《操作系统实验》

实验报告

（实验六）

学 院 名 称 ：	数据科学与计算机学院
专业（班级）：	17 级计算机类教务 3 班
学 生 姓 名 ：	姚森舰
学　　号 ：	17341189
时　　间 ：	2019　年　5　月　8　日

一. 实验目的

1. 在内核实现多进程的二状态，理解简单进程的构造方法和时间片轮转调度过程。
2. 实现解释多进程的控制台命令，建立相应进程并能启动执行。
3. 至少一个进程可用于测试前一版本的系统调用，搭建完整的操作系统框架，为后续实验项目打下扎实的基础。

二. 实验要求

1. 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：
   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

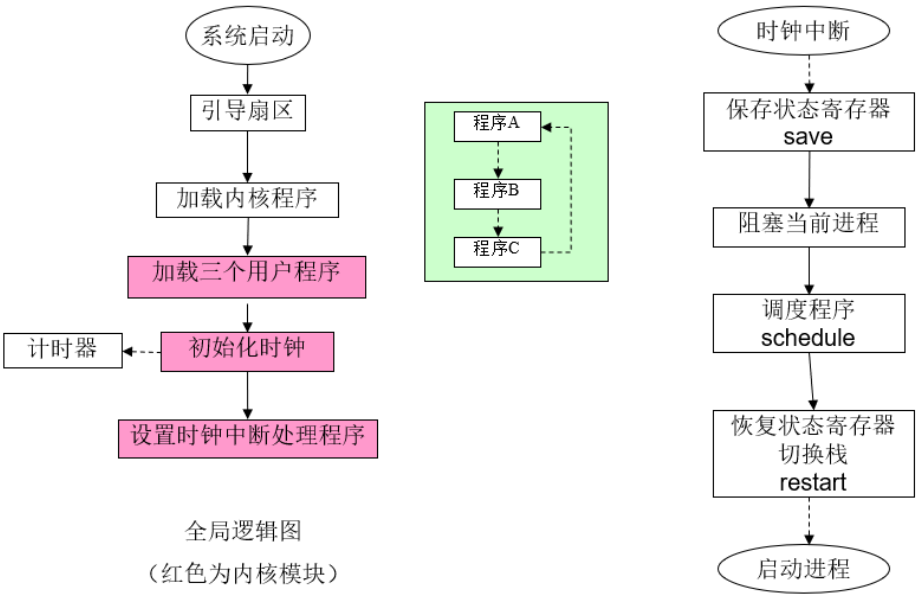
实验环境：Windows10 +VMware

实验工具：NASM + Winhex + Tasm + Tcc + Dosbox + Notepad++

大概流程：编译的流程是差不多的，用Notepad++编写C语言程序和汇编程序，用nasm把实验二中的用户程序和引导程序编译成二进制代码，在Dosbox下用TCC编译C代码，用TASM编译内核的部分汇编代码，在用Tlink将编译好的C和汇编代码link成.com文件。通过VMware创建带软盘的虚拟机，利用winhex将编译好的引导程序二进制代码写到虚拟机的软盘的第1个扇区，将文件数据写到第2个扇区，将内核代码放到3~12扇区，然后将用户程序放到第19~28扇区，然后在Vmware上运行测试虚拟机和程序。

大概思路：这次实验的主要部分是实现多进程，具体思路就是先处理多进程的指令，将要同时运行的用户程序加载到内存正确位置，然后修改时钟中断，在时钟中断发生时，将现在运行的程序的寄存器保存到对应的 PCB 结构中，当然这里面有很多值得注意的地方，在后面会详细说。保存完数据后，在调用 schedule()函数，将另一个进程的数据恢

