《操作系统原理实验》

实验报告

（实验四&五）

学 院 名 称：	数据科学与计算机学院
专业（班级）：	16 计科 2 班
学 生 姓 名：	朱志儒
学 号：	16337341
时 间：	2018 年 4 月 13 日

实验四&五 ： 中断机制编程技术与系统调用服务

一. 实验目的和要求

1、 掌握pc微机的实模式硬件中断系统原理和中断服务程序设计方法，实现对时钟、键盘/鼠标等硬件中断的简单服务处理程序编程和调试，让你的原型操作系统在运行以前已有的用户程序时，能对异步事件正确捕捉和响应。

2、 掌握操作系统的系统调用原理，实现原型操作系统中的系统调用框架，提供若干简单功能的系统调用。

3、 学习握掌c语言库的设计方法，为自己的原型操作系统配套一个c程序开发环境，实现用自建的c语言开发简单的输入/输出的用户程序，展示封装的系统调用。

二. 实验方案

1、 虚拟机配置

使用Vmware Workstation配置虚拟机，虚拟机的配置：核心数为1的处理器、4MB的内存、10MB的磁盘、1.44MB的软盘。

2、 软件工具与作用

Notepad++：编写程序时使用的编辑器；

16位编辑器WinHex：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

WinImage：可以创建虚拟软盘。

3、 基础原理

（1）异步事件

在操作系统世界里，许多活动或事件可能并发进行，随时可能发生或结束，不可预测。例如，在计算机硬件系统上，硬件系统的各个工作部件之间也可以并行工作，如CPU与I/O设备可以并行工作、不同I/O设备之间也可并行工作。这些硬件系统的并发活动提高了计算机系统的效率，但这些活动必须由操作系统进行有效的管理。计算机硬件系统提供中断技术，支持CPU与外部设备的并发工作，也利用中断技术处理硬件错误、支持程序调试、实现软件保护和信息安全等。

（2）中断技术

中断指对处理器正常处理过程的打断。中断与异常一样，都是在程序执行过程中的强制性转移，转移到相应的处理程序。中断分为三大类，即硬中断、软中断和异常/程序中断。

硬中断又名为外部中断，由外部（主要是外设[即I/O设备]）的请求引起的中断，硬中断又分为时钟中断（计时器产生，等间隔执行特定功能）、I/O中断（I/O控制器产生，通知操作完成或错误条件）和硬件故障中断（故障产生，如掉电或内存奇偶校验错误）。

软中断又名为内部中断，由指令的执行引起的中断，中断指令有软中断int n、溢出中断into、中断返回iret、单步中断TF=1等。

异常/程序中断是指令执行结果产生错误，如溢出、除0、非法指令、越界等。

（3）PC中断系统

x86 PC机的中断系统的功能强大、结构简单、使用灵活。采用32位的中断向量，即中断处理程序的映射地址，可处理256种不同类型的中断。x86处理器有两条外部中断请求线：NMI（Non Maskable Interrupt，不可屏蔽中断）和INTR（Interrupt Request，中断请求[可屏蔽中断]）。CPU是否响应在INTR线上出现的中断请求，取决于标志寄存器FLAGS中的IF标志位的状态值是否为1。可用机器指令STI/CLI置IF标志位为1/0来开/关中断。在系统复位后，会置IF=0（中断响应被关闭）。在任意中断被响应后，也会置IF=0（关中断）。若想允许中断嵌套，必须在中断处理程序中，用STI指令来打开中断。在NMI线上的中断请求，不受标志位IF的影响。CPU在执行完当前指令后，会立即响应。不可屏蔽中断的优先级要高于可屏蔽中断的。

（4）PC中断的处理过程

首先，保护断点现场，要将标志寄存器FLAGS压栈，然后清除它的IF位和TF位，再将当前的代码段寄存器CS和指令指针寄存器IP压栈。接着，执行中断处理程序，由于处理器已经拿到了中断号，它将该号码乘以4，毕竟每个中断在中断向量表中占4字节，就得到了该中断入口点在中断向量表中的偏移地址。从表中依次取出中断程序的偏移地址和段地址，并分别传送到IP和CS，自然地，处理器就开始执行中断处理程序了。注意，由于IF标志被清除，在中断处理过程中，处理器将不再响应硬件中断。如果希望更高优先级的中断嵌套，可以在编写中断处理程序时，适时用sti指令开放中断。最后，返回到断点接着执行，所有中断处理程序的最后一条指令必须是中断返回指令iret。这将导致处理器依次从堆栈中弹

出（恢复）IP、CS和FLAGS的原始内容，于是转到主程序接着执行

（5）中断向量

x86计算机在启动时会自动进入实模式状态，系统的BIOS初始化8259A的各中断线的类型，在内存的低位区（地址为0~1023[3FFH]，1KB）创建含256个中断向量的表IVT （每个向量[地址]占4个字节，格式为：16位段值:16位偏移值）。

当系统进入保护模式，IVT（Interrupt Vector Table，中断向量表）会失效，需改用IDT（Interrupt Descriptor Table，中断描述表），必须自己编程来定义8259A的各个软中断类型号和对应的处理程序。

（6）8259A的主要功能

在有多个中断源的系统中，接受外部的中断请求，并进行判断，选中当前优先级最高的中断请求，再将此请求送到CPU的INTR端，当CPU响应中断并进入中断子程序的处理过程后，中断控制器仍负责对外部中断请求的管理。

在一个8259A芯片中，有三个内部寄存器，即IMR（Interrupt Mask Register，中断屏蔽寄存器）用作过滤被屏蔽的中断、IRR（Interrupt Request Register，中断请求寄存器）用作暂时放置未被进一步处理的中断、ISR（In-Service Register，在使用中断）当一个中断正在被CPU处理时，此中断被放置在ISR中。

