

# 操作系统

# Operating Systems

## L9. 多进程图像

### Multiple Processes...

[lizhijun\\_os@hit.edu.cn](mailto:lizhijun_os@hit.edu.cn)

授课教师：李治军

综合楼416室

# 多个进程使用CPU的图像

用户的视角就是现在有很多个程序同时在运行



## ■ 如何使用CPU呢？

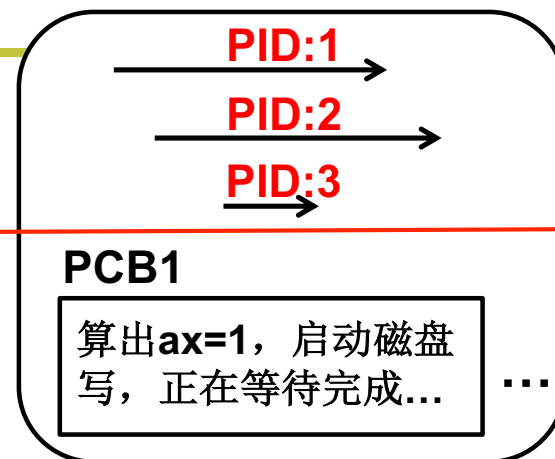
- 让程序执行起来

## ■ 如何充分利用CPU呢？

- 启动多个程序，交替执行

## ■ 启动了的程序就是进程，所以是多个进程推进

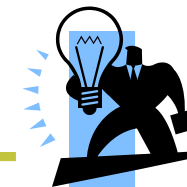
- 操作系统只需要把这些进程记录好、要按照合理的次序推进(分配资源、进行调度)
- 这就是多进程图像...



但是对于OS来做，需要维护各个进程的运行过程，调度等



# 多进程图像从启动开始到关机结束



## ■ main中的fork()创建了第1个进程

### ■ init执行了shell(Windows桌面)

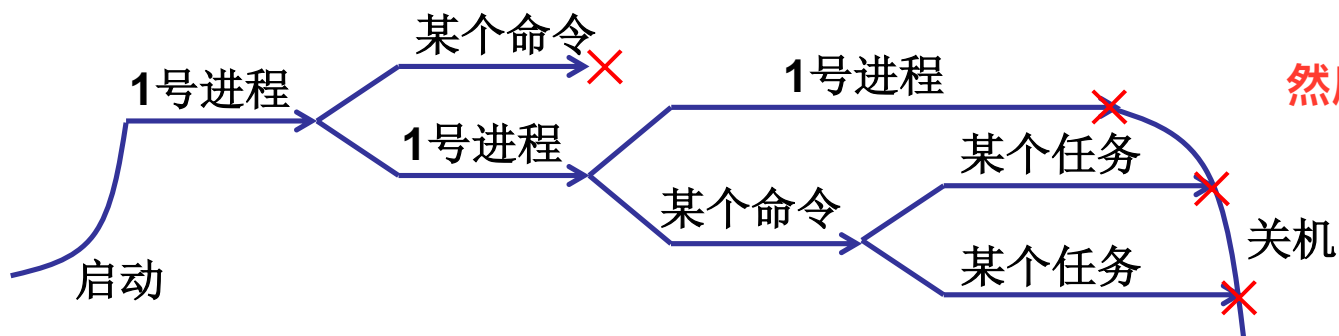
## ■ shell再启动其他进程

```
if(!fork()){init();}
```

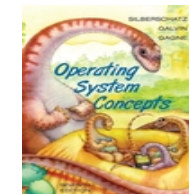
```
login as: tnp
tnp@192.168.1.1's password:
Last login: Mon Nov 13 23:47:50 2006 from 192.168.1.2
[tnp@tevsull ~]$ ls
work.txt
[tnp@tevsull ~]$
```

```
int main(int argc, char * argv[])
{ while(1) { scanf("%s", cmd);
  if(!fork()) {exec(cmd);} wait(); } }
```

一命令启动一个进程，返回shell再启动其他进程...



