实验报告 实验四

姓名：王钦　学号：13349112　班级：计科二班

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和键盘中断响应程序
3. 扩展MyOS，增加一个系统服务的实现。
4. 建立具有简单异步事件处理的原型操作系统。

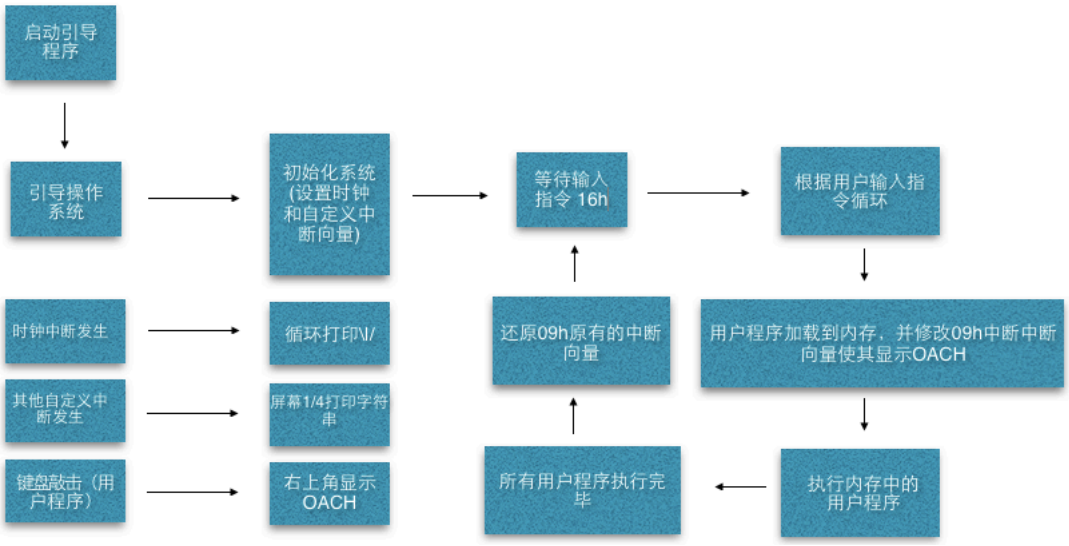
实验内容

在实验三的基础上，进化你的原型操作系统，增加下列操作系统功能：
(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，用户程序运行时，键盘事件有事反应：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，程序服务是分别在屏幕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，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。