8259A还有一个单元叫做优先级分解器（Priority Resolver），当多个中断同时发生时，优先级分解器根据它们的优先级，将高优先级者优先传递给CPU。

（7）8259A的I/O端口

每个可编程中断控制器8259A都有两个I/O端口，主8259A所对应的端口地址为20h和21h，从8259A所对应的端口地址为A0h和A1h，程序员可以通过in/out指令读写这些端口，来操作这两个中断控制器。

（8）系统调用

操作系统除了执行用户的程序，还有义务为用户程序开发提供一些常用的服务。高级语言中，可以使用系统调用，实现软件重用的效果。操作系统提供的服务可以用多种方式供用户程序使用。

例如，子程序库静态链接，即采用子程序调用方式，如汇编语言中用call指令调用操作系统提供服务的子程序，静态链接到用户程序代码中，这种方式优点是程序执行快，最大缺点是用户程序内存和外存空间占用多，子程序库管理维护工作复杂。

内核子程序软中断调用，即采用软中断方式，如汇编语言中用int指令调用操作系统提供服务的子程序，系统服务的子程序在内核，这种方式的优点是服务由系统提供，程序效率较高，且被所有用户程序代码共享，有利于节省内存，最大缺点是需要防止内核再入或内核设计为可再入，且用户程序陷入内核和内核返回用户程序的开销较大。

子程序库动态链接，即采用动态链接技术，操作系统在运行时响应子程序调用，加载相应的子服务程序并链接致用户地址空间，这种方式优点是可由多方提供服务程序，服务更内容丰富，增加和变更服务方便，最大缺点是链接耗时多，程序响应变慢，实现复杂。

（9）软中断实现系统服务

a. 使用BIOS调用，其实与内核子程序软中断调用方式原理是一样的，每一种服务由一个子程序实现，指定一个中断号对应这个服务，入口地址放在中断向量表中，中断号固定并且公布给用户，用户编程时才可以中断调用，参数传递可以使用栈、内在单元或寄存器。

b. 使用系统调用，因为操作系统要提供的服务更多，服务子程序数量太多，但中断向量有限，因此，实际做法是专门指定一个中断号对应服务处理程序总入口，然后再将服务程序所有服务用功能号区分，并作为一个参数从用户中传递过来，服务程序再进行分支，进入相应的功能实现子程序。这种方案至少要求向用户公开一个中断号和参数表，即所谓的系统调用手册，供用户使用。如果用户所用的开发语言是汇编语言，可以直接使用软中断调用。如果使用高级语言，则要用库过程封装调用的参数传递和软中断等指令汇编代码。并且规定系统调用服务的中断号是21h。

（10）系统调用服务程序

系统调用功能实现可以用汇编与c语言混合编程，在内核增加一个过程作为系统调用的总入口，用以获取参数和分析功能号，再根据功能号产生分枝结构，根据系统调用号决定选择对应的分支完成相应的服务。通常每个分枝实现一种系统调用功能，简单的功能可以用汇编实现，也可以用c程序实现。

4、 方案思想

（1）在kliba.asm文件中编写一个时间中断，使得在操作系统工作的期间内，在屏幕的右下角轮流动态显示‘|’，‘\’，‘—’，‘/’。

（2）在kliba.asm文件中编写一个键盘中断程序，使得在用户程序运行时，当有键盘按下时，在屏幕的左上角闪现“OUCH! OUCH!”，当连续按下按键时，从屏幕的左上角到右下角会连续闪现“OUCH! OUCH!”。

（3）在kliba.asm文件中分别编写34号、35号、36号、37号和38号中断。

· 34号中断会在屏幕左上方1/4区域显示一个带有Hi字符的气泡。

· 35号中断会在屏幕右上方1/4区域的中心位置显示一句“This experiment is difficult!”。

· 36号中断会在屏幕左下方1/4区域显示一个超人标志。

·37号中断会在屏幕右下方1/4区域的中心位置显示一句“It’s a nice day!”。

· 38号中断会使整个系统休眠3秒钟。

（4）在kliba.asm文件的run()函数中在加载软盘数据前，将键盘中断载入中断向量表，使得在用户程序运行时，响应键盘中断，显示“OUCH！OUCH！”，再另外编写一个名为run_test()的加载程序，将测试所有系统调用功能的用户程序加载入内存，并能正常响应按键‘q’返回操作系统。

（5）在kliba.asm文件中编写21h中断实现系统调用服务，仿照DOS系统在ah寄存器存储服务号，经过判断实现相应的功能。

·1号服务号会在屏幕左上方1/4区域显示“I Love The Operating System!”。

· 2号服务号会在屏幕右上方1/4区域显示“When call system server, you will see this!”。

· 3号服务号会在屏幕左下方1/4区域显示“Thank you!”。

（6）在kernal.c文件中加入一些功能，使得用户在输入“int+中断号”指令时，调用相关的中断服务程序。当用户输入指令“t”时，将加载并运行测试所有系统调用功能的用户程序。

（7）为操作系统配套一个C程序开发环境时，首先编写一个名为stdio.asm的文件，里面实现了基本的printChar(char str)输出一个字符函数、getChar(char in)获取一个输入字符函数、cls()清屏函数、sleep()系统休眠函数。