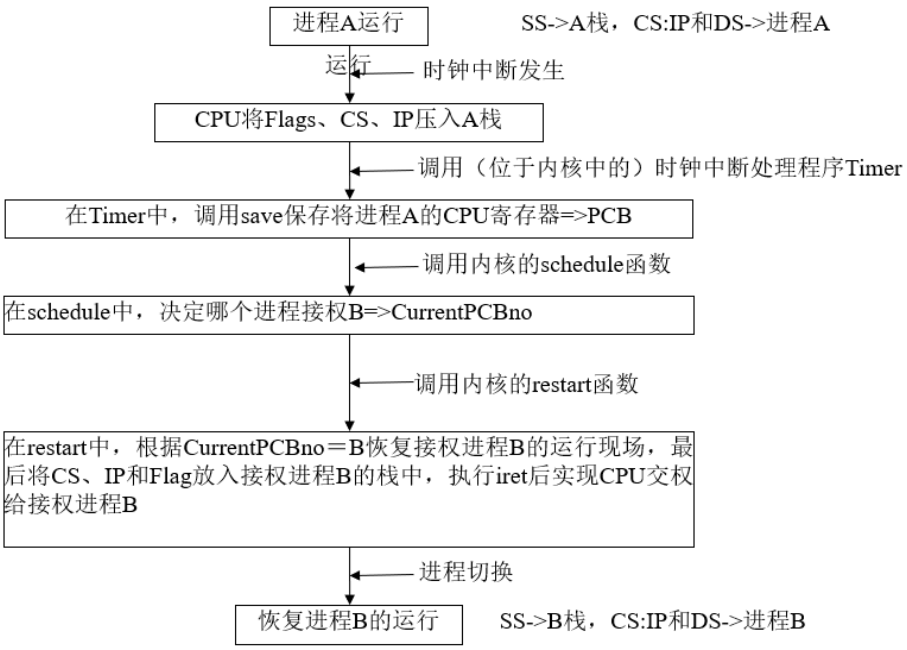
复，跳出时钟中断，继续运行该进程。即：



全局逻辑图

（红色为内核模块）

时钟中断里面做的事：



扇区安排：

1：引导程序

2：程序信息及批处理文件内容

3~18：内核，虽然内核还没有这么大

19~28：用户程序，每个用户程序占2个扇区

内存安排：

1. 操作系统内核：第 1 个 64k
2. 用户程序：一次按顺序紧接着放到内存，每一个占 64k

关键代码即及部分程序解释：

在PCB.H中定义了PCB结构体：

```
typedef struct RegisterImage{

typedef struct PCB{
    RegisterImage regImg;       /* 各个寄存器的值，多个int组成 */
    int Process_Status;         /* 进程状态 */
}PCB;
```

Current_Process()：返回现在运行的进程的PCB指针。

Save_Process()：保存所有寄存器值到对应PCB，传入参数是多个int，即寄存器内的值，具体见代码。

Schedule()：调度函数。

special()：程序第一次运行时，改变状态用。

init()：初始化各个PCB，需要注意的是内核也要有自己的PCB,当多进程运行完后，会利用该PCB返回内核。

```
void init_Pro()
{
    Program_Num = 0;                        /*  此时状态都是new */
    init(&pcb_list[0],0xA00,0x100);         /* 内核 段值，用时其实要乘16*/
    init(&pcb_list[1],0x1000,0x100);        /*64k 1000h*16*/
    init(&pcb_list[2],0x2000,0x100);
    init(&pcb_list[3],0x3000,0x100);
    init(&pcb_list[4],0x4000,0x100);
    init(&pcb_list[5],0x5000,0x100);
}
```

还有一些新加入的函数：

Delay()：双重循环延时，用于在多进程加载程序到内存后短暂延时，不然，会出现操作系统的一些信息提前出现，导致被用户程序会覆盖一些信息的情况。**有一个更好的想法是，判断Program_Num不为0，即还有用户程序时就一直循环，都执行完了再返回内核，相当于多线程常用的join函数，**但是由于Delay()这个本来仅仅是个延时函数，在其他地方都有用到，而上述想法仅仅为了多进程时使用，所有没有再做修改。

load_multi_process()：加载多进程程序，关键代码见下：

```
for( i = 1 ; i < len; i++){                    /* load */
    int repeat = 0;
    for( j = i-1; j >0; j--){                  /* 避免输入的指令重复 */
        if(cmdp[i]==cmdp[j]){
            repeat = 1;
        }
    }
    if(repeat==1) continue;

    if(cmdp[i] >='1' && cmdp[i] <='4' ){
        j = cmdp[i] - '0';
        justLoadp(Segment,j*2-1);              /* seg , begin_section */
        Segment += 0x1000;
        num_of_p++;                            /* 进程数 */
    }
}
Program_Num = num_of_p;                        /* 在外面一次性赋值，否则，当其++时 */
cls();                                         /* 时钟中断就去执行其他程序了,导致其他程序没有被加载 */
```

