

# 实验五：二状态的进程模型

实验者：张海涛，张晗宇，张浩然，张镓伟  
学号：15352405, 15352406, 15352407, 15352408  
院系：数据科学与计算机学院  
专业：15 级软件工程（移动信息工程）  
指导老师：凌应标

## 【实验题目】

在设计的操作系统中实现二状态的进程模型

## 【实验目的】

1. 设计实现进程控制块与进程表
2. 在 myos 操作系统实现二状态的进程模型
3. 理解交替执行原理，利用时钟中断实现用户程序的轮流执行

## 【实验要求】

保持原型原有的基础特征的基础上，设计满足以下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. 修改内核，使内核一次性加载 4 个用户程序运行，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，展示你的所有系统调用服务还能工作。

## 【实验方案】

### 一. 虚拟机配置方法

无操作系统，10M 硬盘，4MB 内存，启动时连接软盘

### 二. 软件工具和作用

Notepad++: 用于生成汇编语言文件

Nasm: 用于编译.asm 类型的汇编语言文件，生成 bin 文件

Vmware Workstation 12Player: 用于创建虚拟机，模拟裸机环境

TCC: 用于由.c 文件生成.obj 文件

TNASM: 用于由.asm 文件生成.obj 文件。

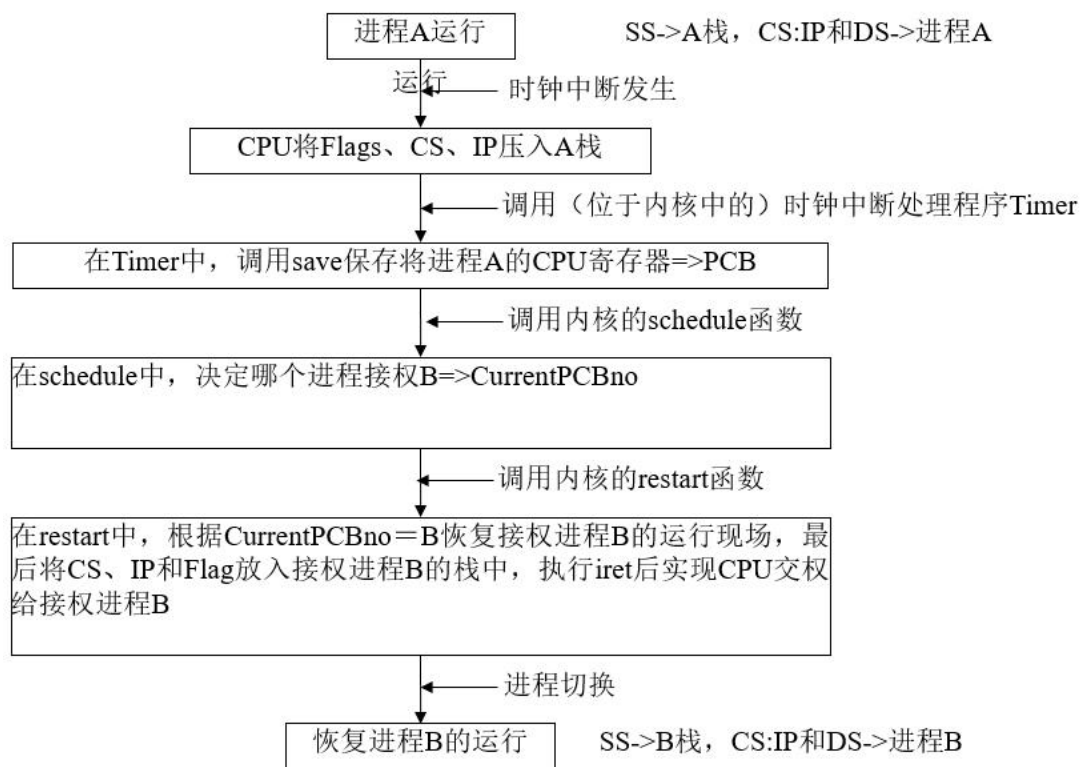
Tlink: 用于链接 obj 文件生产.com 可执行文件

### 三. 实验原理

#### 1. 进程表 (PCB)

初级的进程模型可以理解为将一个物理的 CPU 模拟成多个逻辑独立的 CPU。每个进程具有一个独立的计算环境。而要在同一台计算机中并发地执行多个不同的用户程序，操作系统要保证独立的用户程序之间不会相互干扰。为此需要在内核中建立：进程表和进程控制块 PCB。其中包含：进程标识，内存分配信息和逻辑 CPU 模拟。