然后编写一个名为stdio.c的文件，导入stdio.asm的printChar(char str)、

getChar(char in)、cls()、sleep()这些函数，在此基础上，实现了

· print(char *shr)程序，显示一个字符串；

· getline(char *ptr, int len)程序，读取输入字符串；

· strcmp(char *str1, char *str2)程序，比较两个字符串；

· strlen(char *str)程序，计算字符串长度；

· substr(char *src, char *sstr, int pos, int len)程序，得到字符串的子字符串。

（8）编写tsyscall.asm和ctsyscall.c两个文件测试所有的中断程序、系统调用服务和一个C程序开发环境。在操作系统中输入指令 t 即可运行。

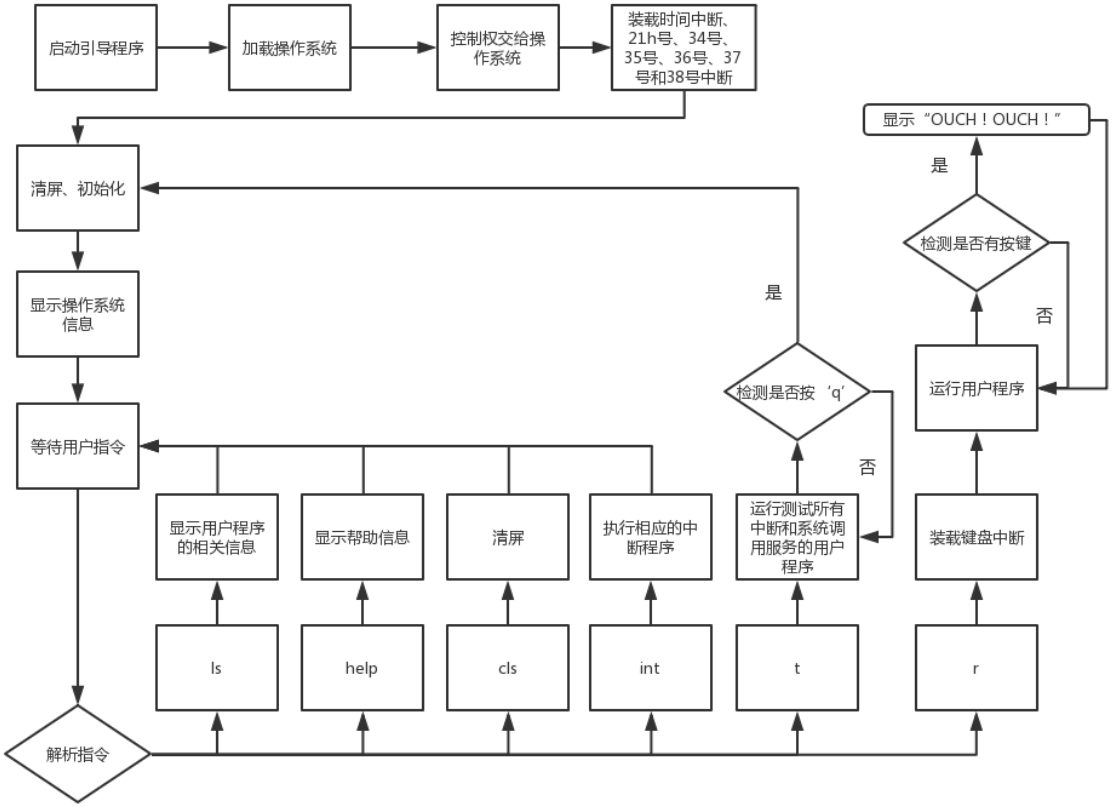
值得注意的是：

在ctsyscall.c文件中 #include “stdio.c”，即导入了C库，调用测试了C库中的print(char *str) 显示一个字符串程序、getline(char *ptr, int len)读取输入字符串程序和strcmp(char *str1, char *str2)比较两个字符串程序。

由于这个测试程序使用了C库，所以生成.com文件的方式有所不同，首先使用指令ta stdio.asm生成stdio.obj文件，再使用指令ta tsyscall.asm生成tsyscall.obj文件，接着使用指令tc ctsyscall.c生成ctsyscall.obj文件，最后使用指令tlink /3 /t tsyscall.obj stdio.obj ctsyscall.obj, tsyscall.com生成tsyscall.com。

注：本来想实现跑边框版本的时间中断，但由于没有立即进行动态画框，才改为动态显示‘|’，‘\’，‘—’，‘/’。（实现跑边框版本的时间中断的镜像文件为MYOS.img）

5、 程序流程



6、 算法和数据结构

算法：

（1）在时间中断中，设置一个名为cccount的变量用于计数，当cccount为0时才显示，显示后将cccount置为8，若不为0，则将cccount减1；设置一个名为tmp的变量用于计数来控制显示内容，当tmp为1时显示‘/’，当tmp为2时显示‘|’，当tmp为3时显示‘\’，当tmp为4时显示‘—’,再将tmp置为0。每次显示后将tmp加1。

（2）在34号、35号、36号和37号中断均是显示字符串，所以调用BIOS的10h中断13h号功能即可。

（3）键盘中断中，设置名为odd，cnn的变量，每次有键盘中断时odd加1，当odd为1时，在第cnn行第cnn列显示“OUCH!OUCH!”并将cnn加1，若cnn为25则将cnn置为0；当odd不为1时，将odd置为0且不显示。

（4）休眠功能中断中，只要执行多次循环即可将实现时延。

（5）21h中断实现系统调用服务中，判断ah寄存器的值实现相应的功能。

（6）C库中的函数算法在实验三的报告中有说明，在此就不在复述。

7、 程序关键模块

装载时间中断和21h、34号、35号、36号、37号和38号中断：

```
mov word ptr es:[20h], offset Timer          ；时间按中断
mov word ptr es:[22h], cs

mov word ptr es:[33*4], offset int_21h       ；21h号中断
mov word ptr es:[33*4+2], cs

mov word ptr es:[34*4], offset int_34        ；34号中断
mov word ptr es:[34*4+2], cs

mov word ptr es:[35*4], offset int_35        ；35号中断
mov word ptr es:[35*4+2], cs

mov word ptr es:[36*4], offset int_36        ；36号中断
mov word ptr es:[36*4+2], cs

mov word ptr es:[37*4], offset int_37        ；37号中断
mov word ptr es:[37*4+2], cs

mov word ptr es:[38*4], offset sleep         ；38号中断
mov word ptr es:[38*4+2], cs
```

在run()程序中装载键盘中断替换原来的9h中断：

```
mov word ptr es:[24h], offset KeyInt
mov word ptr es:[26h], cs
```

时间中断：

在时间中断中，设置一个名为cccount的变量用于计数，当cccount为0时才显示，显示后将cccount置为8，若不为0，则将cccount减1；设置一个名为tmp的变量用于计数来控制显示内容，当tmp为1时显示‘/’，当tmp为2时显示‘|’，当tmp为3时显示‘\’，当tmp为4时显示‘—’,再将tmp置为0。每次显示后将tmp加1。

