实验报告

院系：数据科学与计算机学院　　专业、年级：17 级计算机科学与技术
学号：17341097　　姓名：廖永滨　　指导教师：凌应标

【实验题目】

1、操作系统工作期间，利用时钟中断，在屏幕 24 行 79 列位置轮流显示’|’、’/’和’\’，适当控制显示速度，以方便观察效果。

2、编写键盘中断响应程序，原有的你设计的用户程序运行时，键盘事件会做出有事反应：当键盘有按键时，屏幕适当位置显示”OUCH! OUCH!”。

3、在内核中，对 33 号、34 号、35 号和 36 号中断编写中断服务程序，分别在屏幕 1/4 区域内显示一些个性化信息。再编写一个汇编语言的程序，作为用户程序，利用 int 33、int 34、int 35 和 int 36 产生中断调用你这 4 个服务程序。

【实验目的】

1. 学习中断机制知识，掌握中断处理程序设计的要求。

2. 设计时钟中断处理程序

3. 扩展 MyOS1，增加一个系统服务的实验。

4. 建立具有简单异步事件处理的原型操作系统。

【实验要求】

1. 实验四必须在实验三基础上进行，保留或扩展原有功能，实现部分新增功能。

【实验方案】

1. 实验工具：

Notepad++：编写程序时使用的编辑器；
Subline：可以以 16 进制的方式打开并编辑任意文件；
TAMS 汇编工具：可以将汇编代码编译成对应的二进制代码；
NAMS 汇编工具：可以将汇编代码编译成对应的二进制代码；
TCC 编译器：可以将 c 代码编译成对应的二进制代码；
TLINK 链接器：将多个.obj 文件链接成.com 文件
VmWare 虚拟机：创建裸机环境，生成虚拟磁盘

2. 实验基本操作方法：

a. 使用 TCC 编译命令：tcc -mt -c - o cfile.obj cfile.c >ccmsg.txt
b. TASM 汇编命令：tasm afile.asm afile.obj > amsg.txt
c. 链接命令：tlink /3 /t cfile.obj afile.obj , final.com
d. NASM 汇编指令：NASM afile.asm

3. 实验原理

中断机制基础知识，中断程序的编写规范。

【实验过程】

1、实现的内容：

该实验中有 6 个程序，分别为 loading，MyOS，a1，a2, a3, a4。

lording 作为系统引导程序，放在软盘第一个扇区。

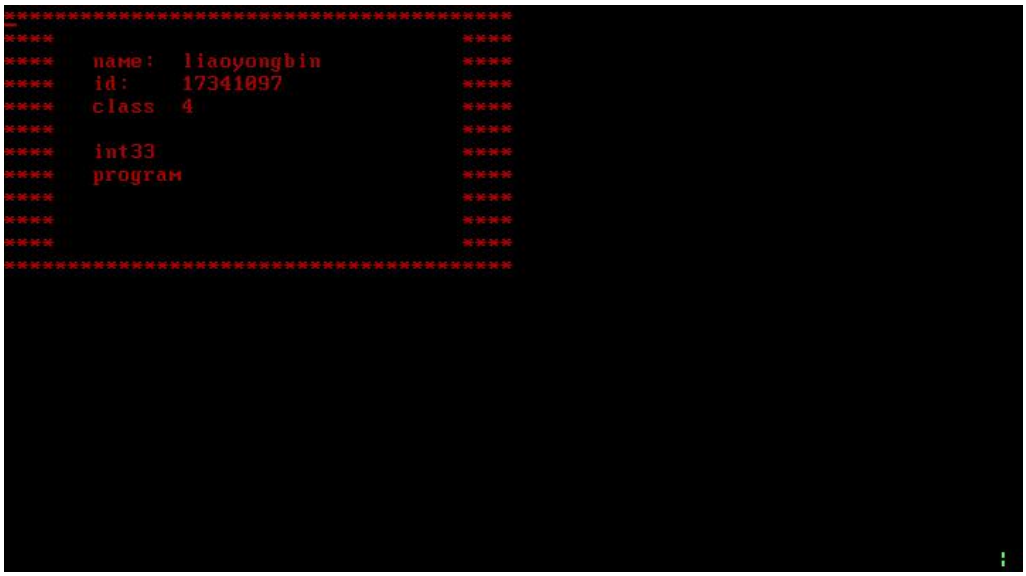
MyOS 为操作系统，由 TASM 汇编语言与 TURBO C 语言汇合编写，放在软盘第二个到第十个扇区（占据 9 个扇区）。