实验平台

gcc+nasm+Linux+vim

算法流程图

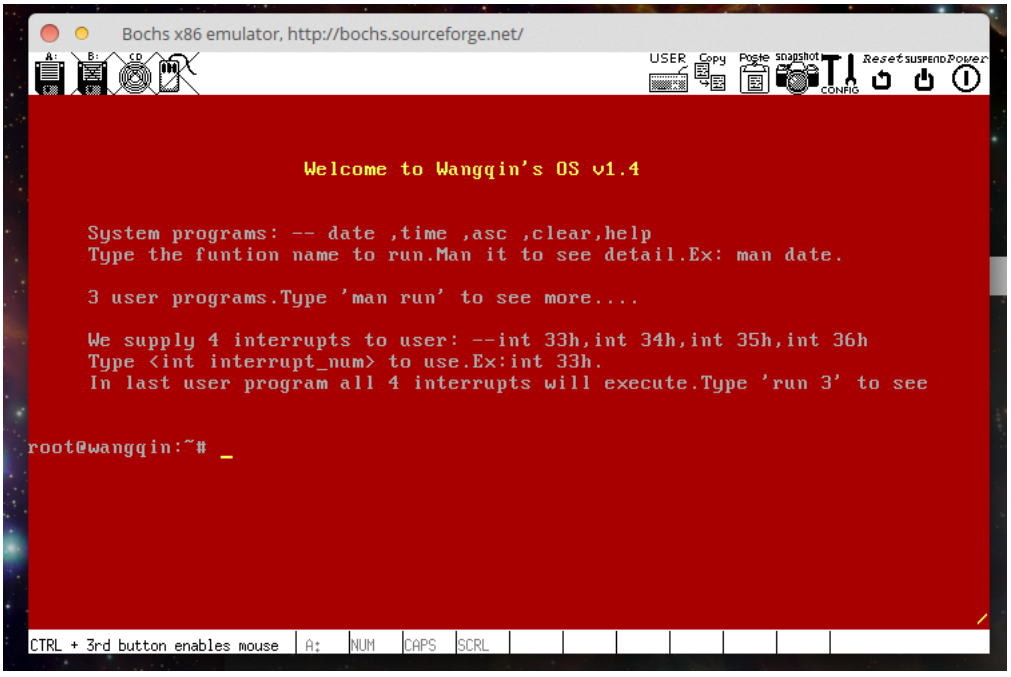


实验步骤及效果

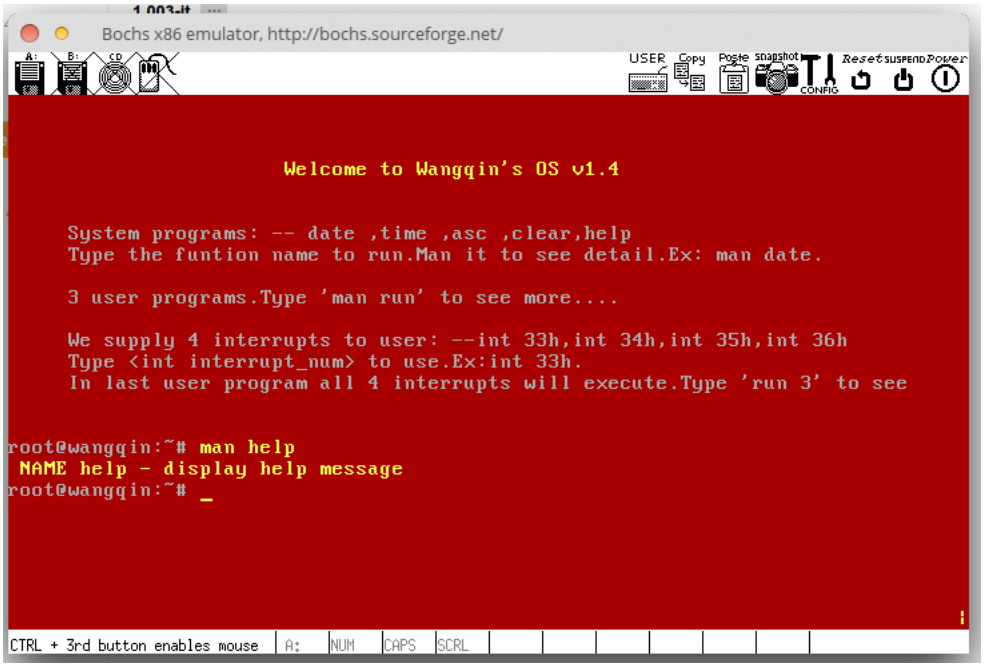
1. 编辑修改ASM 文件,和C文件

2. 使用make命令配合makefile文件进行编译