#### 2. 时钟中断服务程序原理示意



### 3.进程交替执行原理

在之前的原型操作系统，是顺序执行用户程序，内存中不会有两个不同的用户程序，所以 CPU 的控制权交接问题简单。而现在内存中会存放多个用户程序，需要采用时钟中断打断执行中的用户程序，实现 CPU 在进程之间的交替执行。

在系统启动时，将把需要并发执行的用户程序加载到内存中，并建立相应的 PCB，利用时钟中断实现用户程序的轮流值执行。在这个过程中，最关键的地方在于：保护现场和进程切换。

为了及时保护中断现场，必须在中断处理函数的最开始处，立即保存被中断程序的所有上下文寄存器中的当前值。不能先进行栈切换，再来保存寄存器。因为切换栈所需的若干指令，会破坏寄存器的当前值。这正是我们在中断处理函数的开始处，安排代码保存寄存器的内容。我们 PCB 中的 16 个寄存器值，内核一个专门的程序 save，负责保护被中断的进程的现场，将这些寄存器的值转移至当前进程的 PCB 中。

针对进程切换解决办法有三个，一个是所有程序，包括内核和各个应用程序进程，都使用共同的栈。即它们共享一个（大栈段）SS，但是可以有各自不同区段的 SP，可以做到互不干扰，也能够用 IRET 进行进程切换。第二种方法，是不使用 IRET 指令，而是改用 RETF 指令，但必须自己恢复 FLAGS 和 SS。第三种方法，使用 IRET 指令，在用户进程的栈中保存 IP、CS 和 FLAGS，但必须将 IP、CS 和 FLAGS 放回用户进程栈中，这也是我们程序所采用的方案。

## 四. 程序功能

这次实验原本我们想在我们上一次的代码中加入新代码以实现二状态进程功能，但是尝试了很久，用户程序的显示始终不见踪影。最后我们仔细研究了一下老师所给的原型代码，发现它只是缺少用户程序而已，其他功能已经很完善了，于是我们决定将我们的用户程序移植到原型代码上面，然后再进行完善。

其中我们保留了原型代码的以下功能：

- 1.显示时间
- 2.显示日期
- 3.求一个字符的 ascii 码
- 4.清屏
- 5.进程交替执行
6. 21 号中断功能，包括：
  - a.在屏幕中央显示“OUCH!OUCH!”
  - b.英文大小写转换
  - c.字符串和数字之间的转换
  - d.在屏幕任意位置显示一个字符串
  - e.二进制转十进制
  - f.在屏幕任意位置显示一个彩色字符串
  - g.判断两个字符串是否相等。

我们做的完善和扩展：

- 1.原型代码在内核界面的右下角动态显示‘\’、‘|’、‘/’、‘—’。我们改成在四个角都显示，然后四个角再分别多显示“WE”“LOVE”“OPERATING”“SYSTEM”。以上显示再加上动态变色显示
2. 原型代码中，输入命令的时候，按退格键仅仅只是光标回退，却没有将字符删去。也就是说假如我想输入“pro”，不小心输了个“pri”，按一下回退再输入“o”，内存中储存的字符串时“pri\bo”，造成了非法输入。我们对此作了修改，使得它能正常回退删除。
3. 原型代码中，求一个 ascii 码这个功能不完善，它只是默认求 ascii 码为两位数的字符的 ascii 码，假如输入一个 ascii 码为三位数的字符如‘z’，求出来的结果就是错的，于是我们将求 ascii 码这部分按照考虑一位数，两位数，三位数三种情况的方式去编写，修复了这个 bug。
4. 原型代码中屏幕显示到底部的时候会跟分时中断变色显示的字符起冲突。为了不让它们起冲突，我在编写代码让它每接收三次输入，就清一次屏。
5. 进程交替执行功能中，我们接入了 5 个上次实验的用户程序，其中第 5 个是用来测试 33h,34h,35h,36h 这 4 个中断的测试程序。由于我们原本第四个用户程序是显示日期，与原型代码已有的功能重复，所以我们将第 4 个程序修改成字符弹射，但是方向与第 1 个用户程序的相反，这样当同行运行 1、4 两个程序的时候就会显示一个对称的弹射，看上去有对称美。

