操作系统(1)
第二章 进程描述与控制 课程作业

严昕宇 20121802

上海大学 计算机工程与科学学院

1 如果系统中有N个进程，那么运行进程最多几个，最少几个？就绪进程最多几个，最少几个？等待进程最多几个，最少几个？

1.1 单核心CPU

对于单核心的CPU（单处理机），如果系统中有N个进程，那么运行进程最多1个，最少0个；就绪进程最多N-1个，最少0个；等待进程最多N个（死锁状态），最少0个。

1.2 多核心CPU

还不太会，在考虑中...

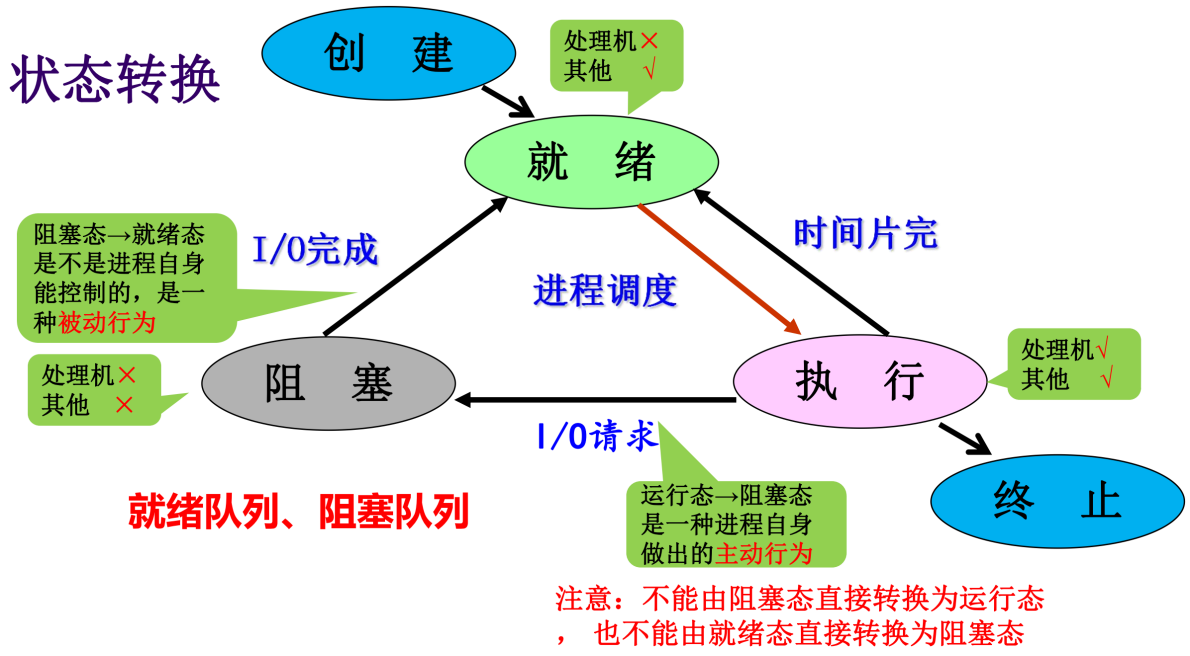
2 进程有无如下状态转换，为什么？

2.1 等待—运行

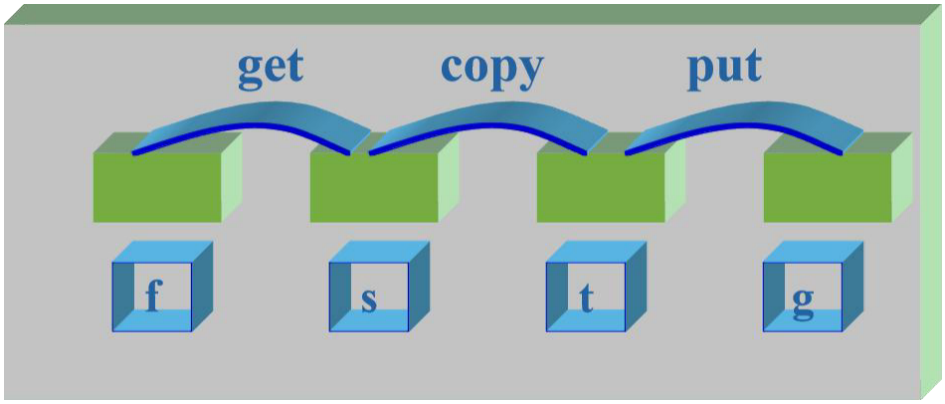
无。因为处于等待状态的进程，处理机资源和其他资源皆未获得，且不可能同时获得两种资源直接开始运行。应该是处于等待状态的进程获得需要的其他资源后，加入就绪队列，变为就绪态，等待被分配处理机。在分配了处理机资源后运行。