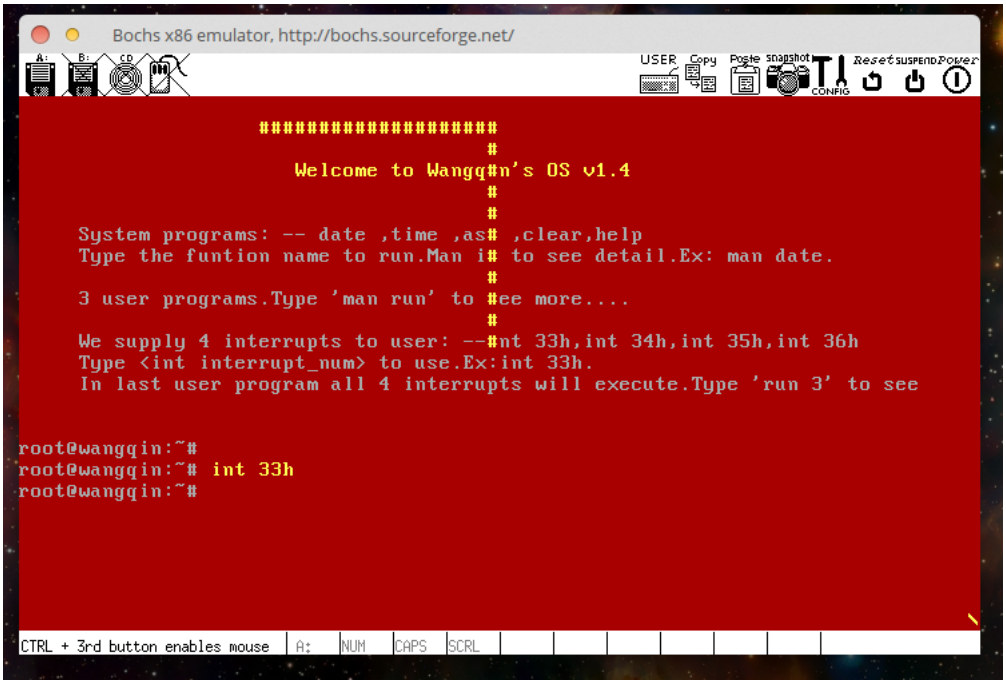
3. 运行bochs□vmware虚拟机进行测试,进入后所看到的欢迎界面



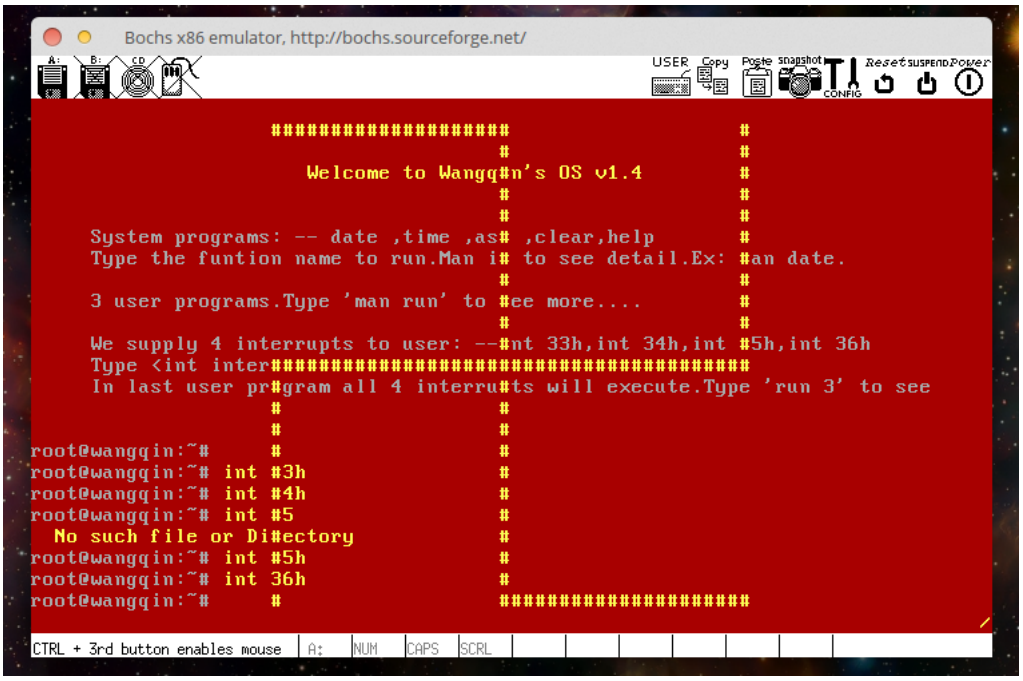
4. 这次比实验三多了一个系统调用help.我们可以执行man help查看这个调用的作用