代码如下：

```
Timer:
    push ax                          ;寄存器压栈
    push bx
    push cx
    push dx
    push bp
    push es
    dec byte ptr es:[cccount]        ;递减计数变量
    jnz fin                          ; >0  跳转
    inc byte ptr es:[tmp]            ;自增tmp
    cmp byte ptr es:[tmp], 1         ;根据tmp选择显示内容
    jz ch1                           ;1显示‘/’
    cmp byte ptr es:[tmp], 2         ;2显示‘|’
    jz ch2
    cmp byte ptr es:[tmp], 3         ;3显示‘\’
    jz ch3
    cmp byte ptr es:[tmp], 4         ;4显示‘-’
    jz ch4
    jmp showch
ch1:
    mov bp, offset str1              ;1显示‘/’
    jmp showch
ch2:
    mov bp, offset str2              ;2显示‘|’
    jmp showch
ch3:
    mov bp,offset str3               ;3显示‘\’
    jmp showch
ch4:
    mov byte ptr es:[tmp],0          ;将tmp置0
    mov bp, offset str4              ;4显示‘-’
    jmp showch
```

```
showch:
    mov ah,13h                          ; 功能号
    mov al,0                            ; 光标放到串尾
    mov bl,0Fh                          ; 0000：黑底、1111：亮白字
    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:[cccount],dddelay
fin:
    mov al,20h                          ; AL = EOI
    out 20h,al                          ; 发送EOI到主8529A
    out 0A0h,al                         ; 发送EOI到从8529A
    pop es                              ; 恢复寄存器信息
    pop bp
    pop dx
    pop cx
    pop bx
    pop ax
    iret                                ;退出时间中断
    str1 db '/'
    str2 db '|'
    str3 db '\'
    str4 db '-'
    dddelay equ 8                        ; 计时器延迟计数
    cccount db dddelay                   ; 计时器计数变量，初值=dddelay
    tmp db 0
```

34号中断：

调用BIOS的10h中断13h号功能显示字符串。代码如下：

```
int_34:
    push es
    push ds
    push bp                             ; 寄存器压栈
    mov ax, cs
    mov ds, ax
    mov es, ax
    mov ah, 13h                         ; 13h号功能
    mov al, 0                           ; 字符串中只含显示字符
    mov bl, 0ah                         ; 亮绿色
```

```
    mov bh, 0                                   ; 0页
    mov dh, 0                                   ; 第0行
    mov dl, 0                                   ; 第0列
    mov bp, offset mes1
    mov cx, 360                                 ; 字符串长度
    int 10h                                     ; 调用10h中断
    pop bp                                      ; 恢复寄存器
    pop ds
    pop es
    iret
mes1:                                           ; 数据定义
    db "      ====================      ", 0ah, 0dh
    db "     =      **      **     **    =     ", 0ah, 0dh
    db "    =       **      **     **      =   ", 0ah, 0dh
    db " =          **      **             = ", 0ah, 0dh
    db "=           ********    **           =", 0ah, 0dh
    db "=           ********    **           =", 0ah, 0dh
    db " =          **      **     **       = ", 0ah, 0dh
    db "    =       **      **     **      =   ", 0ah, 0dh
    db "     =      **      **     **    =     ", 0ah, 0dh
    db "      ====================      ", 0ah, 0dh
    db "                              =     ", 0ah, 0dh
    db "                                =   ", 0ah, 0dh
```

35号、36号、37号中断程序与34号中断程序的代码相似就不再贴上代码。

键盘中断：

键盘中断中，设置名为odd，cnn的变量，每次有键盘中断时odd加1，当odd为1时，在第cnn行第cnn列显示“OUCH!OUCH!”并将cnn加1，若cnn为25则将cnn置为0；当odd不为1时，将odd置为0且不显示。

代码如下：

```
KeyInt:
    push ax                                     ; 寄存器压栈
    push bx
    push cx
    push dx
    push bp
    push es
```

```
        push ds
        mov ax, cs
        mov ds, ax
        mov es, ax
        inc byte ptr es:[odd]                ; odd加1
        cmp byte ptr es:[odd], 1             ; 判断odd是否为1
        je print                             ; odd为1则显示”OUCH!OUCH!”
        mov byte ptr es:[odd], 0             ;odd置0
        jmp final                            ; 退出键盘中断
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:[cnn]             ; 第 cnn 行
        mov dl,byte ptr es:[cnn]             ; 第 cnn 列
        mov bp, offset OUCH                  ; BP=串地址
        mov cx,10                            ; 串长为 10
        int 10h                              ; 调用10H号中断
        call Delay                           ; 显示延迟一段时间
        mov ax, 0601h                        ;清除OUCH!OUCH!
        mov bh, 0Fh
        mov ch, byte ptr es:[cnn]            ; “OUCH！OUCH！”字符串开头
        mov cl, byte ptr es:[cnn]
        mov dh, byte ptr es:[cnn]
        mov dl, byte ptr es:[cnn]
        add dl, 10                           ; “OUCH！OUCH！”字符串结尾处
        int 10h      ·                       ; 调用10H号中断
        inc byte ptr es:[cnn]                ; cnn加1
        cmp byte ptr es:[cnn], 25            ; 判断cnn是否为25
        jne final
        mov byte ptr es:[cnn], 0             ; cnn为25则置0
final:
        in al,60h                            ; 清空键盘输入缓冲区
        mov al,20h                           ; AL = EOI
        out 20h,al                           ; 发送EOI到主8529A
        out 0A0h,al                          ; 发送EOI到从8529A
        pop ds                               ; 恢复寄存器
        pop es
        pop bp
        pop dx
        pop cx
        pop bx
        pop ax
```

```
        iret                                ; 从中断返回
    OUCH:                                   ; 数据定义
        db "OUCH!OUCH!"
        cnn db 0
        odd db 1
```