## 五.程序关键代码及解释

完整的时间中断程序实现的功能是进程间切换时对于进程控制块表的保存和回复。本次实验中这个功能是使用 c 和汇编混合编写。

在 PC 中，时间中断由 8259A 芯片控制。它是可以用程序控制的，专门针对 8085A 和 8086/8088 进行中断控制的芯片。我们使用 IN 和 OUT 指令来对该芯片进行初始化编程。

```
SetTimer:
push ax
mov al,34h ; 设控制字值
out 43h,al ; 写控制字到控制字寄存器
mov ax,29830 ; 每秒 20 次中断（50ms 一次）
out 40h,al ; 写计数器 0 的低字节
mov al,ah ; AL=AH
out 40h,al ; 写计数器 0 的高字节
pop ax
ret
```

这次的时间轮转运行各个用户程序的过程是，在每次时间中断发生时，将当前进程的相关信息保存在一个数据结构 PCB 中。然后改变当前要运行的程序，后面又想运行上一个程序的时候，就从 PCB 中将上一次运行的信息取出来继续运行。那么这一个过程就涉及到几个过程:加载用户程序、创建 PCB 表、存储当前进程、恢复进程。代码如下：

random\_load 的作用是处理我们输入的指令，加载指定用户程序。

```
void Random_Load()
{
    int i = 0;
    int j = 0;
    for( i=0; i<StringLen;i++ )
        if( Buffer[i]!=' ' && (Buffer[i]<'1' || Buffer[i]>'5') )
        {
            Print("Error Input!");
            return;
        }
    for( i=0; i<StringLen;i++ )
    {
        if( Buffer[i] == ' ' )
            continue;
        else
        {
            j = Buffer[i] - '0';
            if( Segment > 0x6000 )
            {
                Print("There have been 5 Processes !");
                break;
            }
            another_load(Segment,j*2);
            Segment += 0x1000;
            Program_Num ++;
        }
    }
}
```

another\_load 的作用就是读取扇区并将指定程序加载到对应地址

```
public _another_load
_another_load proc
    push ax
    push bp

    mov bp,sp

    mov ax,[bp+6]    ;段地址 ; 存放数据的内存基地址
    mov es,ax        ;设置段地址 (不能直接mov es,段地址)
    mov bx,100h      ;偏移地址; 存放数据的内存偏移地址
    mov ah,2         ;功能号
    mov al,1         ;扇区数
    mov dl,0         ;驱动器号 ; 软盘为0, 硬盘和U盘为80H
    mov dh,1         ;磁头号 ; 起始编号为0
    mov ch,0         ;柱面号 ; 起始编号为0
    mov cl,[bp+8]    ;起始扇区号 ; 起始编号为1
    int 13H          ;调用中断

    pop bp
    pop ax

    ret
_another_load endp
```

下面是 PCB 相关的 c 代码：

```

typedef struct RegisterImage{
    int SS;
    int GS;
    int FS;
    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 Process_Status;
}PCB;

```

```

PCB pcb_list[8];
int CurrentPCBno = 0;
int Program_Num = 0;

extern void printChar();

PCB* Current_Process();
void Save_Process(int,int, int, int, int, int, int, int,
    int,int,int,int, int,int, int,int );
void init(PCB*, int, int);
void Schedule();
void special();

```

```

void init(PCB* pcb,int segment, int offset)
{
    pcb->regImg.GS = 0xb800;
    pcb->regImg.SS = segment;
    pcb->regImg.ES = segment;
    pcb->regImg.DS = segment;
    pcb->regImg.CS = segment;
    pcb->regImg.FS = segment;
    pcb->regImg.IP = offset;
    pcb->regImg.SP = offset - 4;
    pcb->regImg.AX = 0;
    pcb->regImg.BX = 0;
    pcb->regImg.CX = 0;
    pcb->regImg.DX = 0;
    pcb->regImg.DI = 0;
    pcb->regImg.SI = 0;
    pcb->regImg.BP = 0;
    pcb->regImg.FLAGS = 512;
    pcb->Process_Status = NEW;
}

void special()
{
    if(pcb_list[CurrentPCBno].Process_Status==NEW)
        pcb_list[CurrentPCBno].Process_Status=RUNNING;
}