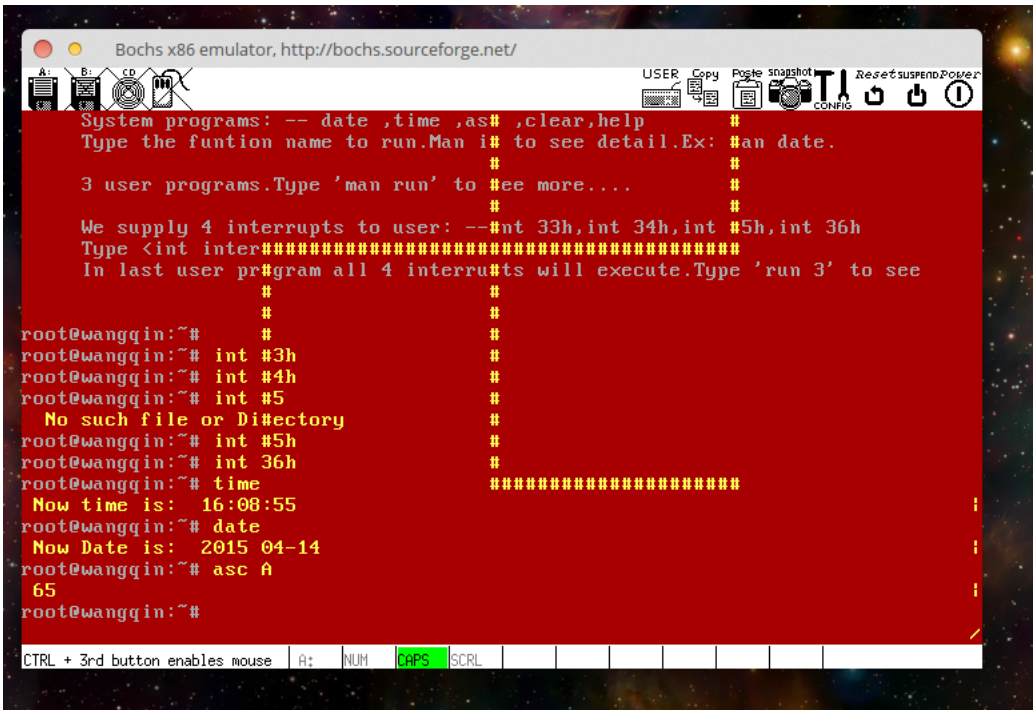
5. 输入int 33h，执行33h中断，可以看到左上角1/4屏幕打印出一个旋臂



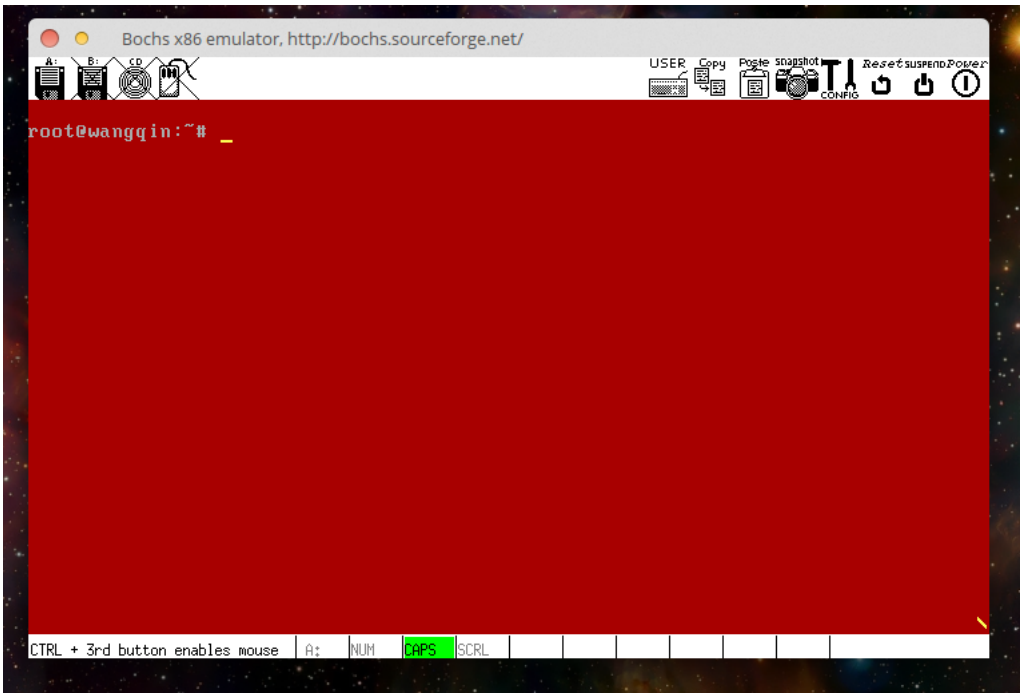
6. 继续输入`int 34h,int 35h,int 36h`，可以看到每次都在1/4屏幕上打印一个旋臂，直到生成了纳粹法西斯的标志:万字旗