a1，a2, a3, a4 为用户程序，延用实验三的四个程序。分别放在软盘第十一、第十二、第十三和第十四个扇区，主要用于演示键盘中断。

运行时首先启动引导程序 lording,把系统 MyOS 装载到内存 8100h 中，并把计算机的控制权交给 MyOS 系统，系统立即将时钟中断向量的偏移地址设置为时钟中断程序的地址，系统开始定时执行时钟中断程序，然后设置 33h，34h,35h,36h 中断向量的偏移地址，作为系统服务准备接受用户调用，接着等待用户输入指令。

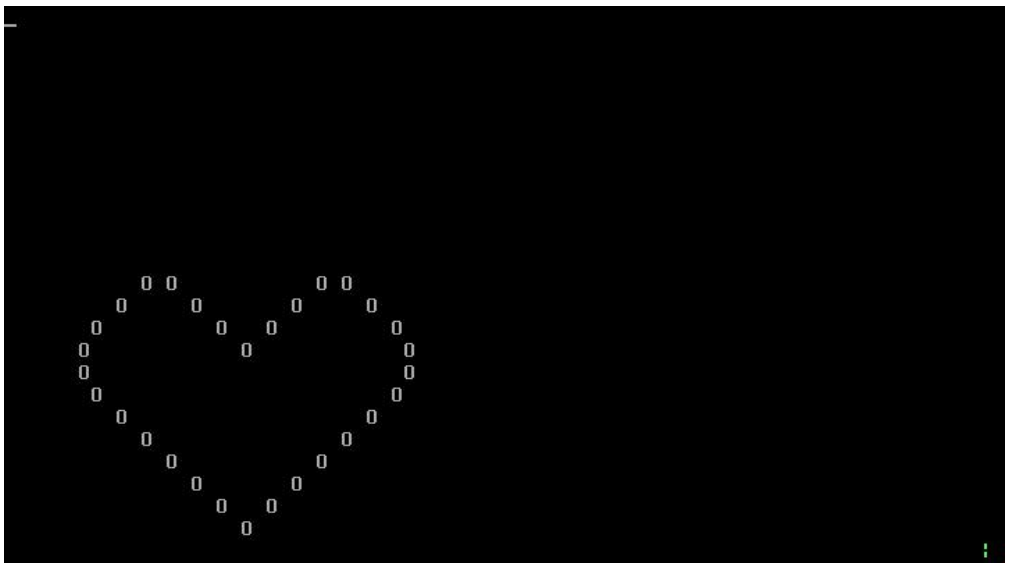
实验三已有的功能和程序不再累述。这里展示新出的四个服务程序和时间中断效果、键盘中断效果。如下：