21h中断系统调用服务程序:

仿照DOS系统，在用户程序中调用21h中断前需要将服务号存进ah寄存器，在21h中断中根据ah寄存器的值执行相应的服务。

代码如下:

```
int_21h:
    push bp                             ;寄存器压栈保护
    push ds
    push es
    mov bx, cs
    mov ds, bx
    mov es, bx
    cmp ah, 1                           ; 判断ah是否为1
    je showstring1
    cmp ah, 2                           ; 判断ah是否为2
    je showstring2
    cmp ah, 3                           ; 判断ah是否为3
    je showstring3
    jmp end21h                          ; ah功能号无效退出中断
showstring1:                            ; 执行1号功能
    mov bp, offset string1
    mov dh, 3
    mov dl, 6
    mov cx, 28
    mov bl, 0ah
    jmp showstring
showstring2:                            ; 执行2号功能
    mov bp, offset string2
    mov dh, 5
    mov dl, 37
    mov cx, 43
    mov bl, 0fh
    jmp showstring
showstring3:                            ; 执行3号功能
```

```
    mov bp, offset string3
    mov dh, 19
    mov dl, 8
    mov cx, 10
    mov bl, 71h
    jmp showstring
showstring:
    mov ah, 13h                          ; BIOS 13h功能号
    mov al, 0
    mov bh, 0
    int 10h                              ; BIOS 10h中断
end21h:
    pop es                               ; 恢复寄存器
    pop ds
    pop bp
    iret                                 ; 退出21h中断
string1 db "I Love The Operating System!"   ; 数据定义
string2 db "When call system server, you will see this!"
string3 db "Thank you!"
```

stdio.asm中的printChar(char str)输出一个字符函数：

```
public _printChar
_printChar proc
    push bp
        mov bp,sp
        mov al,[bp+4]
        mov bl,0
        mov ah,0eh
        int 10h
        mov sp,bp
    pop bp
    ret
_printChar endp
```

stdio.asm中的getChar(char in)获取一个输入字符函数：

```
public _getChar
_getChar proc
    mov ah,0
    int 16h
    mov byte ptr[_input], al
    ret
_getChar endp
```

stdio.asm中的cls()清屏函数：

```
public _cls
_cls proc                       ; 清屏
        push ax
        push bx
        push cx
        push dx
            mov ax, 600h  ; AH = 6,    AL = 0
            mov bx, 700h  ; 黑底白字(BL = 7)
            mov cx, 0     ; 左上角: (0, 0)
            mov dx, 184fh ; 右下角: (24, 79)
            int   10h     ; 显示中断

            mov ah, 02h
            mov bh, 0
            mov dx, 0100h
            int 10h
        pop dx
        pop cx
        pop bx
        pop ax
        ret
_cls endp
```

stdio.asm中的sleep()系统休眠函数：

```
public _sleep
_sleep proc
    push cx
    mov cx, 50
loop3:
    call Delay
    loop loop3
    pop cx
    ret
_sleep endp
```

stdio.c中的print(char *shr)程序，显示一个字符串：

```
void print(char *str) {
    while(*str != '\0') {
        printChar(*str);
        str++;
    }
}
```

stdio.c中的getline(char *ptr, int len)程序，读取输入字符串：

```
void getline(char *ptr, int length) {
    int count = 0;
    if (length == 0) {
        return;
    }
    else {
        getChar();
        while (input != 13) {
            printChar(input);
            ptr[count++] = input;
            if (count == length) {
                ptr[count] = '\0';
                print("\n\r");
                return;
            }
            getChar();
        }
        ptr[count] = '\0';
        print("\n\r");
        return;}}
```

stdio.c中的strcmp(char *str1, char *str2)程序，比较两个字符串：

```
int strcmp(char *str1, char *str2) {
    while ((*str1) && (*str2)) {
        if (*str1 != *str2) {
            if (*str1 < *str2) return -1;
            return 1;
        }
        ++str1;
        ++str2;
    }
    return (*str1) - (*str2);}
```

stdio.c中的strlen(char *str)程序，计算字符串长度：

```
int strlen(char *str) {
    int i = 0;
    while(*(str++)) i++;
    return i;}
```

stdio.c中的substr(char *src, char *sstr, int pos, int len)程序，得到字符串的子字符串：

```
int substr(char *src, char *sstr, int pos, int len) {
    int i = pos;
    for (; i < pos + len; ++i) {
        sstr[i - pos] = src[i];
    }
    sstr[pos + len] = '\0';
    return 1;}
```

csyscall.c中的测试程序代码：

```
#include "stdio.c"
cmain() {
    cls();
    while (1) {
        char IN[100];
        print("Please input something(input 'q' to quit): ");
        getline(IN, 100);
        if (strcmp(IN, "q") == 0) break;
        print("\n\rYour input is ");
        print(IN);
        print("\n\n\r");
    }
}
```