修改时钟中断，以下是时钟中断中的内容，先看save过程：

```
Finite dw 0
Timer:
;****************************************
;*                Save                  *
; ****************************************
    cmp word ptr[_Program_Num],0     ; 检查是否有程序要运行
    jnz Save                         ; 如果有，就保存其数据，做相应的操作
    jmp No_Progress                  ; 如果没有，就去执行风火轮程序
Save:
    inc word ptr[Finite]             ; 检查是否已达到运行次数的限制
    cmp word ptr[Finite],120         ;
    jnz Pass_agru                    ; 没有就传参，调用PCB.H中的save函数，保存数据
    mov word ptr[_CurrentPCBno],0    ; 当1600次调度完成后，将_CurrentPCBno置0，运行内核程序
    mov word ptr[Finite],0           ; 重新计数
    mov word ptr[_Program_Num],0     ; 当前程序改为0
    mov word ptr[_Segment],1000h     ; 可以确保顺序正确
    jmp Pre                          ; 准备数据，准备恢复运行
Pass_agru:                           ; 传参
    push ss
    push ax
    push bx
    push cx
    push dx
    push sp      ;sp已经被改变，保存的sp有误
                 ;cs，ip，flags + 前5个push，故需要+16来修正，栈是向低生长的
    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      ;进入timer时，原cs，ip，flag已在栈中，调用save()函数
    call near ptr _Schedule          ;保存完数据，调用schedule()函数
```

下面值得注意的是，ss和sp比较特殊，在恢复寄存器时，先要判断进程是不是第一次运行，如果是则继续向下执行，因为现在的PCB中的寄存器的值是初始化好的，是正确的值；如果

不是则需调整sp，具体就是将sp+16，这是因为save保存的sp是错误的sp，在进程运行时的sp是我们想保存的sp，此时发生了时钟中断，导致PSW,CS,IP依次入栈，sp+6，而在我上面的代码中，在保存sp之前，还保存了5个寄存器，也就是压栈了5次，sp要+10，栈是向低生长，故应当是+，故需要将sp+16. 可以说这部分代码是最重要的部分。

```
call near ptr _Current_Process  ; 得到现在将要运行的进程的PCB地址
mov bp, ax                      ; Current_Process的起始地址

mov ss,word ptr ds:[bp+0]       ; Current_Process的栈
mov sp,word ptr ds:[bp+16]      ; Current_Process的sp

cmp word ptr ds:[bp+32],0       ; new?
jnz No_First_Time               ; 如果不是第一次运行，需要调整sp

No_First_Time:
    add sp,16       ;修正
    jmp Restart
```

准备工作做好了，就可以恢复数据了，也就是restart部分的工作：先调用special()函数，将进程状态改为RUNNING，再调用Current_Process()函数，将返回在ax中的，将要运行的进程的PCB块的起始地址给bp，再加上对应寄存器的偏移量，开始恢复寄存器：

```
push word ptr ds:[bp+22]        ;bx
push word ptr ds:[bp+24]        ;ax

pop ax
pop bx
```

上面只截取了部分寄存器。恢复完成后，再手动将PCB中的PSW,CS,IP压栈，发EOI表示中断结束后，利用iret调到要执行的进程继续执行。

另外，对以前的代码做了一点修改，就是在单独调用一个用户程序的时候，也使用多进程的实现方式，只不过是相当于只有一个进程的多进程，也就是将该用户程序加载到内存后，将程序信息放到pcb_list[1]中（这些工作在runprog()函数中执行，pcb_list[0]为内核的PCB），置Program_Num为1，利用时钟中断来执行用户程序。