2.2 就绪—等待

无。因为处于就绪状态的进程除了未被分配处理机资源，其他资源均已获取，而等待状态则是处理机资源和其他资源都未获得。因此，应该是处于就绪状态的进程上处理机运行后，由于某些原因（如时间片完或处理机被抢占）进入等待状态。



3　用P.V操作解决下图之同步问题




```
semaphore mutex_s=1,mutex_t=1 // 代表f、s、t、g的互斥信号量
semaphore f_full=f,s_empty=s,s_full=0,t_empty=t,t_full=0,g_empty=g;
void get(){
 while(1){
      P(f_full);
      从f中取数据;
      P(s_empty);
      P(mutex_s);
      将数据放入s;
      V(mutex_s);
      V(s_full);
 }
}

void copy(){
 while(1){
      P(s_full);
      P(mutex_s);
      从s中取数据;
      V(mutex_s);
      V(s_empty);
      P(t_empty);
      P(mutex_t);
      将数据放入t;
      V(mutex_t);
      V(t_full);
 }
}

void put(){
 while(1){
      P(t_full);
      P(mutex_t);
```

```
        从t中取数据;
        V(mutex_t);
        V(t_empty);
        将数据Put到g中;
    }
}
```

4 试从动态性、并发性、独立性和异步性上比较进程和程序

4.1 动态性

进程的实质是进程实体的执行过程，因此，动态性就是进程的最基本的特征。程序只是一组有序指令的集合，并存放于某种介质上，其本身并不具有活动的含义，因而是静态的。简而言之，进程有一定的生命期，而程序只是一组有序的指令集合，是静态实体。进程由创建而产生，由调度而执行，由撤销而消亡。

4.2 并发性

多个进程实体同存于内存中，且能在一段时间内同时运行，这也是引入进程的目的之一，并发性是进程的重要特征，更是OS的重要特征。程序（没有建立PCB）是不能参与并发执行的。

4.3 独立性

进程实体是一个能独立运行，独立获得资源和独立接受调度的基本单位。凡未建立PCB的程序都不能作为一个独立的单位参与运行。

4.4 异步性

进程是按异步方式运行的，按各自独立的、不可预知的速度向前推进。正是因为异步性，传统意义上的程序若参与并发执行，会产生其结果的不可再现性。操作系统要提供“进程同步机制”来解决异步问题。

5 为什么进程在进入临界区之前应先执行“进入区”代码？而在退出前又要执行“退出区”代码？请说明原因

每个进程在进入临界区之前，应先对欲访问的临界资源进行检查，看它是否正被访问。如果此刻临界资源未被访问，进程便可以进入临界区对该资源进行访问，并设置它正被访问的标志；如果此刻该临界资源正被某进程访问，则本进程不能进入临界区。“进入区”就是为了在临界区之前进行上述的检查工作。而“退出区”用于将临界区正被访问的标志恢复为未被访问的标志。

因此进程在进入临界区之前先执行“进入区”代码，而在退出前又要执行“退出区”代码。

6 设P、Q、R共享一个缓冲区，P，Q构成一对生产者和消费者，R既为生产者又为消费者，使用P，V操作实现三个进程同步

```
semaphore mutex = 1;          // 缓冲区互斥信号量
semaphore empty = n, full = 0; // 生产者、消费者进程同步信号量

// 生产者P
```

```
void P() {
    while (1) {
        P(empty); // 如果缓冲区已满，则阻塞
        P(mutex);
        生产;
        V(mutex);
        V(full); // 如果消费者被阻塞，则唤起消费者
    }
}

// 消费者Q
void Q() {
    while (1) {
        P(full); // 如果缓冲区已空，则阻塞
        P(mutex);
        消费;
        V(mutex);
        V(empty); // 如果生产者被阻塞，则唤生产者
    }
}

// 生产者兼消费者R
void R() {
    while (1) {
        if (empty == n) { // 执行生产者功能
            P(empty);
            P(mutex);
            生产;
            V(mutex);
            V(full);
        }

        if (full == n) { // 执行消费者功能
            P(full);
            P(mutex);
            消费;
            V(mutex);
            V(empty);
        }
    }
}
```