时间中断 20h 开始执行，右下角风火轮正在转动（动图没法截。。）



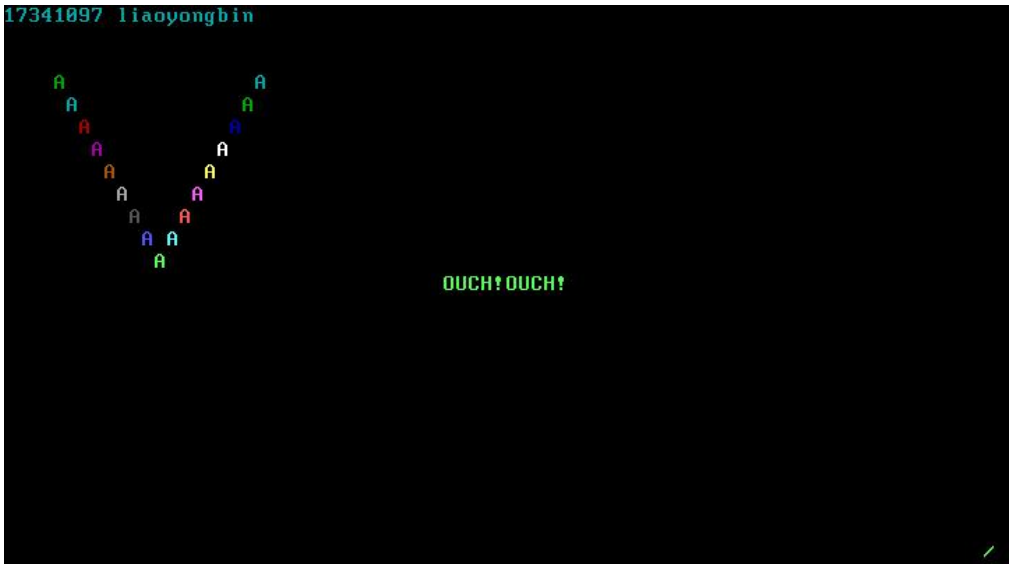
键入 int33 字符串后，中断 33h 的服务程序运行，风火轮依旧在转动

键入 int34 字符串后，中断 34h 的服务程序运行，风火轮依旧在转动



键入 int35 字符串后，中断 35h 的服务程序运行，风火轮依旧在转动

键入 int36 字符串后，中断 36h 的服务程序运行，风火轮依旧在转动



键入 r 1 字符串后，用户子程序 1 运行，风火轮依旧在转动，此时用户如果有键盘操作，屏幕就会出现 ouch 字符串！

2、中断设置的详细说明：

时钟中断：时钟中断向量入口地址为 ip：20h cs：22h，系统启动后立刻设置该时钟中断向量，中断程序在屏幕右下角轮流打印 \ 、|、/、- 。

键盘中断：键盘中断向量入口地址为 ip:24h cs:26h，每当用户程序装载之前，会先保存原来的 9 号键盘中断，然后设置新的键盘中断向量的偏移地址和段基址，在用户程序执行过程中，每次敲击键盘，都会在屏幕上某个位置打印出 “OUCH!OUCH!”的信息，在用户程序执行完后，系统将恢复原来的 9 号键盘中断。

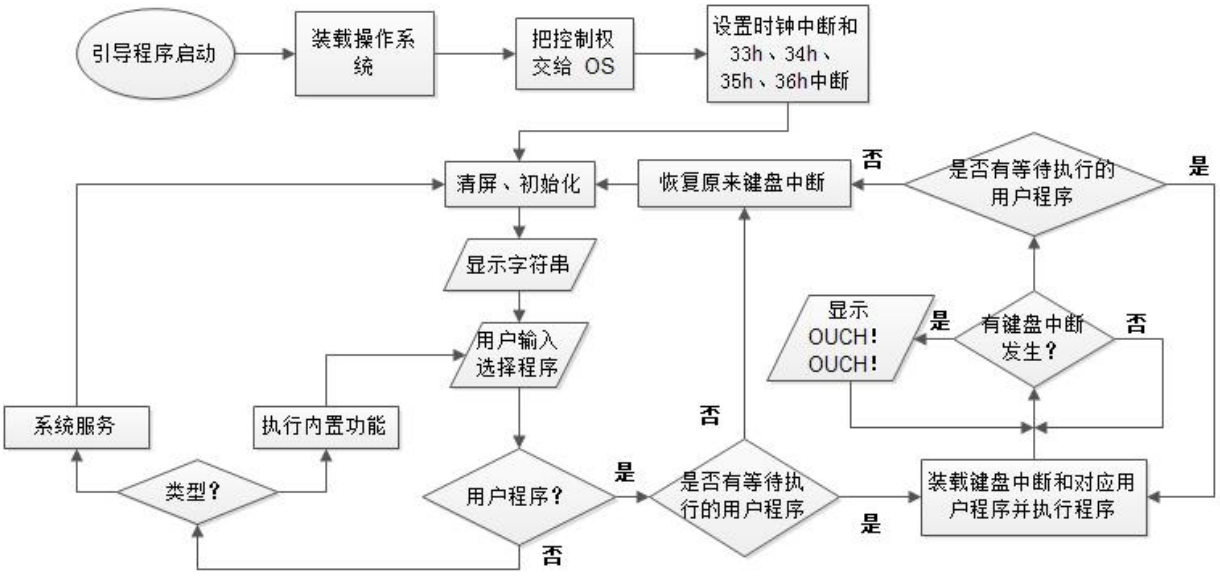
33h 中断：入口地址为 ip:51*4 cs:51*4+2 ，该中断在系统启动后立刻装载到对应中断向量，可以通过 int 33h 进行调用，调用时在左上角打印某些信息。