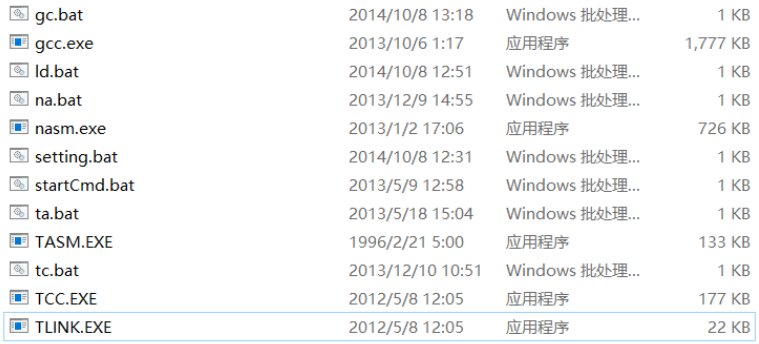
```
for(i = 1; i < len0; i++){
    int index_of_prog = recv[i]-'0';
    if(recv[i]==32) continue;
    init_Pro();
    if(index_of_prog > 0 && index_of_prog < 6){ /*只有5个程序可用*/
        runprog(index_of_prog);
        Program_Num = 1;
        Delay();
        cls();
    }
    else{
        prints("Invalid input.\n\r");
        break;
    }
}
```

```
void runprog(int index){
    int run_programs_seg = index*0x1000;              /* 段地址 */
    int begin_section = index*2-1;                    /* 起始扇区 */
    cls();
    justLoadp(run_programs_seg,begin_section);
    init(&pcb_list[1],run_programs_seg,0x100);        /* 只有1个程序 */
    cls();
}
```