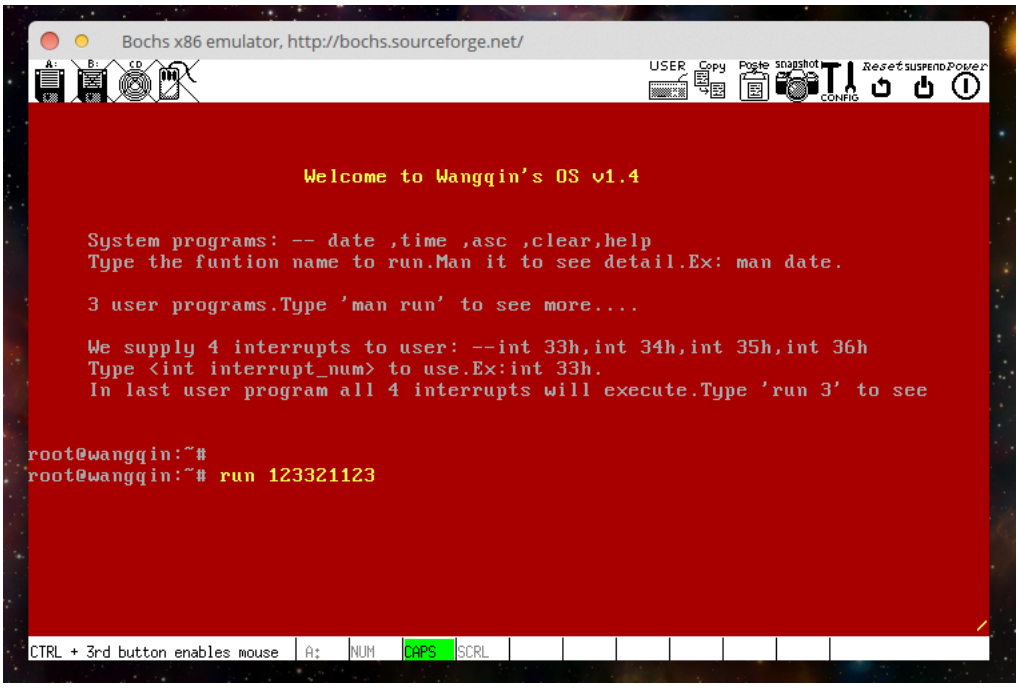
7. 继续输入几个系统调用程序的名字,date time asc..可以看到屏幕向上滚动，其实这个功能在上个实验已经实现