34h 中断：入口地址为 ip:52*4 cs:52*4+2 ，该中断在系统启动后立刻装载到对应中断向量，可以通过 int 34h 进行调用，调用时在右上角打印某些信息。

35h 中断：入口地址为 ip:53*4 cs:53*4+2 ，该中断在系统启动后立刻装载到对应中断向量，可以通过 int 35h 进行调用，调用时在左下角打印某些信息。

36h 中断：入口地址为 ip:54*4 cs:54*4+2 ，该中断在系统启动后立刻装载到对应中断向量，可以通过 int 36h 进行调用，调用时在右下角打印某些信息。

3、程序流程图



4、代码解析（实现方法）

loading.asm：

由于实验三中我用自己定义的新中断调用的键盘中断来实现子程序退出功能，所以为满足实验四的需要，删除实验三中的中断向量表设置的这一部分内容，并修改子程序。

删除部分如下（其余部分与实验三完全一致）：

```
;写入中断向量表
%macro write_inerrupt_vector 2
    pusha
    mov ax, 0h
    mov es, ax
    mov ax, %1
    mov bx, 4
    mul bx
    mov bp, ax
    mov ax, %2
    mov word[es:bp], ax
    add bp, 2
    mov ax, cs
    mov word[es:bp], ax
    popa
%endmacro

;定义20h中断向量
write_inerrupt_vector 20h, myinterrupt20h
```

```
myinterrupt20h:
    pusha
    print_message message1, 24, 24, 3
    mov ah, 01h
    int 16h
    jz no_input    ;没有按键，则跳转至no_input
    mov ah, 00h
    int 16h
    cmp al, 'q'
    jne no_input   ;若没按q，跳转至no_input
    jmp 800h:100h
no_input:
    popa
    iret
```

MyOS.asm：

在实验三的基础之上，新增了共计 5 个中断向量表的设置（由于键盘中断是在子程序开始运行时才设置的，所以此处不进行展示）。并伪包含了一个 services.asm 用于编写各中断服务的内置程序。