```

Save 相关部分:

```

Pro_Timer:
*****
*          Save          *
*****
    cmp word ptr[_Program_Num],0
    jnz Save
    jmp No_Progress
Save:
    inc word ptr[Finite]
    cmp word ptr[Finite],1600
    jnz Lee
    mov word ptr[_CurrentPCBno],0
    mov word ptr[Finite],0
    mov word ptr[_Program_Num],0
    mov word ptr[_Segment],2000h
    jmp Pre
Lee:
    push ss
    push ax
    push bx
    push cx
    push dx
    push sp
    push bp
    push si
    push di
    push ds
    push es
    .386
    push fs
    push gs
    .8086

```

```

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

    call near ptr _Save_Process
    call near ptr _Schedule

Pre:
    mov ax, cs
    mov ds, ax
    mov es, ax

    call near ptr _Current_Process
    mov bp, ax

    mov ss,word ptr ds:[bp+0]
    mov sp,word ptr ds:[bp+16]

    cmp word ptr ds:[bp+32],0
    jnz No_First_Time

```

```

void Save_Process(int gs,int fs,int es,int ds,int di,int si,int bp,
                 int sp,int dx,int cx,int bx,int ax,int ss,int ip,int cs,int flags)
{
    pcb_list[CurrentPCBno].regImg.AX = ax;
    pcb_list[CurrentPCBno].regImg.BX = bx;
    pcb_list[CurrentPCBno].regImg.CX = cx;
    pcb_list[CurrentPCBno].regImg.DX = dx;

    pcb_list[CurrentPCBno].regImg.DS = ds;
    pcb_list[CurrentPCBno].regImg.ES = es;
    pcb_list[CurrentPCBno].regImg.FS = fs;
    pcb_list[CurrentPCBno].regImg.GS = gs;
    pcb_list[CurrentPCBno].regImg.SS = ss;

    pcb_list[CurrentPCBno].regImg.IP = ip;
    pcb_list[CurrentPCBno].regImg.CS = cs;
    pcb_list[CurrentPCBno].regImg.FLAGS = flags;

    pcb_list[CurrentPCBno].regImg.DI = di;
    pcb_list[CurrentPCBno].regImg.SI = si;
    pcb_list[CurrentPCBno].regImg.SP = sp;
    pcb_list[CurrentPCBno].regImg.BP = bp;
}

```

恢复的相关部分:

<pre> ***** Restart ***** Restart:     call near ptr _special      push word ptr ds:[bp+30]     push word ptr ds:[bp+28]     push word ptr ds:[bp+26]      push word ptr ds:[bp+2]     push word ptr ds:[bp+4]     push word ptr ds:[bp+6]     push word ptr ds:[bp+8]     push word ptr ds:[bp+10]     push word ptr ds:[bp+12]     push word ptr ds:[bp+14]     push word ptr ds:[bp+18]     push word ptr ds:[bp+20]     push word ptr ds:[bp+22]     push word ptr ds:[bp+24]      pop ax     pop cx     pop dx     pop bx     pop bp </pre>	<pre>     pop si     pop di     pop ds     pop es     .386     pop fs     pop gs     .8086      push ax     mov al,20h     out 20h,al     out 0A0h,al     pop ax     iret  No_First_Time:     add sp,16     jmp Restart  No_Progress:     call another_Timer      push ax     mov al,20h     out 20h,al     out 0A0h,al     pop ax     iret </pre>
--	--

接下来是说一下我们的改动部分,原型代码中退格功能与修改后的对比,左边是原型的,右边是我们的,就是对输入为退格,即'\b'做一个判断:

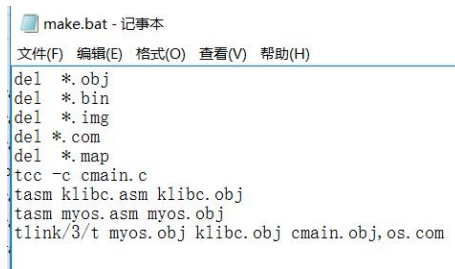
<pre> 1 2 3 4 5 void GetInput() 6 { 7     index = 0; 8     getChar(); 9     printChar(input); 10    while( input != '\n' &amp;&amp; input != '\r' ) 11    { 12        Buffer[index] = input; 13        index ++; 14        getChar(); 15        printChar(input); 16    } 17    Buffer[index] = '\0'; 18    Print("\n"); 19    StringLen = index; 20 } 21 </pre>	<pre> void GetInput() {     index = 0;     getChar();     printChar(input);     while( input != '\n' &amp;&amp; input != '\r' ){         if(input == '\b'){             if(index &gt; 0){                 index--;                 Print("\b");             }             getChar();             if(input == '\b' &amp;&amp; index) Print("\b");             else if (input != '\b') printChar(input);         }         else{             Buffer[index++] = input;             getChar();             printChar(input);         }     }     Buffer[index] = '\0';     Print("\n");     StringLen = index; } </pre>
--	---



## 六.编译.

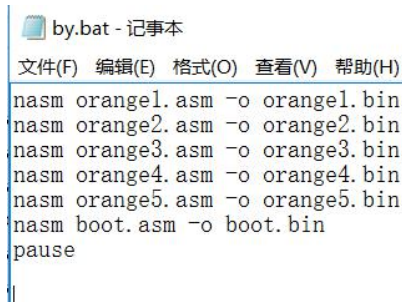
跟上一次实验一样，使用 **bat** 批处理文件进行自动化编译。

make.bat:



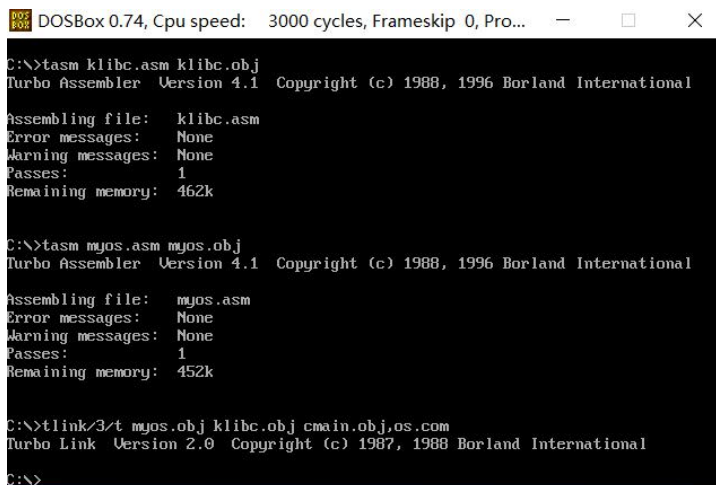
```
make.bat - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
del *.obj
del *.bin
del *.img
del *.com
del *.map
tcc -c cmain.c
tasm klibc.asm klibc.obj
tasm myos.asm myos.obj
tlink/3/t myos.obj klibc.obj cmain.obj,os.com
```

by.bat:



```
by.bat - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
nasm orange1.asm -o orange1.bin
nasm orange2.asm -o orange2.bin
nasm orange3.asm -o orange3.bin
nasm orange4.asm -o orange4.bin
nasm orange5.asm -o orange5.bin
nasm boot.asm -o boot.bin
pause
```

在 DOSBOX 中运行 make:

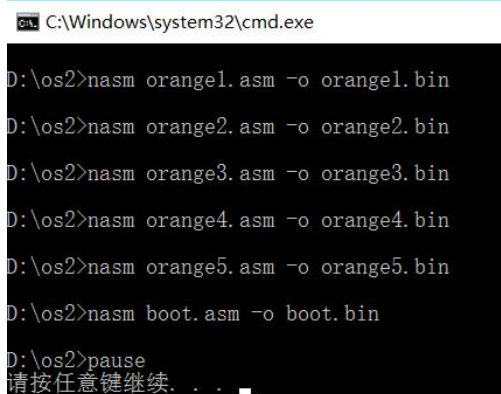


```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Pro...
C:\>tasm klibc.asm klibc.obj
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International
Assembling file: klibc.asm
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 462k

C:\>tasm myos.asm myos.obj
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International
Assembling file: myos.asm
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 452k

C:\>tlink/3/t myos.obj klibc.obj cmain.obj,os.com
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International
C:\>
```

