操作系统实验报告

计科（大数据、人工智能方向）17341155 王永康

（实验六：具有二态进程模型的内核）

一、实验目的

初步了解操作系统的进程原理及其实现，使操作系统具有初步调度功能。

二、实验内容

保留原型原有特征的基础上，设计满足下列要求的新原型操作系统：

- (1)在c程序中定义进程表，进程数量为4个。
- (2)内核一次性加载4个用户程序运行时，采用时间片轮转调度进程运行，用户程序的输出各占1/4屏幕区域，信息输出有动感，以便观察程序是否在执行。
- (3)在原型中保证原有的系统调用服务可用。再编写1个用户程序，展示系统调用服务还能工作。

三、实验方案

操作系统实验工具与环境

- 实验支撑环境
  - 硬件：个人计算机
  - 主机操作系统：Windows10

    - 虚拟机软件：VMware,Dosbox
- 实验开发工具：
    - 汇编语言工具：x86汇编语言
    - 汇编编译工具：TASM+TCC+NASM
    - 磁盘写入工具：Winhex

实验相关原理

- 如何使程序并发执行

  利用时钟中断（8号中断），把中断的目标相应地址设为切换程序的代码起始位置，并且设置合适的中断触发间隔时间，这样就可以在执行程序1时切换去执行程序二，切换时间合适，我们人就会感觉它们在同时执行。

- 如何实现进程的切换

  程序的执行的下一条指令是由`cs:ip`决定，这就意味着改变`cs:ip`到目标代码段，就可以使程序从目前跳到去执行目标代码；

  在执行中断时，过程为：

    - （1）把`flags、cs、ip`压入当前栈，然后把`cs:ip`指向中断入口地址；
    - （2）执行中断程序
    - （3）利用`iret`指令，可以返回被中断的程序，原理是：

      把当前栈顶的三个字依次弹出赋给`ip,cs,falgs`,完成程序执行的转移。

  所以，可以利用中断的`iret`指令的特点，在调度完之后，把下一个程序的`ip,cs,falgs`压入栈顶，然后执行`iret`。

  当然，在这之前需要 进行关键两步。

    - 保护当前进程状态，即所有寄存器的保存。
    - 切换下一个进程到当前进程，下一个寄存器的值赋给当前寄存器。

四、实验过程

修改时钟中断程序，把中断的目标地址改为进程调度的入口地址

利用时钟中断号8号，修改中断向量表，8号中断向量对应8*4和8*4+2两个地址，把8号中断需要执行的部分的偏移入口地址放置在8*4中，段地址放置在8*4+2中，具体代码如下：

```
_Load_Timer proc
    push ds
    push es
    push ax
    push dx

        call SetTimer;设置时钟中断的相关参数，如中断触发间隔时间

        xor ax,ax
        mov es,ax

        cli
            mov word ptr es:[20h],offset Timer1 ; 设置时钟中断向量的偏移地址
            mov word ptr es:[22h],cs
```

```
        sti

    pop dx
    pop ax
    pop es
    pop ds
    ret
_Load_Timer endp
```

在内核建立PCB表

利用C语言编写

```
typedef struct RegisterImage{
    int SS;
    int ES;

    int DS;
    int DI;
    int SI;
    int BP;

    int SP;
    int BX;
    int DX;
    int CX;

    int AX;
    int IP;
    int CS;
    int Flags;

}RegisterImage;

typedef struct PCB{
    RegisterImage regImg;
    int status;
}PCB;

PCB PCB_Table[4];
```

进程切换——保护寄存器

首先，进入时钟中断后，首先需要保护当前进程的寄存器。

注意此时只是`cs,ip`切换成了内核的`cs,ip`,其余寄存器都还是中断前进程的值，这时候，我们需要先**切换数据段即`ds`指下内核数据段**，这样才能利用**内核的相关内存**保存当前寄存器。然后栈也是中断前的栈，暂时利用一下此时的栈，调用C函数`Save_Process()`来把寄存器的值存入PCB表。