设置中断其实很简单，先将 ax 初始化，然后将段寄存器 es 赋值为 0，在偏移对应的中断位置处给予对应中断的偏移位置，再在+2 的地方加入段地址 cs，按 PPT 的来做就 ok 了，而且所有中断都是一样设置的。

```
;设置 20h 时间中断
xor ax,ax                                   ; AX = 0
mov es,ax                                   ; ES = 0
mov ax,offset Timer
mov word ptr es:[20h],offset Timer          ; 设置时钟中断向量的偏移地址
mov ax,cs
mov word ptr es:[22h],cs                    ; 设置时钟中断向量的段地址=CS

;设置 33h 的中断
xor ax,ax
mov es,ax
mov word ptr es:[51*4],offset BIOSService_1 ; 设置 33h 的偏移地址
mov ax,cs
mov word ptr es:[51*4+2],ax

;设置 34h 的中断
xor ax,ax
mov es,ax
mov word ptr es:[52*4],offset BIOSService_2 ; 设置 34h 的偏移地址
mov ax,cs
mov word ptr es:[52*4+2],ax

;设置 35h 的中断
xor ax,ax
mov es,ax
mov word ptr es:[53*4],offset BIOSService_3 ; 设置 35h 的偏移地址
mov ax,cs
mov word ptr es:[53*4+2],ax

;设置 36h 的中断
xor ax,ax
mov es,ax
mov word ptr es:[54*4],offset BIOSService_4 ; 设置 36h 的偏移地址
mov ax,cs
mov word ptr es:[54*4+2],ax
```

```
68      include services.asm
```

