 最近在写一个利用Java多线程的操作系统的课程设计，所以自然就开始认真学习了一下java有关多线程的一段知识，发现java对多线程的支持非常的好，一个锁机制可以解决大部分的问题。 我的第一个小实验性的东西是两个buffer，3中角色producer,consumer,mover。其中我感觉producer和 mover是最好实现的，只要在buffer的定义中定义了相关的put,get同步操作即可，关键是mover。通过分析可以知道，mover必须同时持有两个两个buffer的锁才能进行move的操作，开始的是就想到了这里。做了一个简单的示例，发现不对，总是会假死。事后才知道这是发生了死锁。开始的时候百思不得其解。于是去吃晚饭，过了一会儿，突然有了想法两个mover实例并发导致了死锁。 发生死锁的原因：mover1取得了buf1,buf2的锁，然后发现buf2满，于是释放buf2的锁，notify了 consumer，consumer于是开始cost，cost之后，notify,结果把锁给了另一个Mover，于是就这样了mover1有 buf1的锁，想要buf2，mover2有buf2的锁，想要buf1。死锁形成。我给出的解决方法比较粗糙，就是在mover类中设置一个静态的 Object作为mover所有实例的锁，mover首先要取得该锁，然后才可以取得buf1,buf2的锁，简单说就是叫mover串行。 结果很成功。 第二个实验：多个factory,多个store，都是buffer，然后还是3类producer,mover,consumer。这回的难点是选择策略由于每次producer不一定非要在同样的buffer间搬运，因此就存在每

次running时候的选择问题，如何选更加智能，尤其对于Mover,如何选择才能使之比较高效而且没有死锁。 经过考虑和实验，我的解决方法是利用Java自带的随机函数,new Random().nextInt(bufs.size())来随机选择 这里有Java API相关的解释： nextInt public int nextInt(int n) 返回一个伪随机数，它是取自此随机数生成器序列的、在 0（包括）和指定值（不包括）之间均匀分布的 int 值。nextInt 的常规协定是，伪随机地生成并返回指定范围中的一个 int 值。所有可能的 n 个 int 值的生成概率（大致）相同。Random 类按如下方式实现 nextInt(int n) 方法： public int nextInt(int n) { if (n 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com