汇编代码如下：

```
extern _Total_p:near
```

```
extern _P_begin:near
extern _save_ES:near
extern _save_DS:near
extern _save_DI:near
extern _save_SI:near
extern _save_BP:near
extern _save_SP:near
extern _save_BX:near
extern _save_DX:near
extern _save_DX:near
extern _save_CX:near
extern _save_AX:near
extern _save_SS:near
extern _save_IP:near
extern _save_CS:near
extern _save_Flags:near

Timer1:
    cli
Saves:

;;;保护进程目前情况进PCB

    push ds
    push cs
    pop ds ;;;切换数据段ds = cs，但不能改变其他寄存器的值

    pop word ptr _save_DS
    pop word ptr _save_IP
    pop word ptr _save_CS
    pop word ptr _save_Flags


    mov word ptr _save_BX,bx
    mov word ptr _save_AX,ax
    mov word ptr _save_DI,di
    mov word ptr _save_SI,si

    mov word ptr _save_BP,bp
    mov word ptr _save_DX,dx
    mov word ptr _save_CX,cx
    mov word ptr _save_SS,ss

    mov word ptr _save_SP,sp
    mov word ptr _save_ES,es

    call near ptr _Save_Process
```

C的相关代码：

```
int save_ES;
int save_DS;
int save_DI;
```

```
int save_SI;

int save_BP;
int save_SP;
int save_BX;
int save_DX;

int save_CX;
int save_AX;
int save_SS;
int save_IP;

int save_CS;
int save_Flags;

void Save_Process()
{
    PCB_Table[Now_process].regImg.SS = save_SS;
    PCB_Table[Now_process].regImg.ES = save_ES;
    PCB_Table[Now_process].regImg.DS = save_DS;
    PCB_Table[Now_process].regImg.DI = save_DI;

    PCB_Table[Now_process].regImg.SI = save_SI;
    PCB_Table[Now_process].regImg.BP = save_BP;
    PCB_Table[Now_process].regImg.SP = save_SP;
    PCB_Table[Now_process].regImg.BX = save_BX;

    PCB_Table[Now_process].regImg.DX = save_DX;
    PCB_Table[Now_process].regImg.CX = save_CX;
    PCB_Table[Now_process].regImg.AX = save_AX;
    PCB_Table[Now_process].regImg.IP = save_IP;

    PCB_Table[Now_process].regImg.CS = save_CS;
    PCB_Table[Now_process].regImg.Flags = save_Flags;

}
```

进程切换——调度与切换寄存器

首先，调用C函数`void Sheduler()`进行调度，即把下一个进程的寄存器值赋给用来传递参数的全局变量值。本实验只采用简单的进程调度模式，即**顺序轮转。**

然后切换当前寄存器至下一个进程的寄存器。注意，**ds寄存器**必须**最后更改**，因为这个寄存器的值改变意味着切换数据段，这样会使内核的全局变量就无法正确访问。

在这之前，还需要**切换堆栈**为**下一个进程的堆栈**，并且把`flags,cs,ip`压入栈顶。

汇编代码如下：

```
Restarts:

    call _Scheduler

```

```
    ;切换堆栈
    mov ss,word ptr _save_SS
    mov sp,word ptr _save_SP

    mov di,word ptr _save_DI
    mov si,word ptr _save_SI
    mov bp,word ptr _save_BP
    mov bx,word ptr _save_BX

    mov dx,word ptr _save_DX
    mov cx,word ptr _save_CX
    mov ax,word ptr _save_AX
    mov es,word ptr _save_ES

    ;关键一步，压入flags、cs、ip进堆栈
    push word ptr _save_Flags
    push word ptr _save_CS
    push word ptr _save_IP

    ;;最后切换数据段
    push word ptr _save_DS
    pop ds
```

`void Shecduler()` 的C代码:

```
void Scheduler()
{
    while(1){
        Now_process++;
        if(Now_process>=Total_p) Now_process = 0;
        if(PCB_Table[Now_process].status != _DEL) break;
    }

    save_AX = PCB_Table[Now_process].regImg.AX;
    save_BX = PCB_Table[Now_process].regImg.BX;
    save_CS = PCB_Table[Now_process].regImg.CS;
    save_CX = PCB_Table[Now_process].regImg.CX;

    save_DI = PCB_Table[Now_process].regImg.DI;
    save_DS = PCB_Table[Now_process].regImg.DS;
    save_DX = PCB_Table[Now_process].regImg.DX;
    save_ES = PCB_Table[Now_process].regImg.ES;

    save_Flags = PCB_Table[Now_process].regImg.Flags;
    save_IP = PCB_Table[Now_process].regImg.IP;
    save_SS = PCB_Table[Now_process].regImg.SS;
    save_SP = PCB_Table[Now_process].regImg.SP;

    save_ES = PCB_Table[Now_process].regImg.ES;
    save_Flags = PCB_Table[Now_process].regImg.Flags;
}
```