注：跑边框版本的时间中断代码：

```
Timer:
    push ax                                    ;寄存器压栈
    push bx
    push cx
    push dx
    push bp
```

```
        push es
        jmp begin
position:
        xor ax, ax
        mov bx, word ptr ds:[pos]
        mov al, bl
        mov cx, 80
        mul cx
        xor cx, cx
        mov cl, bh
        add ax, cx
        mov cx, 2
        mul cx
        mov bp, ax
        ret
show:
        call position
        mov al, byte ptr ds:[char]
        mov ah, byte ptr ds:[color]
        mov word ptr es:[bp], ax
        ret
change_position:
        mov ax, word ptr ds:[pos]
        mov bx, word ptr ds:[speed]
        cmp bh, 1
        je right
        cmp bl, 1
        je down
        cmp bl, 0
        je left
        cmp bh, 0
        je up
right:
        add ah, bh
        cmp ah, 79
        jb back
        mov bh, 0
        mov bl, 1
        jmp back
down:
        add al, bl
        cmp al, 24
        jb back
        mov bh, -1
```

```
        mov bl, 0
        jmp back
left:
        add ah, bh
        cmp ah, 0
        ja back
        mov bh, 0
        mov bl, -1
        jmp back
up:
        add al, bl
        cmp al, 0
        ja back
        mov bh, 1
        mov bl, 0
        jmp back
back:
        mov word ptr ds:[pos], ax
        mov word ptr ds:[speed], bx
        ret
change_color_char:
        cmp byte ptr ds:[color], 0Fh
        jnz chage1
        mov byte ptr ds:[color], 0
chage1:
        add byte ptr ds:[color], 1
        cmp byte ptr ds:[char], 'Z'
        jnz chage2
        mov byte ptr ds:[char], 'A'
chage2:
        add byte ptr ds:[char], 1
        ret
have_fun:
        call change_position
        call show
        ret
begin:
        mov ax,0B800h                   ; 文本窗口显存起始地址
        mov es,ax                       ; es = B800h
loop11:
        dec word ptr ds:[count]             ; 递减计数变量
        jnz loop11                              ; !=0：跳转
        mov word ptr ds:[count],delayy
        dec word ptr ds:[dcount]            ; 递减计数变量
```

```
        jnz loop11
        mov word ptr ds:[count],delayy
        mov word ptr ds:[dcount],ddelay
        call have_fun
        call change_color_char
fin:
        mov al,20h                          ; AL = EOI
        out 20h,al                          ; 发送EOI到主8529A
        out 0A0h,al                         ; 发送EOI到从8529A
        pop es                              ; 恢复寄存器信息
        pop bp
        pop dx
        pop cx
        pop bx
        pop ax
        iret
        delayy equ 5000                     ; 数据定义段
        ddelay equ 380
        count dw delayy
        dcount dw ddelay
        color db 0Fh
        speed dw 0100h
        pos dw 0500h
        char db 'A'
```