在本机运行 by.bat:



```
C:\Windows\system32\cmd.exe
D:\os2>nasm orange1.asm -o orange1.bin
D:\os2>nasm orange2.asm -o orange2.bin
D:\os2>nasm orange3.asm -o orange3.bin
D:\os2>nasm orange4.asm -o orange4.bin
D:\os2>nasm orange5.asm -o orange5.bin
D:\os2>nasm boot.asm -o boot.bin
D:\os2>pause
请按任意键继续. . .
```

最后使用 winhex 将 boot.bin、OS.COM、orange1.bin、orange2.bin、orange3.bin、

orange4.bin、orange5.bin 按上述顺序写入软盘

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
00000000	5C	C8	8E	D8	BD	37	7C	8C	D8	8E	C0	B9	16	00	B8	01	EEZ0%7 G0ZA¹
00000016	13	BB	07	00	B6	00	B2	00	CD	10	B8	00	10	8E	C0	BB	» 9 ± f , ZÄ»
00000032	00	01	B4	02	B0	12	B2	00	B6	00	B5	00	B1	02	CD	13	, ° ± 9 p ± f
00000048	EA	00	01	00	10	EB	FE	4C	6F	61	64	69	6E	67	20	4D	è epLoading M
00000064	79	4F	73	20	6B	65	72	6E	61	6C	2E	2E	2E	00	00	00	yOs kernal...
00000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000096	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000112	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000128	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000144	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000176	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000192	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000208	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000224	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000240	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000256	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000272	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

## 七.程序运行效果

由于功能较多，所以只演示一部分功能，剩下的只能麻烦老师亲自测试了

内核界面：

```

\ME                                     LOVE\
                                     Welcome to MyOS!
      Here you can excute some functions for fun by key in specific commands.
      Please input "help" to see what commands that you can key in !

>>>_

\OPERATING                             \SYSTEM
```

键入 ‘help’：

```

\ME                                     LOVE\
                                     Welcome to MyOS!
      Here you can excute some functions for fun by key in specific commands.
      Please input "help" to see what commands that you can key in !

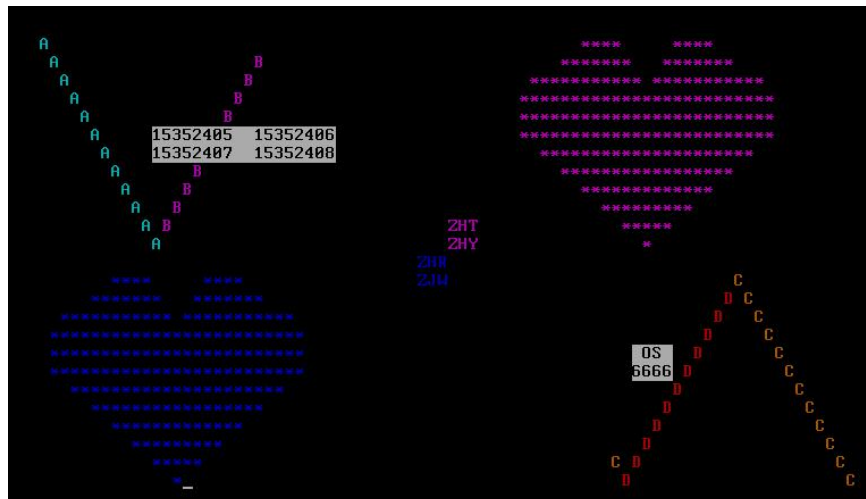
>>>help
time          - Print the System Time.
date          - Print the System Date.
ascii         - Query the ASCII Number of a character.
cls           - Clear the screen.
pro           - Creat Process and excute Process.
int21         - Call Interupt 21h!

>>>_

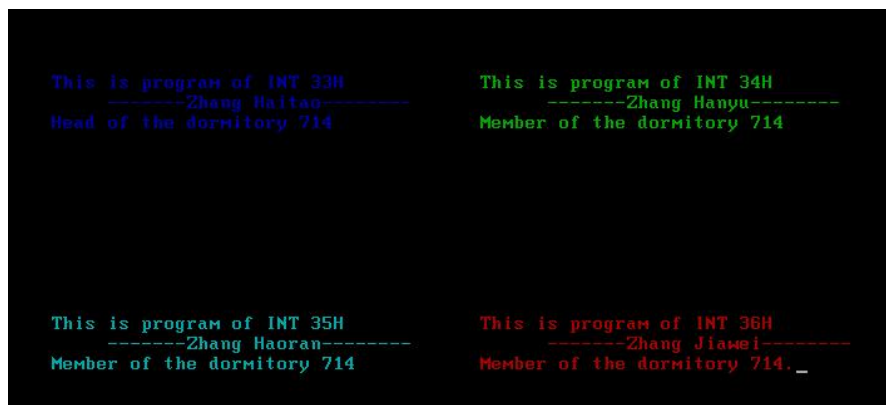
\OPERATING                             \SYSTEM
```