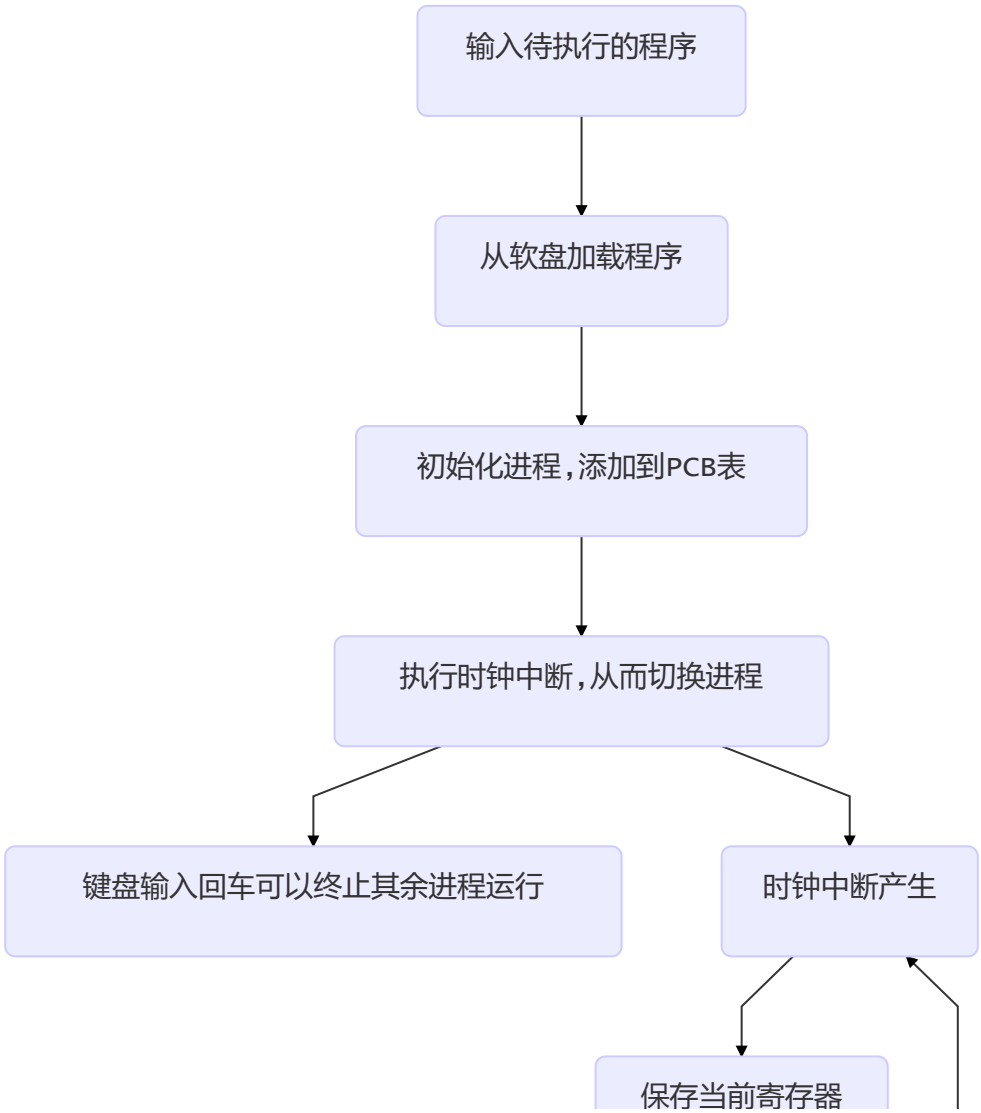
切换进程——iret指令跳转执行下一进程

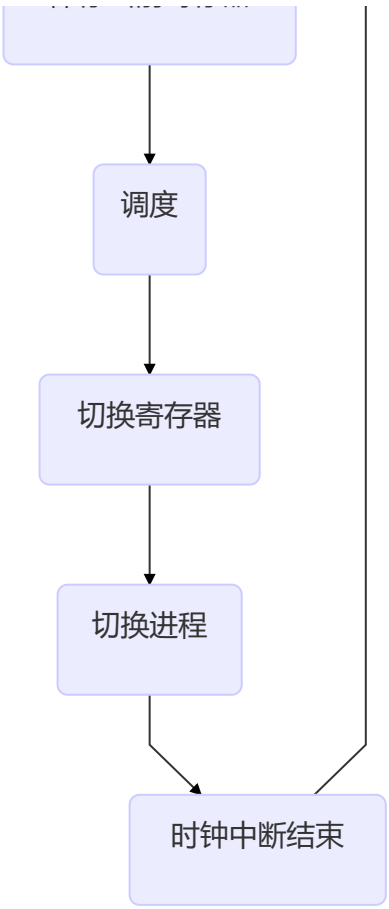
上面的操作已经做好了切换的一切准备，接下来就是进入下一个进程执行了。结束时钟中断，利用`iret`指令，把`flags,cs,ip`置为下一个进程的执行的代码地址，从而完成切换跳转，于是切换完成。