四. 实验过程与思想

1.安装 VMware 并按要求创建一个无操作系统的裸机。

安装 Dosbox 并将 TCC，TASM，Tlink 等 exe 文件粘贴到对应文件

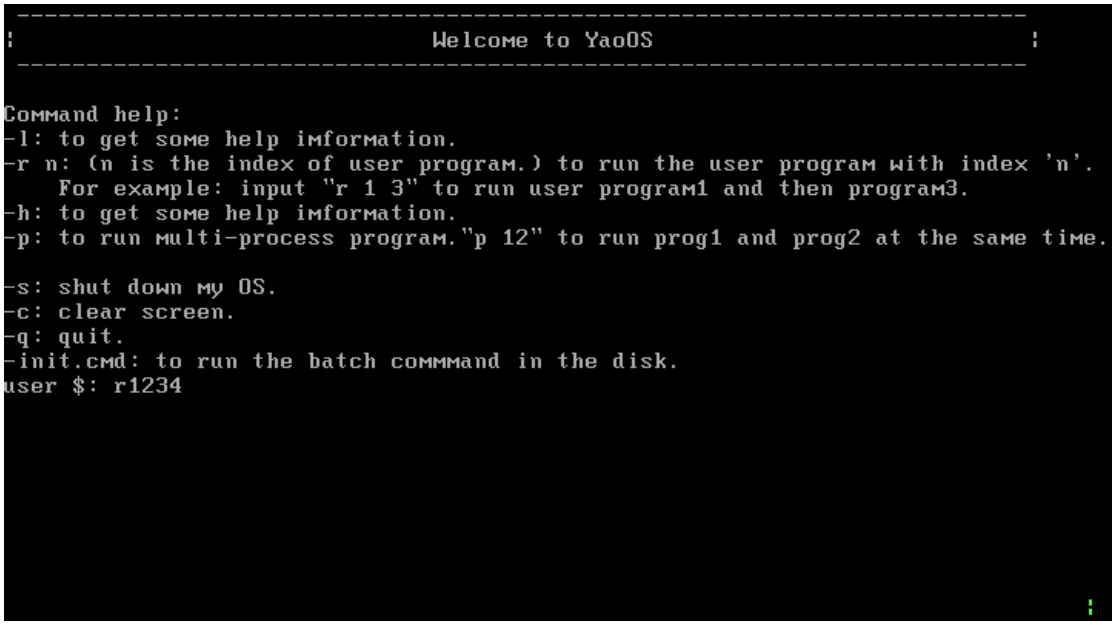


2. 编写汇编程序：
   关键代码见实验方案，具体代码见代码文件。

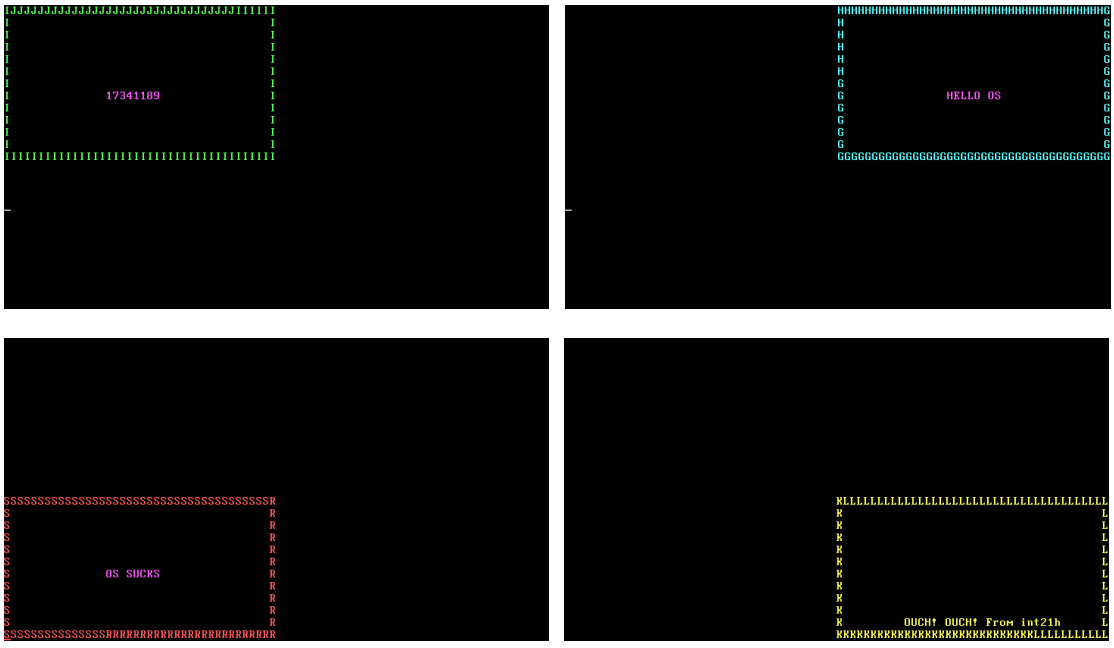
3. 编译：
   利用 DOS 批处理自动化编译，具体可见实验 5 实验报告。

4. 测试：
   因为之前的用户程序不是很动态，所以修改了一下用户程序，用字母不断画框，方便观察。另外，在下面的测试中，运行时按键盘还是会显示“ouch”的，只不过时间将时间调短了，很难在按键后，瞬间退出虚拟机截图，所以只截到一张，关于“ouch”和观察程序运行时的动态变化，可以打开虚拟机测试。

(1) 输入“r1234”，依次运行用户程序 1、2、3、4(批处理指令“init.cmd”的效果也是类似的,只是顺序不同，这里不再赘述，该指令内容已写到第二个扇区，需要修改测试可直接在软盘上修改后测试)：



可以看到程序依次成功运行：



（2）接下来测试多进程。输入“p1234”，同时运行用户程序 1、2、3、4：

成功运行：

（3）输入“p42”，同时运行用户程序 4、2：

成功运行：

（4）输入“r5”，运行用户程序 5，即测试系统调用：

由上可看出，系统调用正常，这和实验 5 中的结果是相同的。

五．实验心得和体会

最开始觉得老师给得原型写得很乱，就自己写了一个 save 函数，但是没有运行起来，在五一放假的时候电脑出了问题不得不重装，代码也丢了，只有用以前的代码重新开始，而且我也没明白在什么地方出了问题，所以后来还是用了老师的原型。在真正把老师的代码看