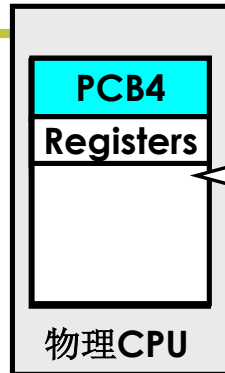
先创建一个shell进程  
然后其他进程都是在shell中创建出来的



操作系统感知进程、管理进程都是通过PCB的，  
通过建立一些数据结构来进行组织、管理。

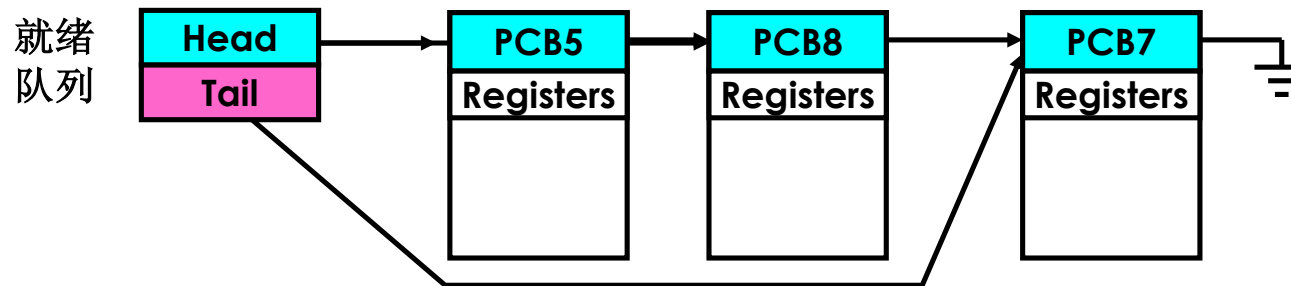
## 多进程图像：多进程如何组织？

- 有一个进程在执行



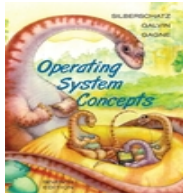
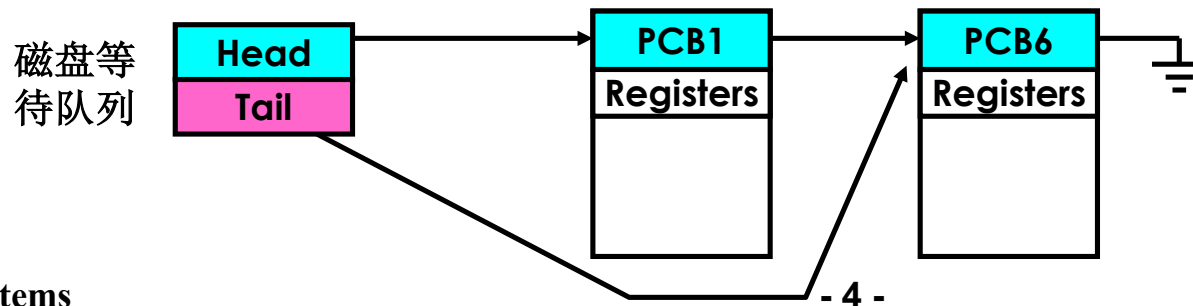
**Process Control Block:** 用来记录进程信息的数据结构