汇编代码如下：

```
donothing:
    push ax

    mov al,20h
    out 20h,al
    out 0A0h,al

    pop ax
    sti
    iret
```

程序流程





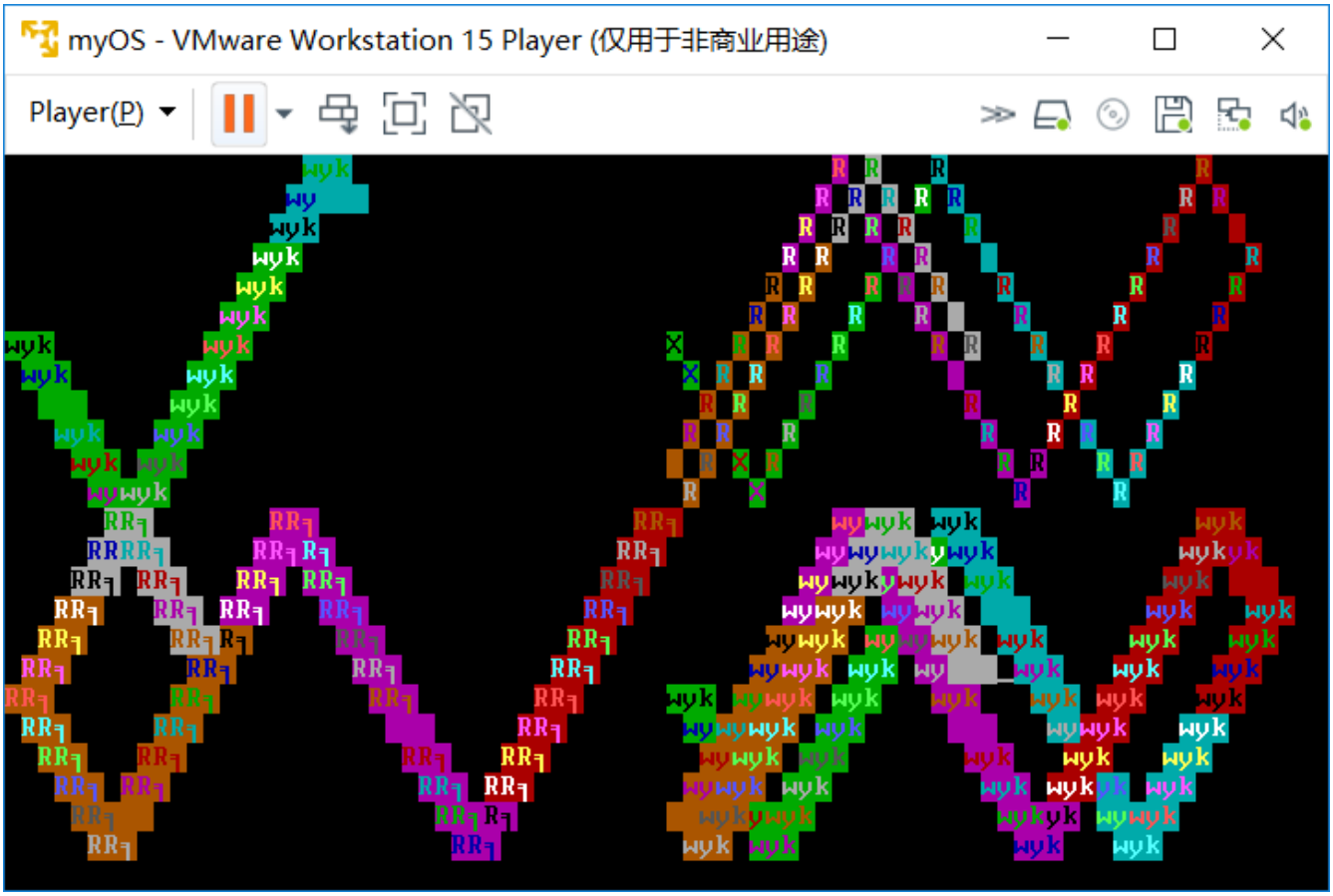
五、实验结果

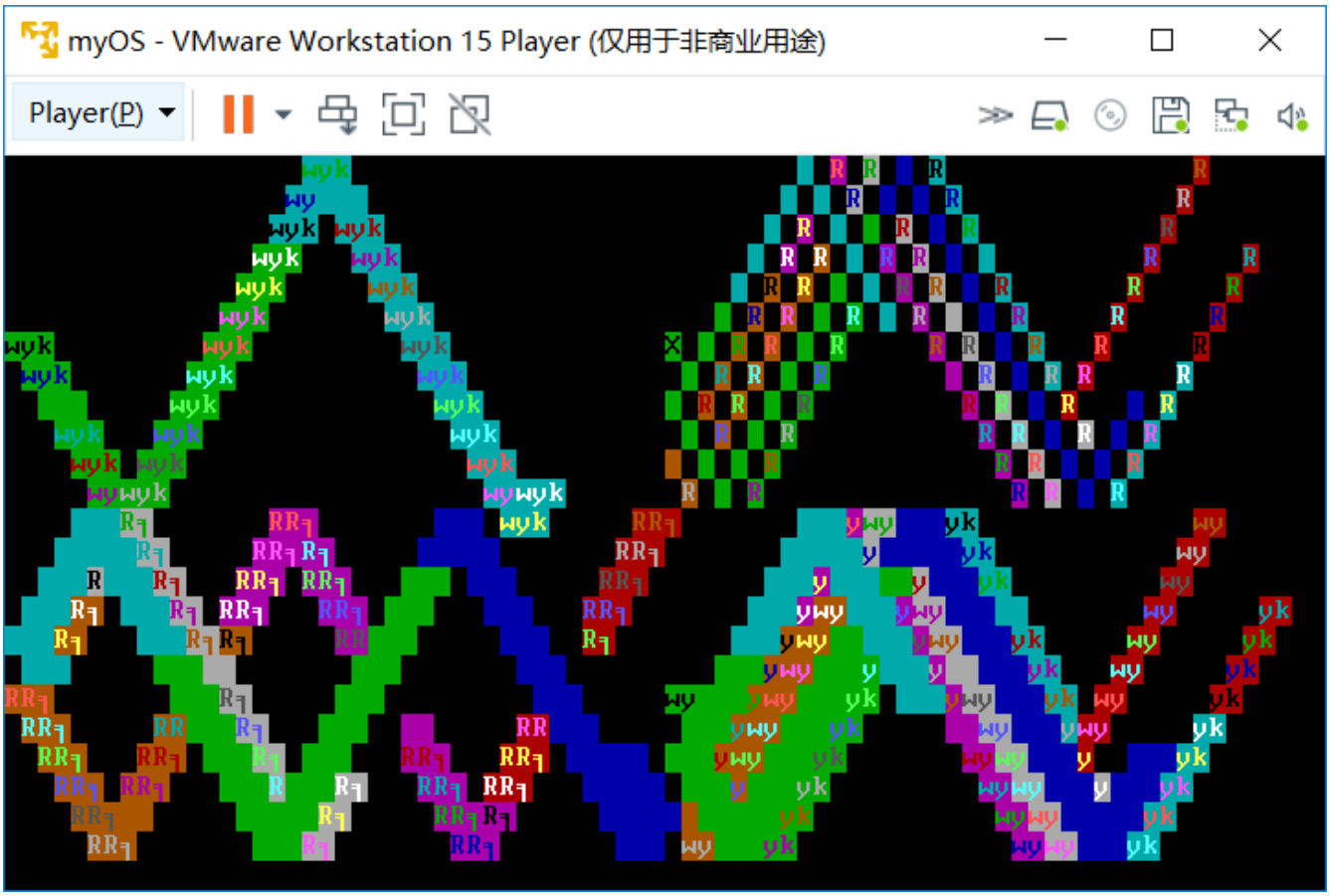
在终端输入

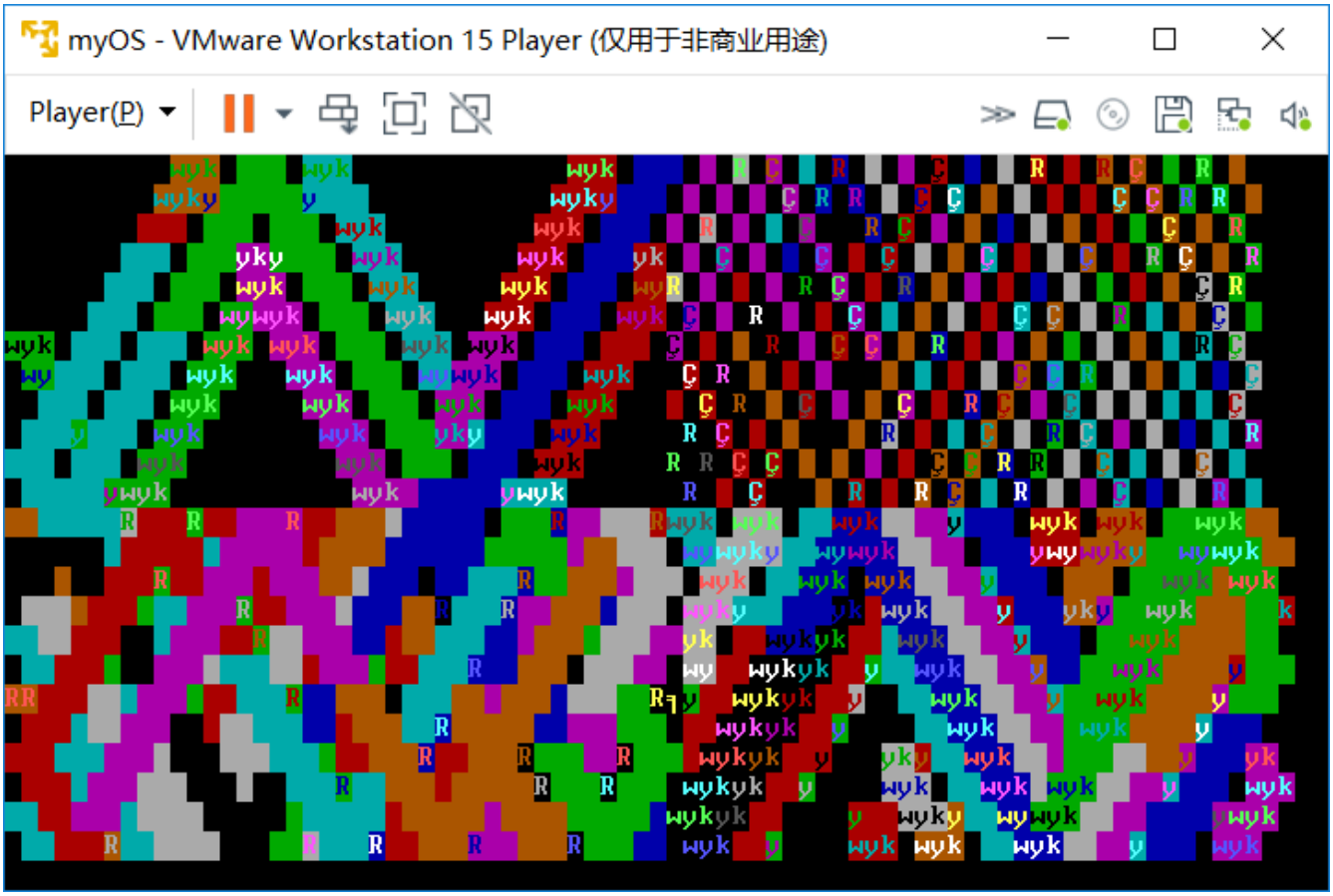