services.asm：

此处为展示内容的核心，实际上却没什么实际性的内容，因为编写方法与子程序完全一致，也就是完全套用实验一和实验二的编写手法即可。具体代码如下：

```
; 6 个系统服务程序

Timer:
    push ax
    push bx
    push cx
    push dx
    push bp
    push es

    dec byte ptr es:[count]             ; 递减计数变量
    jnz End1                            ; >0：跳转
    inc byte ptr es:[bn]                ; 自增变量 bn
    cmp byte ptr es:[bn],1              ; 根据 bn 选择跳转地址，1 则显示 /
    jz ch1
    cmp byte ptr es:[bn],2              ; 2 则显示 |
```

```
    jz ch2
    cmp byte ptr es:[bn],3              ; 3 则显示 \
    jz ch3
    cmp byte ptr es:[bn],4              ; 4 则显示 -
    jz ch4
    jmp showch
ch1:
    mov bp,offset str1
    jmp showch
ch2:
    mov bp,offset str2
    jmp showch
ch3:
    mov bp,offset str3
    jmp showch

ch4:
    mov byte ptr es:[bn],0
    mov bp,offset str4
    jmp showch

showch:
    mov ah,13h                          ; 功能号
    mov al,0                            ; 光标放到串尾
    mov bl,0ah                          ; 亮绿
    mov bh,0                            ; 第 0 页
    mov dh,24                           ; 第 24 行
    mov dl,78                           ; 第 78 列
    mov cx,1                            ; 串长为 1
    int 10h                             ; 调用 10H 号中断
    mov byte ptr es:[count],delay
End1:
    mov al,20h                          ; AL = EOI
    out 20h,al                          ; 发送 EOI 到主 8529A
    out 0A0h,al                         ; 发送 EOI 到从 8529A

    pop ax                              ; 恢复寄存器信息
    mov es,ax
    pop bp
    pop dx
    pop cx
    pop bx
    pop ax
    iret
```

```
    str1 db '\\'
    str2 db '|'
    str3 db '/'
    str4 db '-'
    delay equ 3                         ; 计时器延迟计数
    count db delay                          ; 计时器计数变量，初值=delay
    bn db 0

BIOSService_1:
    push ax
    push bx
    push cx
    push dx
    push bp

    mov ah,13h                      ; 功能号
    mov al,0                        ; 光标放到串尾
    mov bl,04h                      ;
    mov bh,0                        ; 第 0 页
    mov dh,0                        ; 第 0 行
    mov dl,0                        ; 第 0 列
    mov bp,offset MES1              ; BP=串地址
    mov cx,504                      ; 串长为 504
    int 10h                         ; 调用 10H 号中断

    pop bp
    pop dx
    pop cx
    pop bx
    pop ax

    mov al,33h                      ; AL = EOI
    out 33h,al                      ; 发送 EOI 到主 8529A
    out 0A0h,al                     ; 发送 EOI 到从 8529A
    iret                            ; 从中断返回

MES1:
    db "****************************************"
    db 0ah,0dh
    db "****                                ****"
    db 0ah,0dh
    db "****   name:  liaoyongbin           ****"
    db 0ah,0dh
```

```
    db "****   id:    17341097              ****"
    db 0ah,0dh
    db "****   class  4                     ****"
    db 0ah,0dh
    db "****                                ****"
    db 0ah,0dh
    db "****   int33                        ****"
    db 0ah,0dh
    db "****   program                      ****"
    db 0ah,0dh
    db "****                                ****"
    db 0ah,0dh
    db "****                                ****"
    db 0ah,0dh
    db "****                                ****"
    db 0ah,0dh
    db "****************************************"
    db 0ah,0dh,'$'

BIOSService_2:
    push ax
    push bx
    push cx
    push dx
    push bp

    mov ah,13h                  ; 功能号
    mov al,0                    ; 光标放到串尾
    mov bl,07h                  ;
    mov bh,0                    ; 第 0 页
    mov dh,5                    ; 第 5 行
    mov dl,44                   ; 第 44 列
    mov bp,offset MES2          ; BP=串地址
    mov cx,30                   ; 串长为 30
    int 10h                     ; 调用 10H 号中断

    pop bp
    pop dx
    pop cx
    pop bx
    pop ax

    mov al,34h                  ; AL = EOI
    out 34h,al                  ; 发送 EOI 到主 8529A
```

```
    out 0A0h,al                 ; 发送 EOI 到从 8529A
    iret                        ; 从中断返回

MES2:
    db "int34 is here!Can you see me?  "

BIOSService_3:
    push ax
    push bx
    push cx
    push dx
    push bp

    mov ah,13h                  ; 功能号
    mov al,0                    ; 光标放到串尾
    mov bl,07h                  ; 黄色
    mov bh,0                    ; 第 0 页
    mov dh,13                   ; 第 13 行
    mov dl,0                    ; 第 0 列
    mov bp,offset MES3          ; BP=串地址
    mov cx,479                  ; 串长为 479
    int 10h                     ; 调用 10H 号中断

    pop bp
    pop dx
    pop cx
    pop bx
    pop ax

    mov al,35h                  ; AL = EOI
    out 35h,al                  ; 发送 EOI 到主 8529A
    out 0A0h,al                 ; 发送 EOI 到从 8529A
    iret                        ; 从中断返回

MES3:
    db "          0 0           0 0         "
    db 0ah,0dh
    db "         0     0       0     0       "
    db 0ah,0dh
    db "       0         0   0         0      "
    db 0ah,0dh
    db "      0            0            0     "
    db 0ah,0dh
    db "      0                         0     "
```

```
    db 0ah,0dh
    db "       0                    0      "
    db 0ah,0dh
    db "         0                  0       "
    db 0ah,0dh
    db "           0                0        "
    db 0ah,0dh
    db "             0            0          "
    db 0ah,0dh
    db "               0        0            "
    db 0ah,0dh
    db "                 0    0              "
    db 0ah,0dh
    db "                   0                  "
    db 0ah,0dh,'$'


BIOSService_4:
    push ax
    push bx
    push cx
    push dx
    push bp

    mov ah,13h                  ; 功能号
    mov al,0                    ; 光标放到串尾
    mov bl,71h                  ; 白底深蓝
    mov bh,0                    ; 第 0 页
    mov dh,18                   ; 第 18 行
    mov dl,46                   ; 第 46 列
    mov bp,offset MES4          ; BP=串地址
    mov cx,28                   ; 串长为 28
    int 10h                     ; 调用 10H 号中断

    pop bp
    pop dx
    pop cx
    pop bx
    pop ax

    mov al,36h                  ; AL = EOI
    out 36h,al                  ; 发送 EOI 到主 8529A
    out 0A0h,al                 ; 发送 EOI 到从 8529A
    iret                        ; 从中断返回
```

```
MES4:
    db "Tomorrow is another day~ ^_^"

    ;*************** ********************
;*  键盘中断程序
;*************** ********************
keyDo:
    push ax
    push bx
    push cx
    push dx
    push bp

    inc byte ptr es:[c]
    cmp byte ptr es:[c],24
    jnz continue
    call keyInit

continue:
    inc byte ptr es:[odd]
    cmp byte ptr es:[odd],1
    je print
    mov byte ptr es:[odd],0
    jmp next

print:
    mov ah,13h                      ; 功能号
    mov al,0                        ; 光标放到串尾
    mov bl,0ah                      ; 亮绿
    mov bh,0                        ; 第 0 页
    mov dh,byte ptr es:[c]          ; 第 c 行
    mov dl,35                       ; 第 35 列
    mov bp, offset OUCH             ; BP=串地址
    mov cx,10                       ; 串长为 10
    int 10h                         ; 调用 10H 号中断

next:
    in al,60h

    mov al,20h                      ; AL = EOI
    out 20h,al                      ; 发送 EOI 到主 8529A
    out 0A0h,al                     ; 发送 EOI 到从 8529A
```

```
    pop bp
    pop dx
    pop cx
    pop bx
    pop ax

    iret                            ; 从中断返回

DelaySome:                          ; 延迟一段时间
    mov cx,delayTime
toDelay:
    mov word ptr es:[t],cx          ; 把 cx 的值保存到 t 中
    mov cx,delayTime
    loop1:loop loop1
    mov cx,word ptr es:[t]          ; 把 t 的值放回 cx ，恢复 cx
    loop toDelay
    ret

Clear: ;清屏
    MOV AX,0003H
    INT 10H
    ret

keyInit:                            ; 初始化 OUCH! OUCH! 显示的行数为 0
    mov byte ptr es:[c],0           ; 设置变量 c
    ret

OUCH:
    db "OUCH!OUCH!"
    c db 10
    odd db 1

    delayTime equ 40000
    t dw 0
```