- 有一些进程等待执行



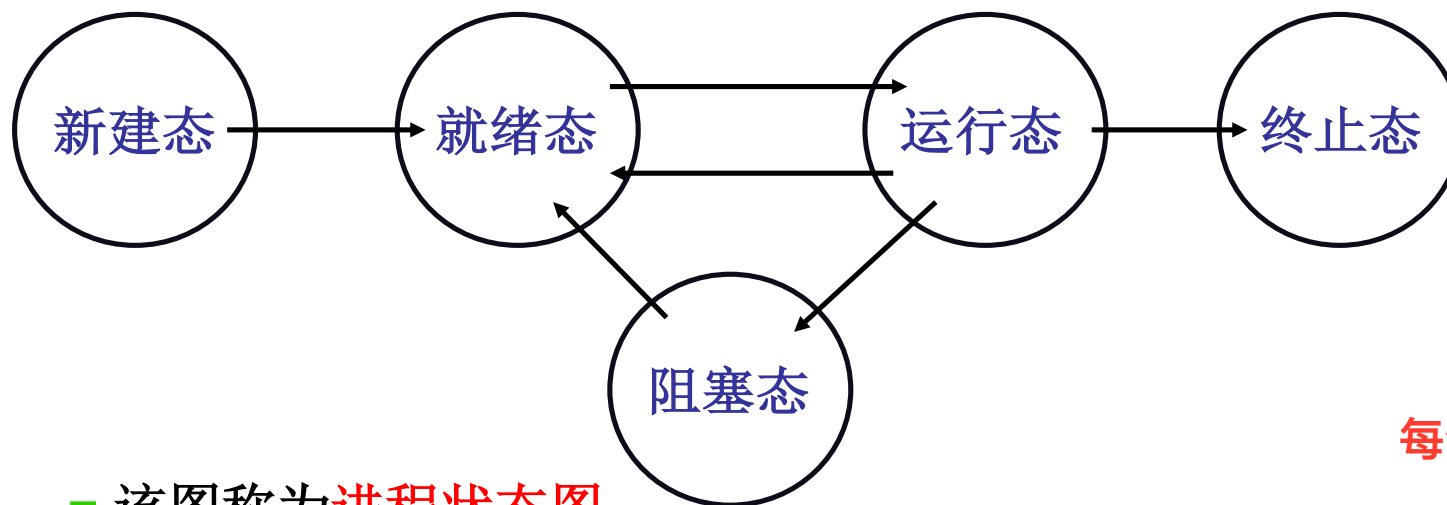
在PCB结构体之上建立一些数据结构来进行操作管理

- 有一些进程再等待某事件



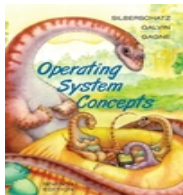
## 多进程的组织：PCB+状态+队列

■ 运行 → 等待; 运行→就绪; 就绪→运行.....



每个进程可能处于不同的状态，  
所以可以将处于不同状态的  
进程进行分类，分别用队列进行管理。

- 该图称为**进程状态图**
- 它能给出进程生存期的清晰描述
- 它是认识操作系统进程管理的一个窗口



## 多进程图像：多进程如何交替？

```
启动磁盘读写;  
pCur.state = 'W';  
将pCur放到DiskWaitQueue;  
schedule();
```

```
schedule()  
{  
    调度  
    pNew = getNext(ReadyQueue);  
    switch_to(pCur,pNew);  
    PCB      PCB  
}
```



# 交替的三个部分：队列操作+调度+切换

---

- 就是进程调度，一个很深刻的话题

- **FIFO?**

  - FIFO显然是公平的策略

  - FIFO显然没有考虑进程执行的任务的差别

- **Priority?**

  - 优先级该怎么设定？可能会使某些进程饥饿



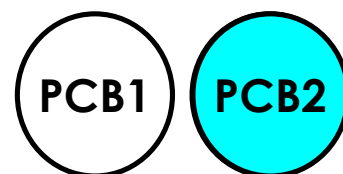
# 交替的三个部分：队列操作+调度+切换

```
switch_to(pCur, pNew) {
```

```
    pCur.ax = CPU.ax;  
    pCur.bx = CPU.bx;  
    ...  
    pCur.cs = CPU.cs;  
    pCur.retpc = CPU.pc;
```

保护现场

一段数据(PCB)



物理CPU

内存

```
    CPU.ax = pNew.ax;  
    CPU.bx = pNew.bx;  
    ...  
    CPU.cs = pNew.cs;  
    CPU.retpc = pNew.pc;
```

恢复现场

```
}
```





多个进程都在内存中，  
必须控制各个进程之间不能够  
相互进行访问各自的内存。  
所以操作系统必须对内存部分  
再进行抽象，让用户进程只能够  
看到虚拟内存，而不是实际的物理内存！