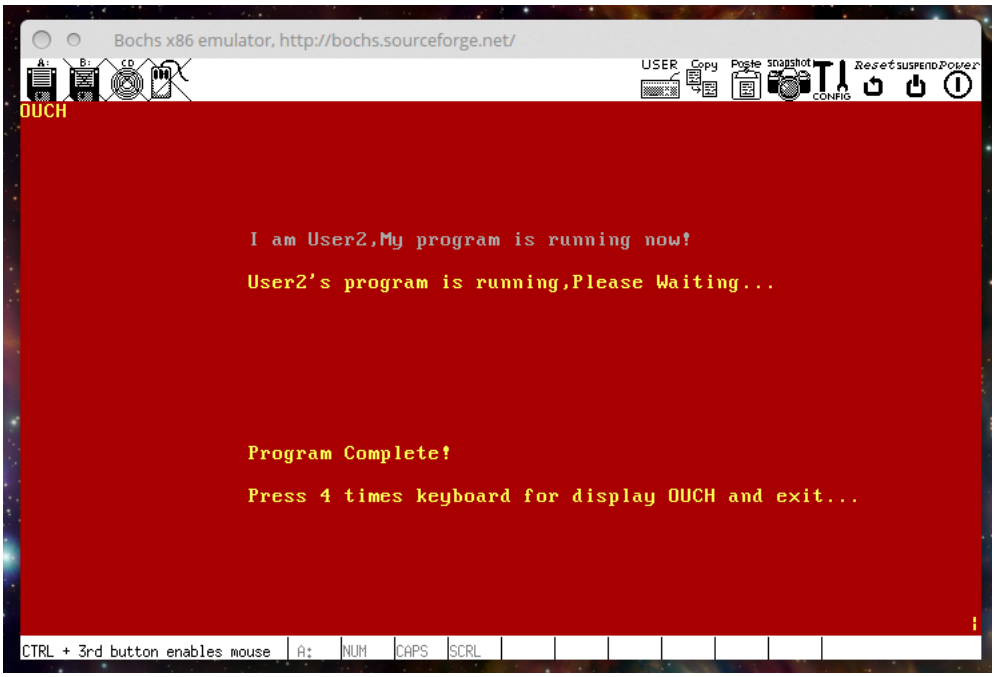
8. 现在输入clear命令清除屏幕



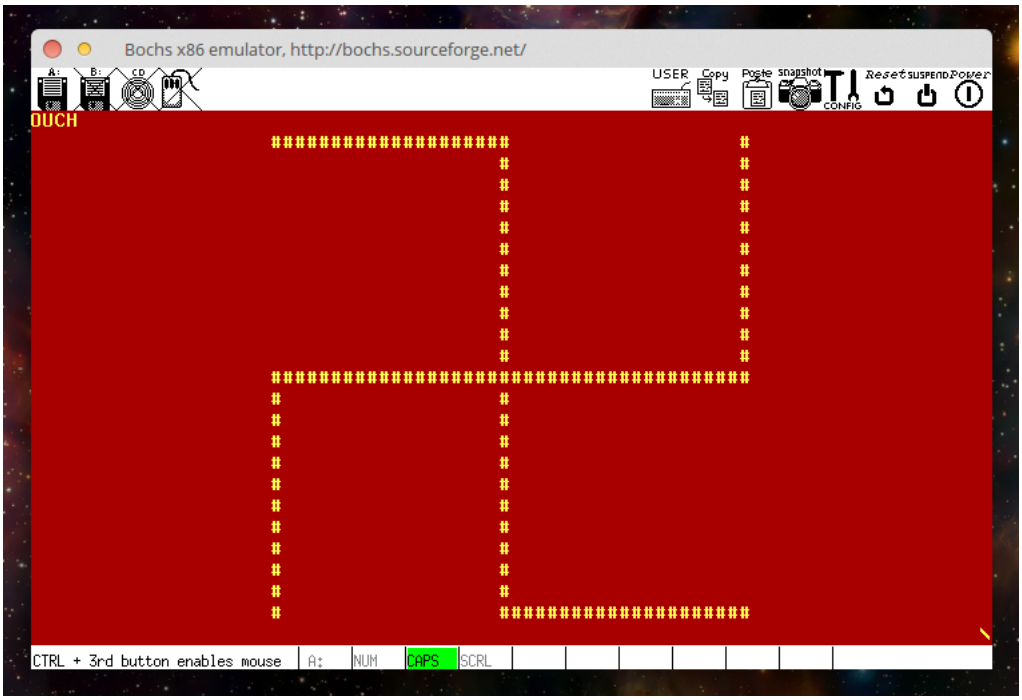
9. 输入run 语句来运行用户程序，一共有三个用户程序,一次run可以重复运行多个用户程序



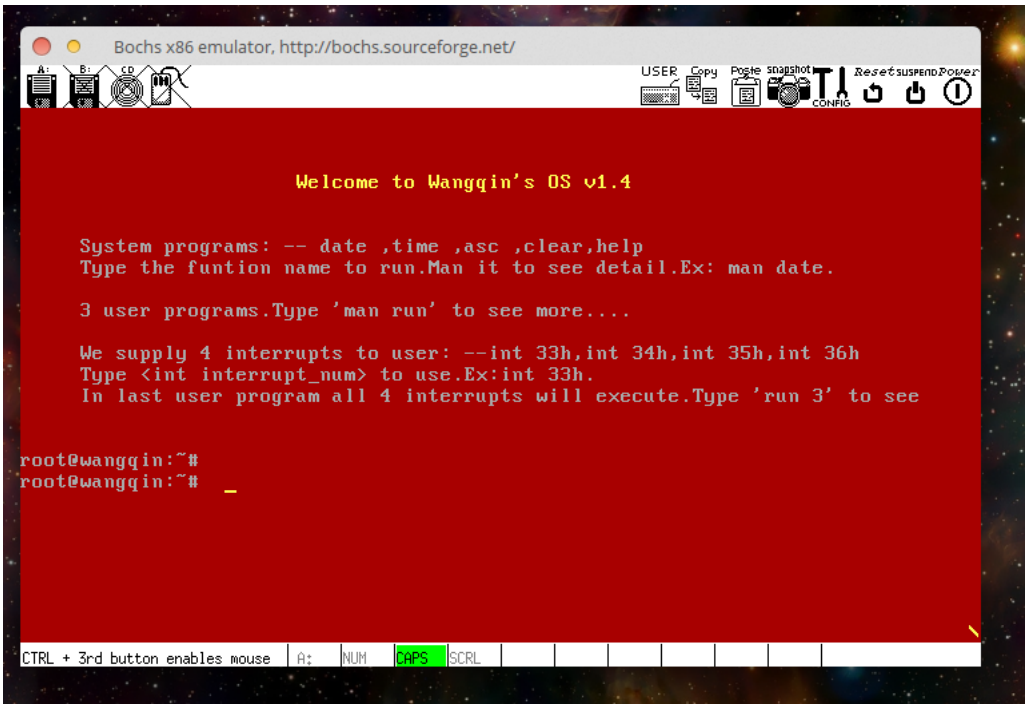
10. 进入用户程序之后键盘中断09h 就被重新修改了中断向量，只要检测到键盘敲击左上角就显示OACH。当敲击4次之后，此用户程序结束