```
>run
 1,2,3,4
```

（支持四个及以下程序输入，可以随便输入1-4之间的数，可以输入多个如：1,2或2，3或2，4，3等等）

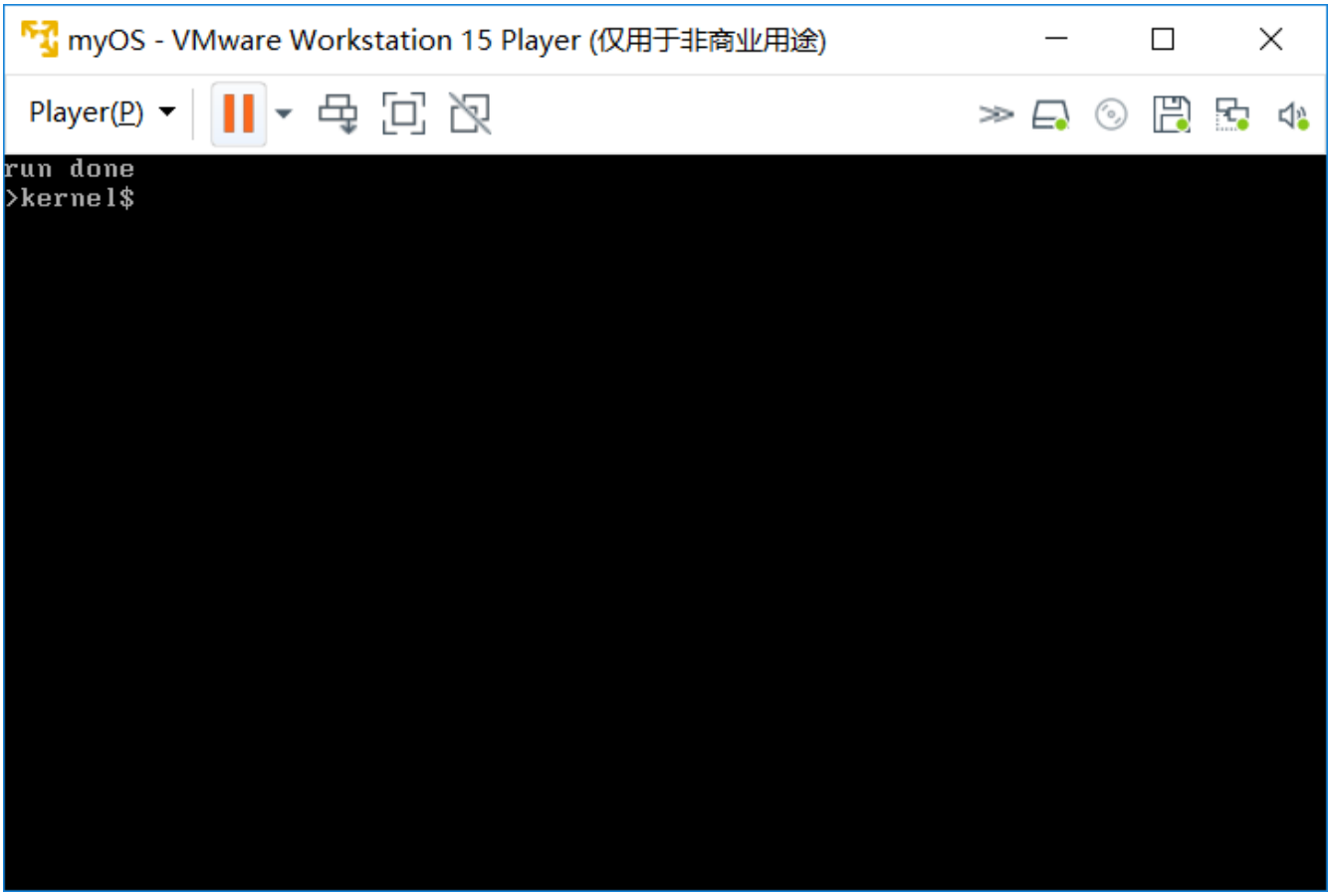
执行结果：


myOS - VMware Workstation 15 Player (仅用于非商业用途)
Player(P)


myOS - VMware Workstation 15 Player (仅用于非商业用途)
Player(P)