## 多进程图像：多进程如何影响？

- 多个进程同时存在于内存会出现下面的问题

进程1代码

```
mov ax, 10100b
mov [100], ax
.....
```

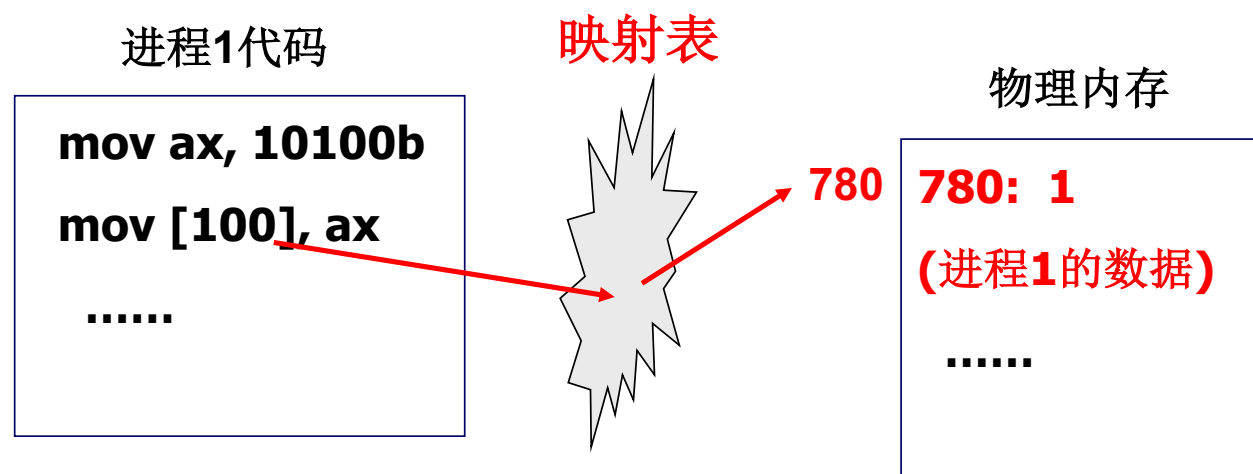
进程2代码

```
100: 00101
.....
```

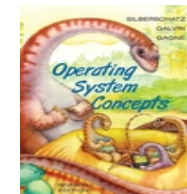
- 解决的办法：限制对地址100的读写
- 多进程的地址空间分离：内存管理的主要内容