懂后发现老师的原型其实写的很巧妙，将内核看做一个进程，也给它一个 PCB，就很好的解决了返回内核的问题。

以下是遇到的一些问题和解决方案：

1. 老师的原型代码存在一些小小的问题，都和 Program_Num 有关：

```
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);
        Segment += 0x1000;
        Program_Num ++;
    }
```

这是原型中的 Random_load()中的部分代码，可以看到这段代码在处理多进程指令时，逐字母检测指令，然后加载程序 Program_Num++，问题就出现了，**一旦 Program_Num 改变了**，此时若发生了时钟中断，时钟中断里面就会发现 Program_Num 不为 0 了，就会开始执行多进程，然后调用 schedule()函数，而 schedule()函数的调度方案为：

```
CurrentPCBno ++;
if( CurrentPCBno > Program_Num )
    CurrentPCBno = 1;
```

也就是说，一旦开始调度，就会一直执行多进程，直到执行完毕才返回内核，而此时我们的多进程还没有全部加载完，就会出问题。所有应该引进应该临时计数的变量，记录程序数，最后在循环外面再一次性赋值给 Program_Num：

```
        Segment += 0x1000;
        num_of_p++;
    }
}
Program_Num = num_of_p;
```

另外一个问题是，没有解决重复输入的问题，其实这也不算是一个大问题：如果用户输入了 “p122”这种指令，老师的代码的 Program_Num 就会为 3，因为初始化的时候，把所有的用户程序都加载到了内存了，这就会导致有三个程序同时运行，所以应该排除输入中重复输入的指令，可以在处理指令的循环中加一些代码来解决：

(做这里的时候发现 TCC 居然不支持 bool 类型？！ 下面的 i 是从 1 开始的)

```
int repeat = 0;
for( j = i-1; j >0; j--){
    if(cmdp[i]==cmdp[j]){
        repeat = 1;
    }
}

if(repeat==1) continue;
```

另外，原型的这部分代码确实很巧妙，原型加载程序时，程序的段值增加都是 Segment+1000h 来完成的，在一开始的时候觉得这样做不是很好，不如知道了要加载第 n

个程序后，用 n*1000h 来算科学，后来才发现后者在程序运行顺序上会出现问题，比如我想运行第 3，4 个用户程序，此时 Program_Num=2，因为 CurrentPCBno 从 0 开始，每次都是加 1，加到 Program_Num 后变为 1，这样就会运行用户程序 1,2，就出错了，但是原型的方法就保证了顺序的正确。

2. 因为多进程和单独运行一个用户程序用的是同样的用户程序，而为了方便多进程展示，我将用户程序改成了无限循环，调度次数到了就同时结束，但是这样单独运行的时候就无法返回了。有想过按键退出，但是没想清楚这样会不会导致多个程序退出，也没去尝试，因为想到了另一个更好的方法，那就是使用多进程的方式实现单独运行程序，也就是把单独运行用户程序看做是一个进程的多进程，这样就很好的解决了问题。因此重写了load函数，去掉了跳转运行的语句。

3. 一开始没有搞懂为什么初始化PCB的时候，sp要是offset-4，问了同学才知道，原来是为了将栈底和代码段错开，至于为什么是-4，我觉得这应该是无关紧要的，至少测试了以下-2也是没问题的。

4. 刚完成实验时，发现其他地方没问题，但是按键盘却无法显示“ouch”，后来发现是设置时钟时将时钟设置的过快，调慢一些即可。原理没有想得很明白，按下键盘之后还没来的及发生中断就被切换到了其他程序?

总的来说，这个实验是做之前不知道如何下手，做完了就会感觉思路其实挺简单直接的，不过这个实验难的地方在于不知道如何DEBUG，对于寄存器的值无法输出对比，不知道是否保存了正确的数据，尤其是会自己改变的sp. 老师的原型给了很大的帮助，我也从中感受到了操作系统的精密。另外，这次实验也让我对多进程的实现原理有了更加深刻的理解，花了很多时间也值得。

六. 参考资料

[1] 老师的原型