11. 第3个用户程序，里面只有`int 33h int 34h int 35h`这几条语句，可以看到其效果就是现实一个万字旗



12. 执行完所有用户程序之后，重新返回操作系统



13. 在上述效果图中可以看到右下角始终有一个横杠在转动

内存和软盘存储管理

1. 引导程序加载到内存0x7c00处运行
2. 引导程序将操作系统加载到0x7e00处运行
3. 操作系统讲用户程序加载到0x1000处运行
4. 软盘第1个扇区存储操作系统引导程序
5. 软盘第2～15扇区存储操作系统内核
6. 软盘第16～18扇区分别存储三个用户程序

主要函数模块解释

内核架构解释：os.c为内核主要控制模块，osclib.c os.asm主要为os.c提供函数实现.oslib.asm 为osclib.c提供更底层的函数封装

1. `os.c: main` 函数模块，这个在实验三报告中已经解释这里就不在赘述

```
1
2  void main(){
3      init_ss();
4      screen_init();
5      interrupt_init();
6      print_welcome_msg();
7      print_message();
```

```
    print_flag(); //root@wangqin4377@:    position

    while(1){
        char length = listen_key();

        unsigned short int now_row = get_pointer_pos()/256;
        if( now_row >23){        // 0~24   24 is deeplist line
            while( screen_sc_T--){
                scroll_screen();
                flag_scroll_up();
            }
        }

        flag_scroll();//move flag to next line
        print_flag();
    }
}
```

2. 本次试验主要为了实现对中断向量表的修改工作，故在 os.asm中实现了下列函数为修改中断向量表提供调用

```
;insert a interrupt vector
insert_interrupt_vector:
mov ax,0
mov es,ax
mov bx,[ interrupt_num]
shl bx,2 ;interrupt num * 4 = entry
mov ax,cs
shl eax,8  ;shl 8 bit   *16
mov ax,[ interrupt_vector_offset]
mov [es:bx], eax
ret
```

3. 要修改键盘中断的中断向量的时候，要考虑用户程序执行完了再把09h中断向量修改回去。所以要几个变量来保存之前09h指向的地址另外执行用户程序的时候用户press下key的时候，按下和抬起分别会触发09h中断故要开一个变量j来区分是按下还是抬起。i表示09中断被触发几次，达到4次即4*2=8则返回内核 oslib.asm:

```

var:
i db 0       ;counter
j db 0       ;reverse every time  0101
flag db 0
duan_1 dw 0
offset_1 dw 0
```

4. 此为设置09h中断和执行用户程序的nasm代码

```
run_user:
push ecx
push ax
call screen_init

;----------------------update 09 vector
cli
mov ax,0
```

```
 9          mov es,ax
10          mov ecx,[ es:36]              ;backup
11          mov [ duan_1],ecx
12
13          mov ax,0x09
14          mov [ interrupt_num], ax
15          mov ax, key_detect
16          mov [ interrupt_vector_offset],ax
17          call insert_interrupt_vector
18          mov ax,0
19          mov [i],ax
20          sti
21          ;----------------------run
22          call 0x1000
23
24          ;----------------------reset 09 vector
25          mov ax,0
26          mov es,ax
27          mov ecx,[ duan_1]
28          mov [ es:36],ecx
29
30          pop ax
31          pop ecx
32          ret
```