# 进程执行时的100...

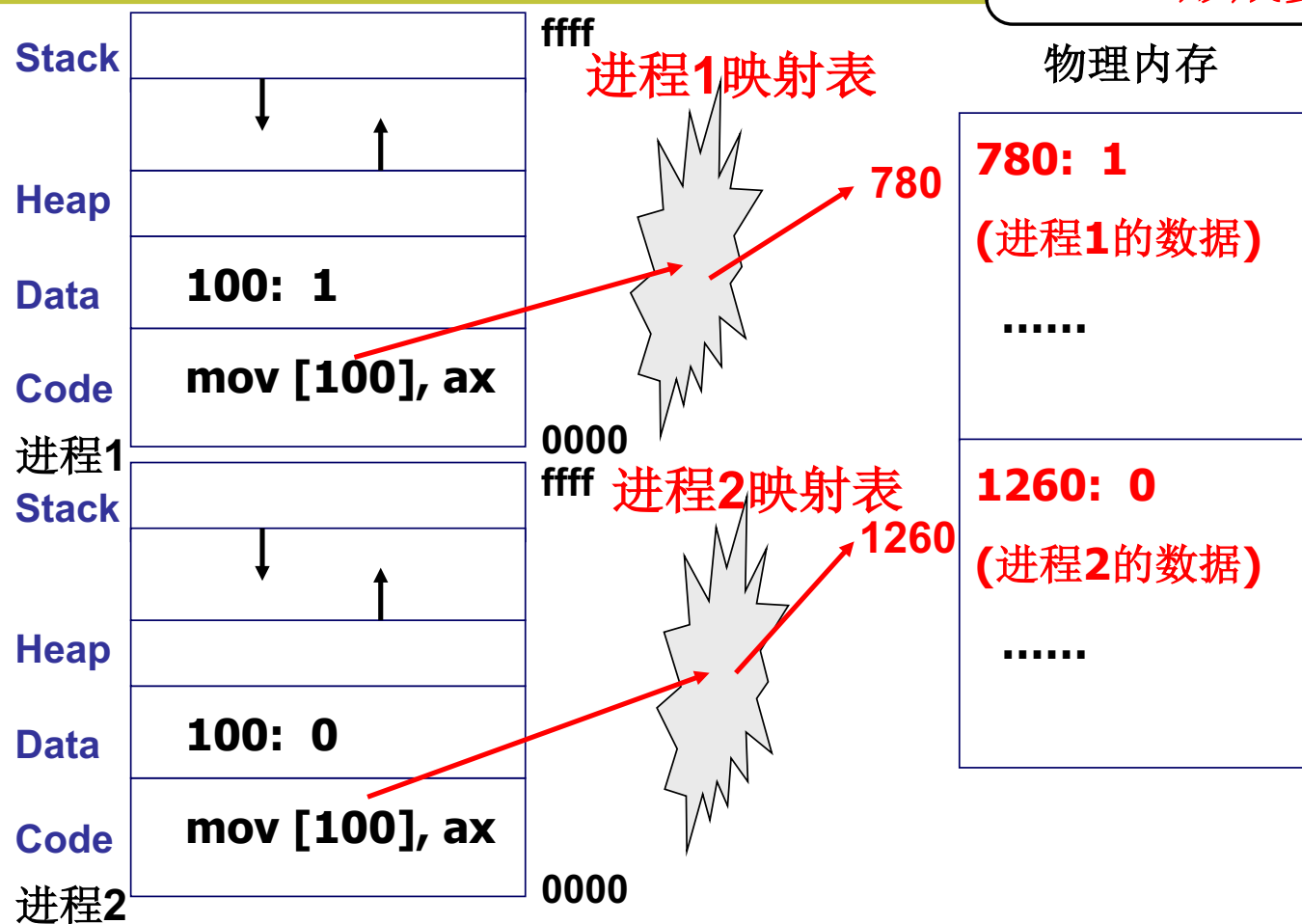


- 进程1的映射表将访问限制在进程1范围内
- 进程1根本访问不到其他进程的内容
- 内存管理...