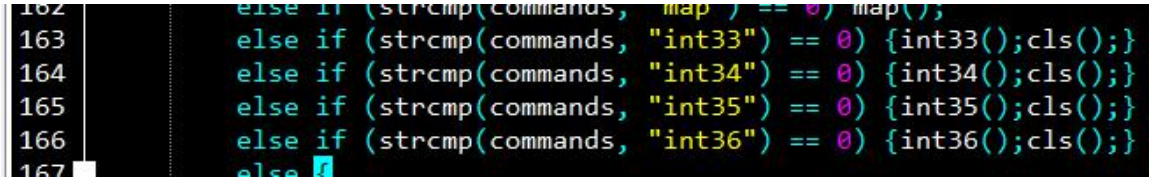
kliba.asm :（只需要改动 run 函数并新增启用 4 个软中断的内置程序就可以了。run 函数的修改就是设置中断向量与恢复中断向量的过程。不过此处容易出现 bug，也就是在修改中断的过程中容易影响寄存器 es，导致时钟偏移向量错误，也就是风火轮消失不见或者不转动的问题。解决方案很简单：保护寄存器。Push 和 pop 能很简单的解决这个问题。）

```
public _run
_run proc
    push es
    xor ax,ax
    mov es,ax
    push word ptr es:[9*4]                ; 保存 9h 中断
    pop word ptr ds:[0]
    push word ptr es:[9*4+2]
    pop word ptr ds:[2]

    mov word ptr es:[24h],offset keyDo    ; 设置键盘中断向量的偏移地址
    mov ax,cs
    mov word ptr es:[26h],ax


    mov ax,cs
    mov es,ax                             ;设置段地址，存放数据的内存基地址
    mov bx,0B100h                         ; ES:BX=读入数据到内存中的存储地址
    mov ah,2                              ; 功能号
    mov al,1                              ; 要读入的扇区数 1
    mov dl,0                              ; 软盘驱动器号（对硬盘和U盘，此处的值应改为80H）
    mov dh,0                              ; 磁头号
    mov ch,0                              ; 柱面号
    mov cl,byte ptr[_pro]                 ; 起始扇区号（编号从1开始）
    int 13H                               ; 调用13H号中断

    mov bx, 0B100h
    call bx                               ; 跳转到该内存地址

    xor ax,ax
    mov es,AX
    push word ptr ds:[0]                  ; 恢复 9h 中断
    pop word ptr es:[9*4]
    push word ptr ds:[2]
    pop word ptr es:[9*4+2]
    int 9h
    pop es
 run endp
```

```
;***************************************
;*  void _int33()                      *
;***************************************
; 调用 33h
public _Int33
_Int33 proc
    push ax
    push bx
    push cx
    push dx
    push es

    call Clear

    int 33h

    call DelaySome
...
    pop ax
    mov es,ax
    pop dx
    pop cx
    pop bx
    pop ax
    ret
_Int33 endp