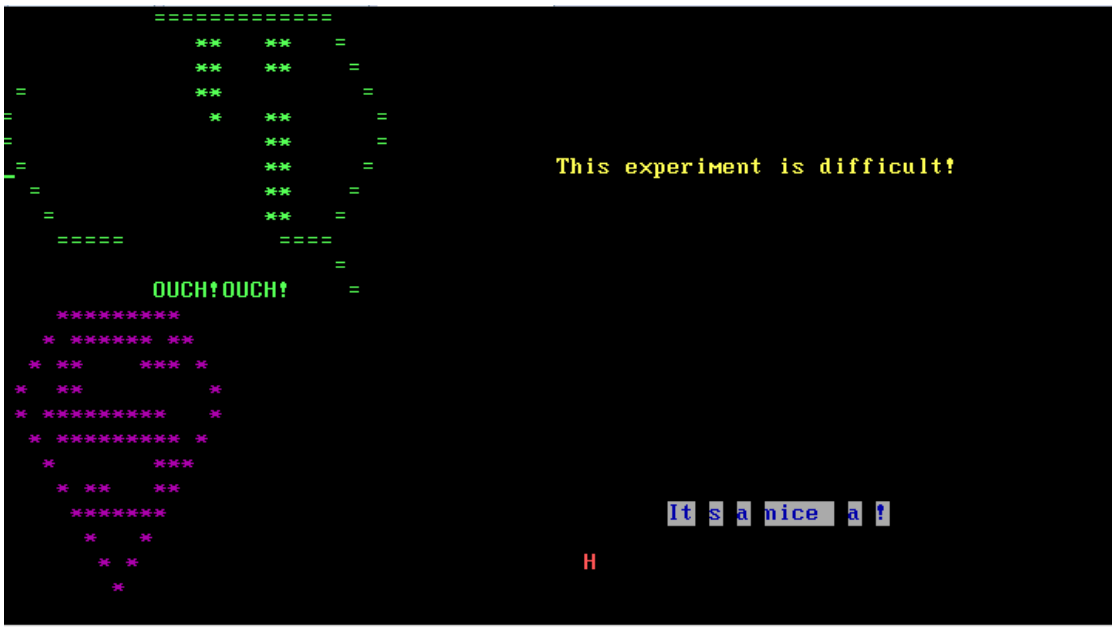
三. 实验过程和结果

运行用户程序1测试35号、36号、37号中断和键盘中断，如图：

运行用户程序2测试34号、35号、36号、37号中断和键盘中断，如图：

运行用户程序3测试34号、35号、37号中断和键盘中断，如图：

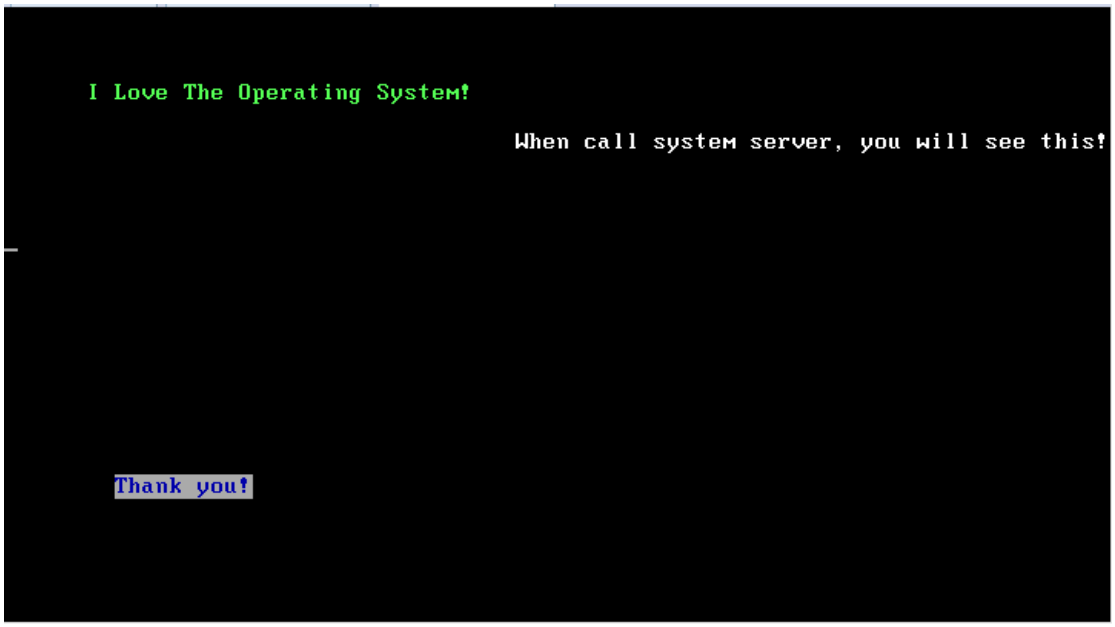
运行用户程序4测试34号、35号、36号、37号中断和键盘中断，如图：



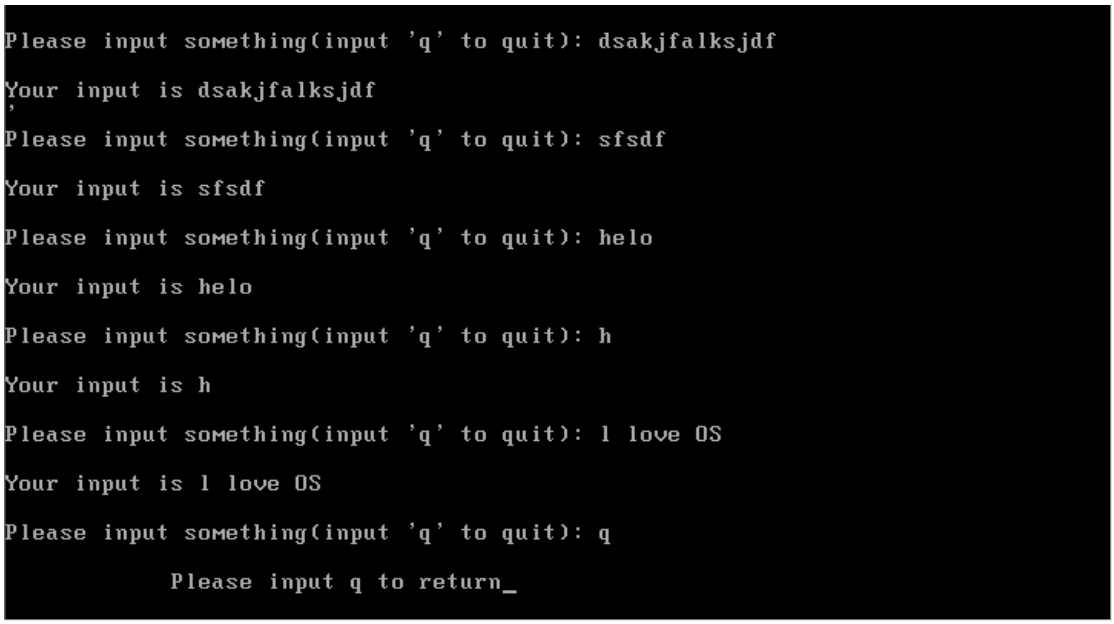
在操作系统界面可以看到右下角的动态显示‘|’，‘\’，‘—’，‘/’

输入指令t后，测试所有的中断程序。首先显示34号中断内容，调用38号中断停几秒后，再显示35号中断内容，调用38号中断停几秒后，再显示36号中断内容，调用38号中断停几秒后，再显示37号中断内容。如图：

过几秒后显示系统调用1、2、3号的服务号内容。如图：



再过几秒后，即可测试基本的输入出。如图：



再按下‘q’即可返回操作系统（期间可能会听到声音）。

注：通过时间中断实现跑边框效果如图：

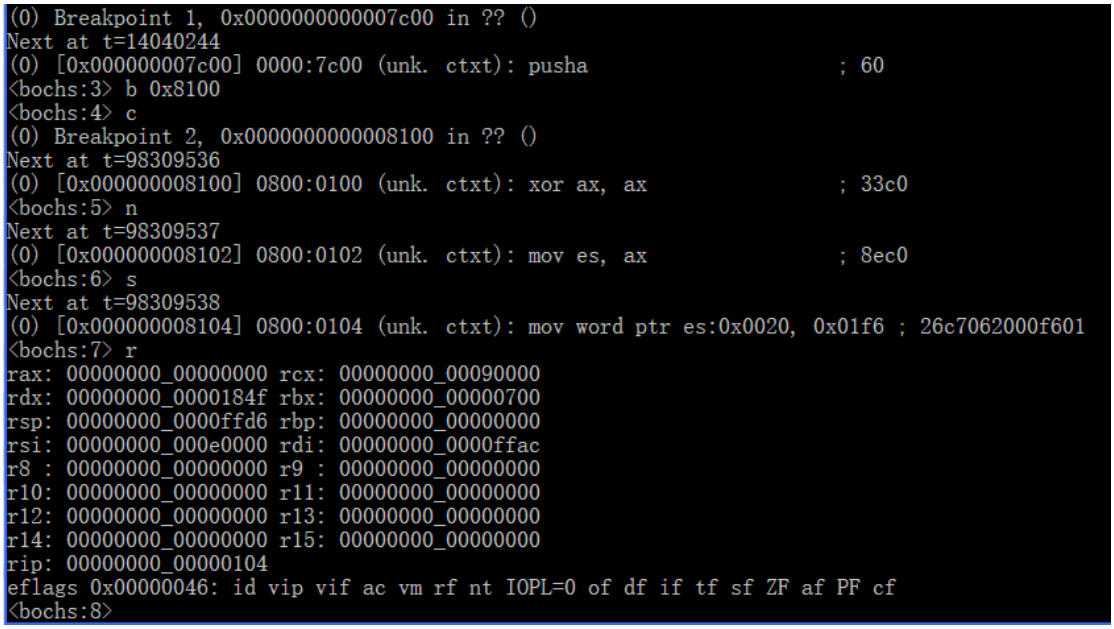
其他实现效果与上述相似就不再一一截图。

四. 实验创新点

使用Bochs调试

前几个实验都是脑袋跑程序，眼睛debug。每次操作系统出现问题，我都是对着代码想着程序在哪里出现了问题，无法查看程序运行到这里时各个寄存器的值和内存的值，每次都是想当然的认为哪里哪里出错了，然后再修改，这样debug的效率是特别低的，一个小问题可能会花费一下午的时间。