# 进程带动内存的使用

为什么说进程管理连带内存管理  
形成多进程图像？



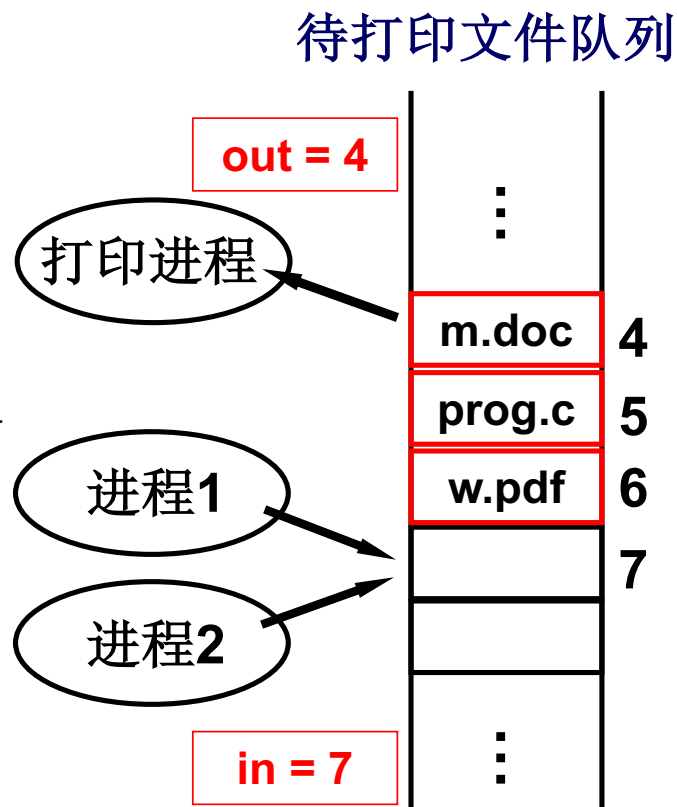
内存管理是为多进程  
服务的



# 多进程图像：多进程如何合作？

## ■ 想一想打印工作过程

- 应用程序提交打印任务
- 打印任务被放进打印队列
- 打印进程从队列中取出任务
- 打印进程控制打印机打印



# 从纸上到实际：生产者-消费者实例

生产者进程

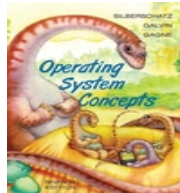
```
while (true) {  
    while(counter == BUFFER_SIZE)  
        ;  
    buffer[in] = item;  
    in = (in + 1) % BUFFER_SIZE;  
    counter++;  
}
```

共享数据

```
#define BUFFER_SIZE 10  
typedef struct { ... } item;  
item buffer[BUFFER_SIZE];  
int in = out = counter = 0;
```

消费者进程

```
while (true) {  
    while(counter == 0)  
        ;  
    item = buffer[out];  
    out = (out + 1) % BUFFER_SIZE;  
    counter--;  
}
```



# 两个合作的进程都要修改counter

## 一个可能的执行序列

### 共享数据

```
int counter=0;
```

### 生产者进程

```
counter++;
```

### 消费者进程

```
counter--;
```

### 初始情况

```
counter = 5;
```

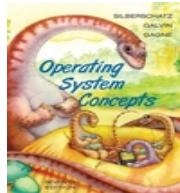
### 生产者P

```
register = counter;  
register = register + 1;  
counter = register;
```

### 消费者C

```
register = counter;  
register = register - 1;  
counter = register;
```

```
P.register = counter;  
P.register = P.register + 1;  
C.register = counter;  
C.register = C.register - 1;  
counter = P.register;  
counter = C.register;
```



# 核心在于进程同步(合理的推进顺序)

- 写**counter**时阻断其他进程访问**counter**

## 一个可能的执行序列

```
P.register = counter;  
P.register = P.register + 1;  
C.register = counter;  
C.register = C.register - 1;  
counter = P.register;  
counter = C.register;
```

生产者P

给**counter**上锁

```
P.register = counter;  
P.register = P.register + 1;
```

消费者C

检查**counter**锁

生产者P

```
counter = P.register;
```

给**counter**开锁

消费者C

给**counter**上锁

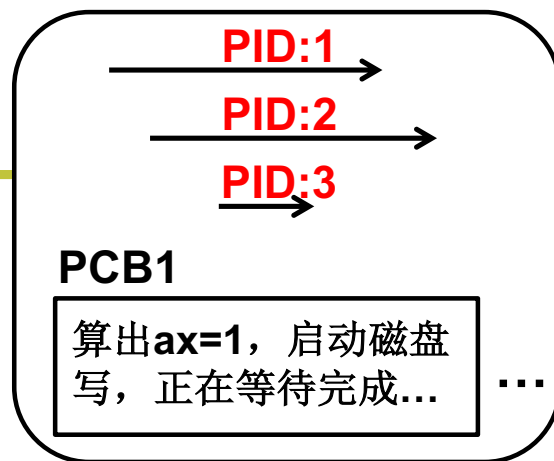
```
C.register = counter;  
C.register = C.register - 1;  
counter = C.register;
```

给**counter**开锁

不是在任意时刻都可以切换进程



如何形成多进程图像？



- 读写**PCB**, **OS**中最重要的结构, 贯穿始终
- 要操作寄存器完成切换 (**L10, L11, L12**)
- 要写调度程序 (**L13, L14**)
- 要有进程同步与合作 (**L16, L17**)
- 要有地址映射 (**L20**)