利用时钟中断同时运行 4 个用户程序：





33h,34h,35h,36h 四个中断能正常显示:



键入三次命令自动清屏:



```

:WE                                                     LOVE:
                Welcome to MyOS!
    Here you can excute some functions for fun by key in specific commands.
    Please input "help" to see what commands that you can key in !
time          - Print the System Time.
date          - Print the System Date.
ascii         - Query the ASCII Number of a character.
cls           - Clear the screen.
pro           - Creat Process and excute Process.
int21         - Call Interrupt 21h!

>>>_

:OPERATING                                           SYSTEM:

```

求 Ascii 码功能正常:

```

-WE                                                     LOVE-
                Welcome to MyOS!
    Here you can excute some functions for fun by key in specific commands.
    Please input "help" to see what commands that you can key in !
time          - Print the System Time.
date          - Print the System Date.
ascii         - Query the ASCII Number of a character.
cls           - Clear the screen.
pro           - Creat Process and excute Process.
int21         - Call Interrupt 21h!

>>>ascii

Key in the character you want to check for ASCII Number:z
The ASCII Number of z is 122.

>>>_

-OPERATING                                           SYSTEM-

```

求输入命令时的退格功能正常:

```

^WE                                                     LOVE^
                Welcome to MyOS!
    Here you can excute some functions for fun by key in specific commands.
    Please input "help" to see what commands that you can key in !
time          - Print the System Time.
date          - Print the System Date.
ascii         - Query the ASCII Number of a character.
cls           - Clear the screen.
pro           - Creat Process and excute Process.
int21         - Call Interrupt 21h!

>>>datet_

                Welcome to MyOS!
    Here you can excute some functions for fun by key in specific commands.
    Please input "help" to see what commands that you can key in !
time          - Print the System Time.
date          - Print the System Date.
ascii         - Query the ASCII Number of a character.
cls           - Clear the screen.
pro           - Creat Process and excute Process.
int21         - Call Interrupt 21h!

>>>date_

```

```

\4E                                     LOUE\
                                     Welcome to MyOS!
                                     Here you can excute some functions for fun by key in specific commands.
                                     Please input "help" to see what commands that you can key in !
time                                - Print the System Time.
date                                - Print the System Date.
ascii                               - Query the ASCII Number of a character.
cls                                 - Clear the screen.
pro                                 - Creat Process and excute Process.
int21                               - Call Interrupt 21h!

>>>date

Date: 2017/5/28

>>>_
```