输入回车，结束所有进程执行，结果如下：

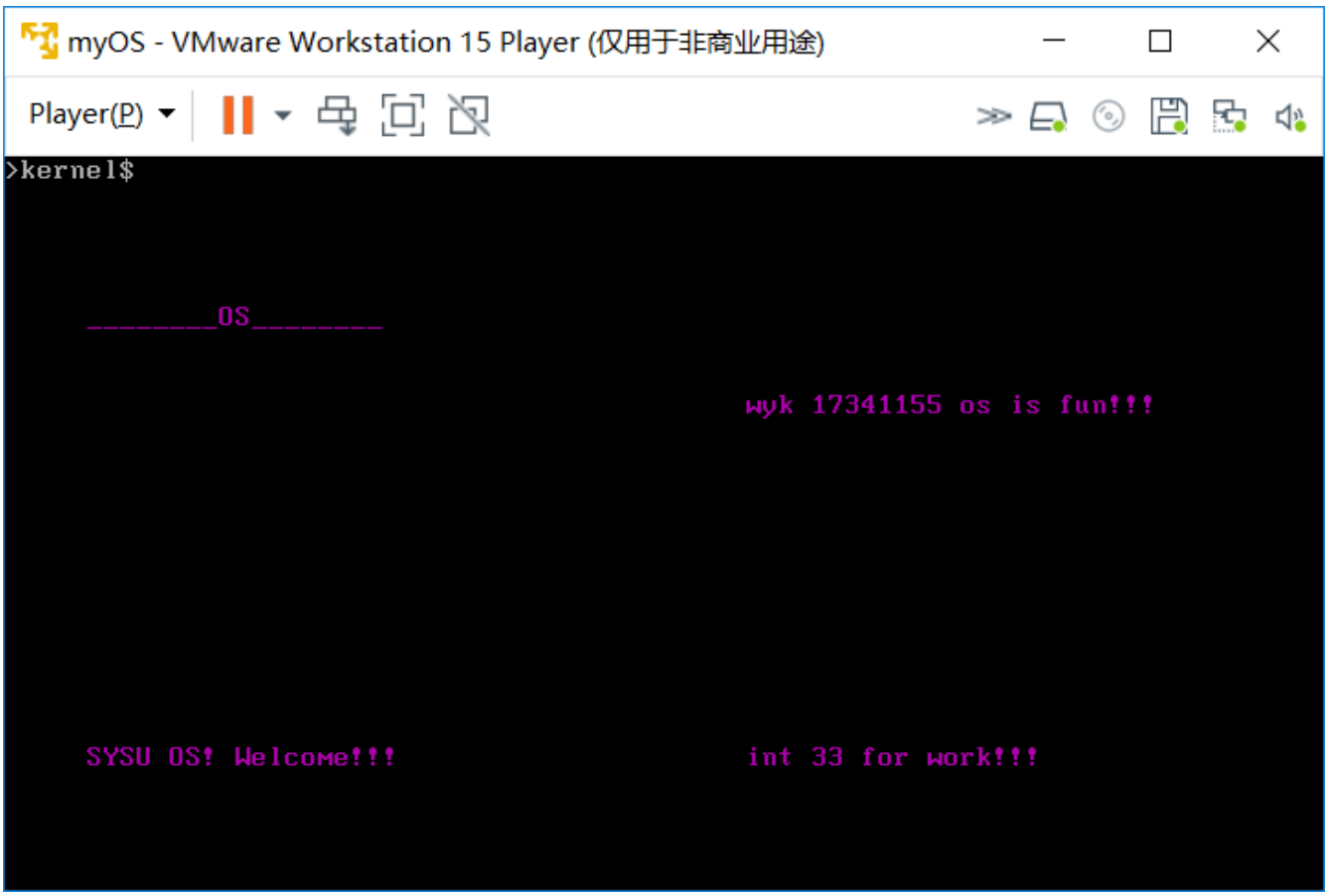


展示系统调用，int 21h

键盘输入int33，即可执行，int33程序的测试代码如下：

```
org 0A100h
start:

    mov ax,cs
    mov ds,ax
    mov es,ax

    mov ah,0
    int 21h

    mov ah,1
    int 21h

    mov ah,2
    int 21h

    mov ah,3
    int 21h


    ret
```

运行结果：



所有的文件如下：

> boot
>
> > myOS.asm
>
> kernel
>
> > kernel.asm
> >
> > > system.asm Sys.h
> > >
> > > klib.asm Stdio.h
> >
> > cmain.c
>
> 用户程序
>
> > pro1.asm
> >
> > pro2.asm
> >
> > pro3.asm
> >
> > pro4.asm
> >
> > int33_test.asm

六、实验总结

这一次多进程的操作系统的实验有些挑战性，也确实涉及到非常多知识，而且涉及到非常多的细节，这使得这一次实验着实具有不小难度，总结起来感触有以下两点：

- **更加深入理解了cs、ip、ss、sp、ds这些寄存器的功能**

  这个多进程的实验，理解起来非常关键的就是要清楚这些寄存器的功能，还要iret等指令的操作，其中ss,sp两个指向堆栈的寄存器，非常重要，保证这一对寄存器的正确取值直接影响了个人这一次实验的成败，开始时，一直无法并发执行用户程序，就是因为ss:sp的指向错误。保证栈的正确性，其次保证cs:ip的正确指向，是本次实验的关键。

- **对x86体系架构有了进一步理解**

  x86体系的指令执行取决于cs:ip,但是却不能通过mov指令修改cs,ip,只能利用跳转jmp,iret等指令来修改。本实验中，就非常巧妙的利用iret指令的特点，进行了进程的切换；还要一点就是数据段ds的问题，数据的访问的基地址取决于ds,所以保证正确的ds就非常关键了，在内核的保存寄存器进PCB表时，可以说这个过程的设计还是颇费一番心思的。

参考文献

[1] 王爽.《汇编语言(第3版)》[M].清华大学出版社，2013-9

[2] 凌应标. “06实验课.ppt”，中山大学计算机科学系，2015-3