使用Bochs调试的话就会方便很多。比如说，我可以设置断点，使用指令b 0x7c00，就会在0x7c00处设置一处断点；使用指令c，即可使操作系统运行到0x7c00处停止；再使用指令r，即可查看当前寄存器的值；使用指令s，即可单步执行；使用指令n，也可单步执行，只是遇到函数则跳过；使用指令u/20，即可查看接下来的20条汇编指令；使用指令print-stack，可以查看堆栈等等。使用效果如图。



```
<bochs:8> u/10
00008104: (                    ): mov word ptr es:0x0020, 0x01f6 ; 26c7062000f601
0000810b: (                    ): mov word ptr es:0x0022, cs ; 268c0e2200
00008110: (                    ): mov word ptr es:0x0088, 0x0273 ; 26c70688007302
00008117: (                    ): mov word ptr es:0x008a, cs ; 268c0e8a00
0000811c: (                    ): mov word ptr es:0x008c, 0x03fb ; 26c7068c00fb03
00008123: (                    ): mov word ptr es:0x008e, cs ; 268c0e8e00
00008128: (                    ): mov word ptr es:0x0090, 0x0438 ; 26c70690003804
0000812f: (                    ): mov word ptr es:0x0092, cs ; 268c0e9200
00008134: (                    ): mov word ptr es:0x0094, 0x0530 ; 26c70694003005
0000813b: (                    ): mov word ptr es:0x0096, cs ; 268c0e9600
<bochs:9> print-stack
Stack address size 2
| STACK 0xffd6 [0x8f11]
| STACK 0xffd8 [0xf000]
| STACK 0xffda [0x0001]
| STACK 0xffdc [0x0000]
| STACK 0xffde [0x0000]
| STACK 0xffe0 [0x0000]
| STACK 0xffe2 [0x0000]
| STACK 0xffe4 [0x0000]
| STACK 0xffe6 [0x0000]
| STACK 0xffe8 [0x0000]
| STACK 0xffea [0xffff]
| STACK 0xffec [0x0000]
| STACK 0xffee [0x7c00]
| STACK 0xfff0 [0x0000]
| STACK 0xfff2 [0x0000]
| STACK 0xfff4 [0x0000]
<bochs:10>
```

五. 实验总结

总结:

这次实验的重点是编写中断服务程序，由于前几次实验我都编写了20h中断服务来响应用户的按键并退出用户程序，有前几次实验的基础，这次实验中编写34号、35号、36号、37号和38号中断服务并不是特别困难。

这次实验的最大困难就在于编写时间中断和键盘中断。

在编写键盘中断时， 我借鉴了老师ppt上的代码，在老师代码的基础上进行了一次魔改。原本准备实现在屏幕边缘处动态画框，画框时字符和颜色的不断变化的，但由于在实际测试时发现加载并进入操作系统后，并没有立即进行动态画框，而是等待13秒后才出现动态画框（**实现的镜像名为MYOS**）。我在这个问题上纠结了很久，并没有找到问题的关键所在，于是就放弃了，改为在屏幕的右下角动态显示‘|’，‘\’，‘—’，‘/’，经过我的一番努力最终成功了。

在编写键盘中断程序时，经查阅资料发现每当有键盘按下时就会触发一次9h号中断，那么要实现每当键盘有按键时，屏幕适当位置显示“OUCH! OUCH!”这一功能，就要将原本的9h中断替换为自己编写的9h中断。我在载入用户程序前才替换9h中断，这样就不会影响在操作系统下的指令输入了。

问题和解决方法：

（1）在测试最初版本的键盘中断时，我发现进入用户程序后键盘中断并没有起作用，按下按键屏幕上没有显示“OUCH！OUCH！”。当初以为是没有成功将9h中断载入中断向量表，使用bochs调试发现9h中断成功载入中断向量表了。看来问题并没有出现在这里。

经查阅资料后，我知道当键盘上有键按下时，会产生该键的扫描码，并被送入端口地址为60h的寄存器中，然后，CPU会接受到9h号中断，如果该键是字符码，会将扫描码连同字符码(ASCII码)一起放入缓冲区，而如果该键是控制键和切换键，则会改变内存中对应键盘状态的字节中。原来没有清空输入缓冲区才是罪魁祸首，于是我在键盘中断中加入一句 in al, 60h指令便解决了问题。

（2）在测试键盘中断时，我发现每次我按一次键，屏幕上会出现两个“OUCH! OUCH！”，而我预期的是只显示一个“OUCH！OUCH！”。

经查阅资料，我了解到按下按键会产生一个9h中断，松开按键又会产生一个9h中断。所以为了只显示一个“OUCH！OUCH！”，我在键盘中断中添加了一个名为odd的变量，每次进入9h中断就将odd加1，当odd为1时才显示，反之，不显示，将odd置为0。这样每一次按键只会显示一个“OUCH！OUCH！”。

（3）在测试所有的中断时，我发现在中断中显示的字符串都是乱码。经过bochs调试后，我才知道出现这样的问题是因为我没有修改中断中的ds、es的值，

它们均为用户程序的段值而不是中断程序所在的段值。所以我在中断程序中先将ds、es压栈保存，再将cs的值赋给ds、es，最后在退出中断前恢复之前压栈保存的ds和es。

六. 参考文献

1、《x86 PC汇编语言，设计与接口》

2、汇编程序：按键松开时中断的处理

https://blog.csdn.net/sxhelijian/article/details/72869738

3、Windows 下BOCHS的使用

http://blog.51cto.com/liyuelumia/1562508