## 【实验总结】

### 张浩然的实验总结：

这次的实验我们的参考程序不再是王师兄的代码了，改用了其他前辈的原型。按照要求，我们根据所给原型，进行修改完善。在实验过程中出了很多问题。比较明显的是进入用户程序中无法返回主界面，有时是所有的用户程序组合无法返回，有时是所有的情况都无法返回。造成这样的结果可能的原因是比较多样的，因为我们中间慢慢加了很多新的东西，每次的测试并不是把所有的组合测试过的，所以很难找出错误。最后我找出能够正确返回主界面的备份文件，再把新完善和实现的部分慢慢加上，并且每一步都把所有的用户程序输入组合情况测试一遍，确认没有问题后再加入下一个新的部分。这样得到了没有出错的程序。已知过程中犯的一个错误是测试四个中断的用户程序 5 的开头设定的开始位置没有修改，直接用了上次的，但这并不是全部的原因，因为之前的备份中也存在这个问题。我猜想也可能是编译过程中出错导致。但是程序还是有一些 bug。例如运行用户程序结束后很偶尔的情况会蓝屏（屏幕背景色变为蓝色，并且卡住），输入多个要运行的用户程序时最后一个不会显示等。这些问题可能会因为输入不合法的内容而增加出现的几率。这是我们目前研究过还是没能解决的问题，十分可惜。

### 张海涛的实验总结：

这次实验主要实现的是时间片轮转法，以此实现用户程序之间的交替进行，与之前的操作系统执行方式有很大的不同，对我自己而言，难度颇高，所以我把实验重心放到理解参考代码。保存当前程序现场，这个实现比较容易：将当前寄存器的内容压入栈中，将 PCB 需要的东西压入栈中作为参数传给 SaveCurrentPCB。此时的寄存器就可以被下一个用户程序使用了，通过调用 getCurrentPCB 载入之前保存的用户程序的信息。

对于 PCB 进程块的管理是本实验的关键，实验使用标记记录：当前进程号，进程计数，当前段地址，栈顶，是否进入用户态。每次程序轮转时，都需要更新这些标记，才能让 PCB 的轮转正常运行。理解了这些基本原理，我们的团队开始了尝试，然后就是无尽的对比 DEBUG，然后继续尝试，最后的成功真是不容易。

### 张晗宇的实验总结：

这次的实验内容是实现真正的多进程处理模型，之前几次的 time 是伪分时，实现的是假的多进程模拟。这次在失去了王师兄的代码后显得异常的困难，虽然看起来就是简单的出入栈的时钟中断，但是当开始处理之前代码之时，发现添加的位置，段地址的选取，调用方式等等细节都有问题，在把老师给的代码直接添加到原代码的之后，出现了执行多进程用户程序时只有光标但是没有用户程序的情况，而且光标也在变动，原本以为可能是 c 中调用的问题，改 c 代码减少 anotherload 的调用，试图只调用一个用户程序观察也未果，由于这次所给代码也不是很全（没有用户程序代码），而且使用的调

用方法也与之之前所用的有着很大的差异，所以也不能通过替换用户程序来观察多进程的模拟方式，经过多次尝试也没能找到原因。对于对段地址以及汇编语言不是很熟悉的我来说，总的来说这次实验比较困难，看了很久尝试了很多次都失败了，尤其是失败了还不能像 c 语言那样可以 `debug` 看到程序在哪里失败了。最后还是按原型的方式重新实现的。

### 张稼伟的实验总结：

这次实验，主要实现时间片轮转法来同时运行几个用户程序。没错，效果就是我们之前实现的伪分时功能，不过现在是真分时。在这次实验过程中，真是遇到了无尽的 `bug`，先是将原型代码中的、时间片轮转、进程块管理的代码移植到我们上一次的代码中，结果用户程序不能正常显示，我们猜测两个原因啊，一是段地址问题，二是延时问题。但最后没有成功解决。然后我们改变思路，将我们的用户程序移植到原型代码中，这次发现想运行某个用户程序，结果运行了下一个用户程序。仔细查看之后，发现一个是原型代码中默认每个用户程序使用一个扇区，而我们的用户程序是使用了两个扇区的，然后就是用 `winhex` 写入软盘的时候没有写对位置。对了，这次实验本来我们不是使用 `winhex` 写软盘的，而是使用 `inlbin` 的方式，不过总是不能正确加载地址，最后我们就去学习其他人用 `winhex` 写入。后面我负责修改 c 代码中求 `ascii` 码、退格功能和输入三次自动清屏这部分时还比较顺利，大概是因为对 C 代码还是比较熟悉吧。这次实验我们的经历可以说是在地狱中挣扎，最后终于得上天堂。过程虽苦，但结果比较甜。