入了死锁——不停地进行分支循环和运行失败。发生这种情况的可能性是极小的，但是，这是可能发生的！我们是否应该放弃进程以及fork调用来消除这个问题呢？

限制打开文件的最大数量与限制索引节点表的大小的方式很相像，因此，当它被完全占用的时候，也会出现相似的问题。硬盘上的交换空间是另一个有限的资源。事实上，几乎操作系统中的每种表都代表了一种有限的资源。如果有$n$个进程，每个进程都申请了$1/n$的资源，然后每一个又试图申请更多的资源，这种情况下我们是不是应该禁掉所有的呢？也许这不是一个好主意。

大多数的操作系统（包括UNIX和Windows）都忽略了一个问题，即比起限制所有用户去使用一个进程、一个打开的文件或任意一种资源来说，大多数用户可能更愿意选择一次偶然的活锁（或者甚至是死锁）。如果这些问题能够免费消除，那就不会有争论。但问题是代价非常高，因而几乎都是给进程加上不便的限制来处理。因此我们面对的问题是从便捷性和正确性中做出取舍，以及一系列关于哪个更重要、对谁更重要的争论。

值得一提的是，一些人对饥饿（缺乏资源）和死锁并不作区分，因为在两种情况下都没有下一步操作了。还有些人认为它们从根本上不同，因为可以很轻易地编写一个进程，让它做某个操作$n$次，并且如果它们都失败了，再试试其他的就可以了。一个阻塞的进程就没有那样的选择了。

### 6.7.4 饥饿

与死锁和活锁非常相似的一个问题是饥饿（starvation）。在动态运行的系统中，在任何时刻都可能请求资源。这就需要一些策略来决定在什么时候谁获得什么资源。虽然这个策略表面上很有道理，但依然有可能使一些进程永远得不到服务，虽然它们并不是死锁进程。

作为一个例子，考虑打印机分配。设想系统采用某种算法来保证打印机分配不产生死锁。现在假设若干进程同时都请求打印机，究竟哪一个进程能获得打印机呢？

一个可能的分配方案是把打印机分配给打印最小文件的进程（假设这个信息可知）。这个方法让尽量多的顾客满意，并且看起来很公平。我们考虑下面的情况：在一个繁忙的系统中，有一个进程有一个很大的文件要打印，每当打印机空闲，系统纵观所有进程，并把打印机分配给打印最小文件的进程。如果存在一个固定的进程流，其中的进程都是只打印小文件，那么，要打印大文件的进程永远也得不到打印机。很简单，它会“饥饿而死”（无限制地推后，尽管它没有被阻塞）。

饥饿可以通过先来先服务资源分配策略来避免。在这种机制下，等待最久的进程会是下一个被调度的进程。随着时间的推移，所有进程都会变成最“老”的，因而，最终能够获得资源而完成。

## 6.8 有关死锁的研究

死锁在操作系统发展的早期就作为一个课题被详细地研究过。死锁的检测是一个经典的图论问题，任何对数学有兴趣的研究生都可以在其上做3～4年的研究。所有相关的算法都已经经过了反复修正，但每次修正总是得到更古怪、更不现实的算法。大部分工作已经结束了，但是仍然有很多关于死锁各方面内容的论文发表。这些论文包括由于错误使用锁和信号量而导致的死锁的运行时间检测（Agarwal和Stoller, 2006；Bensalem等人, 2006）；在Java线程中预防死锁（Permandia等人，2007；Williams等人，2005）；处理网络上的死锁（Jayasimha, 2003；Karol等人，2003；Schafer等人，2005）；数据流系统中的死锁建模（Zhou和Lee, 2006）；检测动态死锁（Li等人，2005）。Levine（2003a, 2003b）比较了文献中关于死锁各种不同的（经常相矛盾的）定义，从而提出了一个分类方案。她也从另外的角度分析了关于预防死锁和避免死锁的区别（Levine，2005）。而死锁的恢复也是一个正在研究的问题（David等人，2007）。

然而，还有一些（理论）研究是关于分布式死锁检测的，我们在这里不做表述，因为它超出了本书的范围，而且这些研究在实际系统中的应用非常少，似乎只是为了让一些图论家有事可做罢了。

## 6.9 小结

死锁是任何操作系统的潜在问题。在一组进程中，每个进程都因等待由该组进程中的另一进程所占有的资源而导致阻塞，死锁就发生了。这种情况会使所有的进程都处于无限等待的状态。一般来讲，这是进程一直等待被其他进程占用的某些资源释放的事件。死锁的另外一种可能的情况是一组通信进程都