mov bx, 0082h

        mov al, es:[bx]
        inc bx
        cmp al, '/'
        jne NULL_CMD

        mov al, es:[bx]
        inc bx
        cmp al, 'u'
        jne NULL_CMD

        mov al, es:[bx]
        inc bx
        cmp al, 'n'
        jne NULL_CMD

        mov al, 0001h
NULL_CMD:
        pop es

        ret

```

```

CHECK_COMMAND_PROMT ENDP
;-----
LOAD_INTERRUPTION PROC NEAR
    push ax
    push bx
    push dx
    push es

    mov ah, 35h
    mov al, 1Ch
    int 21h

    mov KEEP_IP, bx
    mov KEEP_CS, es

    push ds
        mov dx, offset MY_INTERRUPTION
        mov ax, seg MY_INTERRUPTION
        mov ds, ax

        mov ah, 25h
        mov al, 1Ch
        int 21h
    pop ds

    mov dx, offset M_INT_ISLOADED0
    call PRINT_STRING

    pop es
    pop dx
    pop bx
    pop ax

    ret
LOAD_INTERRUPTION ENDP
;-----
UNLOAD_INTERRUPTION PROC NEAR
    push ax
    push bx
    push dx
    push es

    mov ah, 35h
    mov al, 1Ch
    int 21h

    cli

```

```

    push ds
        mov dx, es:[bx + 9]
        mov ax, es:[bx + 7]

        mov ds, ax
        mov ah, 25h
        mov al, 1Ch
        int 21h
    pop ds
    sti

    mov dx, offset M_INT_RESTORED
    call PRINT_STRING

    push es
        mov cx, es:[bx + 3]
        mov es, cx
        mov ah, 49h
        int 21h
    pop es

    mov cx, es:[bx + 5]
    mov es, cx
    int 21h

    pop es
    pop dx
    pop bx
    pop ax

    ret
UNLOAD_INTERRUPTION ENDP
;-----
PRINT_STRING PROC NEAR
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
PRINT_STRING ENDP
;-----
MAIN_PROGRAM PROC FAR
    mov bx, 02Ch
    mov ax, [bx]
    mov PSP_ADDRESS_1, ax
    mov PSP_ADDRESS_0, ds
    sub ax, ax

```

```

    xor bx, bx

    mov ax, DATA
    mov ds, ax

    call CHECK_COMMAND_PROMT
    cmp al, 01h
    je UNLOAD_START

    call IS_INTERRUPTION_SET
    cmp al, 01h
    jne INTERRUPTION_IS_NOT_LOADED

    mov dx, offset M_INT_ISLOADED ; exit with message
    call PRINT_STRING
    jmp EXIT_PROGRAM

    mov ah, 4Ch
    int 21h

INTERRUPTION_IS_NOT_LOADED:
    call LOAD_INTERRUPTION

    mov dx, offset NEED_MEM_AREA
    mov cl, 04h
    shr dx, cl
    add dx, 1Bh

    mov ax, 3100h
    int 21h

UNLOAD_START:
    call IS_INTERRUPTION_SET
    cmp al, 00h
    je INT_IS_NOT_SET
    call UNLOAD_INTERRUPTION
    jmp EXIT_PROGRAM

INT_IS_NOT_SET:
    mov dx, offset M_INT_NOT_SET
    call PRINT_STRING
    jmp EXIT_PROGRAM

EXIT_PROGRAM:
    mov ah, 4Ch
    int 21h
MAIN_PROGRAM ENDP

```

CODE ENDS

;-----

DATA SEGMENT

 ;messages

 M_INT_NOT_SET db "Interruption didnt load!", 0dh, 0ah, '\$'

 M_INT_RESTORED db "Interruption was restored!", 0dh, 0ah, '\$'

 M_INT_ISLOADED db "Interruption already load!", 0dh, 0ah, '\$'

 M_INT_ISLOADED0 db "Interuption is loading now!", 0dh, 0ah, '\$'

DATA ENDS

END MAIN_PROGRAM