```

由于 4 个内置子程序的编写都一模一样，沿用实验三的 c 与汇编交叉的写法就解决了，此处只贴一个 int33 的代码作为示例。

Kernal.c 程序:



改动如上，照应 kliba 汇编代码，引入了四个对应的汇编代码，并在操作系统中调用，调用的 4 个函数的具体代码就在 kliba.asm 中，4 个函数的内容就一个，就是调用对应的中断，而对应中断的样式，在 services.asm 中。

5、具体汇编、编译、运行的操作：同实验三

【实验心得】

这次实验，最大的收获无异于对中断和异步事件的理解。中断其实就是一种电信号，当它发生时，CPU 与对应中断端口之间产生脉冲，驱动处理器转移控制权。一个用户进程在任何时候、任何两条指令之间都可能被中断，即用户进程与中断之间是异步的，为了能正确响应中断并在中断程序执行完后能正确返回，便引出了保存现场的概念。

怎么保护现场呢？其实就是把进程的寄存器、返回地址等存入内存，虚拟出一套 CPU，待中断返回时恢复现场，就实现了异步事件的捕捉、执行与返回。所以我们设计中断时，要对对应中断的处理程序的偏移地址设置进对应的中断号位置里。中断的执行过程就像这样“中断——保存现场信息——调到对应中断的处理程序——跳回保护现场”，至少怎么跳来跳去的代码中断语法已经帮我们实现了，也就是 int 指令。

而这里也有一个很重要的问题，那就是段寄存器保护不到位，会导致各类中断跳不到对应的中断程序的位置，这就要求我们在执行一些会改变 es 或者 cs 等寄存器的操作时，先保护这些寄存器的值。

另外，为了满足实验 4 的要求，我对子程序进行了修改，不再采用 jmp 的方式跳回内核，而是采用 call，ret 的方式跳回去，所以新增了 pusha，popa 指令保护寄存器。批处理操作那里也随着这一变化进行了修改来适应新的跳回方式，不过也没什么大变化。

总的来说，在实验 3 的基础之上，实验 4 已经不再显得困难，步步为营，稳扎稳打就能成功。

【代码清单】

```
Project4
    --操作系统
        --OS.COM
        --OS.MAP
        --KERNAL.OBJ
        --KERNAL.C
        --KLIBA.ASM
        --MYOS.ASM
        --MYOS.OBJ
        --services.asm
    --工具包
        --DOSBox.exe（只是快捷方式）
        --mydoc.conf  (DOSBOX 的配置文件)
        --nasm.exe
        --nasmpath.bat
        --TASM.EXE
        --TCC.EXE
        --TLINK.EXT
    --软盘文件
        --实验 4.flp
    --引导程序
        --loading.asm
        --loading.com
    --用户程序
        --a1
        --a1.asm
        --a2
        --a2.asm
        --a3
        --a3.asm
        --a4
        --a4.asm
```