5. 系统初始化设置各个中断

```
 1          interrupt_init:
 2          mov ax,cs
 3          mov ds,ax
 4
 5          ;#1  setting up time interrupt
 6          mov ax,0x1c
 7          mov [ interrupt_num], ax
 8          mov ax, print_corner
 9          mov [ interrupt_vector_offset],ax
10          call insert_interrupt_vector
11
12          ;#2 int 33
13          mov ax,0x33
14          mov [ interrupt_num], ax
15          mov ax, process_int33
16          mov [ interrupt_vector_offset],ax
17          call insert_interrupt_vector
18
19          ;#3 int 34
20          mov ax,0x34
21          mov [ interrupt_num], ax
22          mov ax, process_int34
23          mov [ interrupt_vector_offset],ax
24          call insert_interrupt_vector
25
26          ;#4 int 35
27          mov ax,0x35
28          mov [ interrupt_num], ax
```

```
29          mov ax, process_int35
30          mov [ interrupt_vector_offset],ax
31          call insert_interrupt_vector
32
33          ;#5 int 36
34          mov ax,0x36
35          mov [ interrupt_num], ax
36          mov ax, process_int36
37          mov [ interrupt_vector_offset],ax
38          call insert_interrupt_vector
39
40          ret
```

6. 循环打印’|’、’/’和’\’

```
1           print_corner:
2           ; \ / [
3             mov ax,cs
4             mov ds,ax
5             mov ax,0xb800
6             mov es,ax
7
8             mov dl,30
9             mov al,[ pointer]        ; alert:  must be al  dont ax
10            cmp al,dl
11            jne next_print_c
12            mov byte [ pointer], 0
13            jmp cotinue_corner
14
15            next_print_c:
16            mov eax, cornerstring
17            mov ebx,0
18            mov bl,[ pointer]           ;ebx is added sum
19            add eax,ebx
20            mov al,[ eax]
21
22            inc ebx
23            mov [ pointer],bl
24
25            mov bx,3998D
26            mov [ es:bx],al
27
28            cotinue_corner:
29            iret
30
31            var:
32            flag_position dd 0x1000
```

屏相关;

```
33            interrupt_num dw 0x1c                ;init
34            interrupt_vector_offset dw 0x7c00    ;init
35            pointer db 0
36            cornerstring db '\\\\\\\\\\||||||||||//////////'
```

要循环打印的字符;

```
37            ;int_user_num db 0
```

7. 显示OACH

```
key_detect:
 cli
push ax
push bx
mov ax,0xb800
mov es,ax

in al,60h
in al,61h
or al,80h
out 61h,al
mov al,61h
out 20h,al

mov bl,0
mov bh,[j]
cmp bl,bh
je change1

mov bh,0
mov [j],bh
mov byte [es:00],'O'
mov byte [es:02],'U'
mov byte [es:04],'C'
mov byte [es:06],'H'

jmp next_de

change1:
mov bh,1
mov [j],bh

mov byte [es:00],' '
mov byte [es:02],' '
mov byte [es:04],' '
mov byte [es:06],' '

next_de:

mov bl,[i]
inc bl
mov [i],bl
mov bh,8          ;press key 8 times
cmp bh,bl
jle closesti
jmp niret
closesti:
mov cl,'A'

niret:

pop bx
pop ax
sti
```

```
55              iret
```

实验心得及仍需改进之处

实验心得:

通过本次实验我了解了x86架构下中断的机制和原理，通过手动编写中断处理程序和修改添加中断向量表中的中断向量实现了本实验的所有要求。 在时钟中断的模块，刚开始打算改写08h的中断向量，后来经过网上查阅资料得知硬件中断08h会自动触发中断号为1ch的时钟中断，又叫time-tick中断。 通过改写1ch的中断处理程序即可实现时钟中断效果。

在后来的实验过程中发现有的用户程序无法读入内存，经过一番调试之后才发现自己操作系统太过庞大 占用了15个扇区，加上四个用户程序和一个引导扇区共需要19个扇区，我只好减少了一个用户程序。并且把 所有内联函数改为函数调用。最后只放了三个用户程序(其中最后一个将自动执行自定义的int 33□int 34, int 35, 36中断)。估计在后面的试验中会不断的丰富操作系统的功能， 目前我存储操作系统的 15个扇区已经有14个被写满，下次实验考虑优化操作系统的代码使其短小精悍。

在后来在编写键盘中断处理程序的过程中，在 编写中断向量号为09h的中断处理程序时由于不了解键盘中断的工作方式，明明写好了中断处理程序却只能第一次按键显示OUCH之后按键就不显示了。后来经过 查阅相关资料发现还要对相关端口61h,60h,20h 做一些处理才可以继续等待输入下个字符。

实验仍需改进之处:

仍需仔细检查多个中断发生的时候避免多个程序有无同时对同一个寄存器进行改写
可以考虑增加文件系统
没有加系统调用表
考虑精简操作系统代码，减少冗余代码
继续优化调整内核架构