不过，有一个小思路是经常可适用的。那就是，避免分配那些不是绝对必需的资源，尽量做到尽可能少的进程可以真正请求资源。

6.6.2 破坏占有和等待条件

Coffman 等表述的第二个条件似乎更有希望。只要禁止已持有资源的进程再等待其他资源便可以消除死锁。一种实现方法是规定所有进程在开始执行前请求所需的全部资源。如果所需的全部资源可用，那么就将它们分配给这个进程，于是该进程肯定能够运行结束。如果有一个或多个资源正被使用，那么就不进行分配，进程等待。

这种方法的一个直接问题是很多进程直到运行时才知道它需要多少资源。实际上，如果进程能够知道它需要多少资源，就可以使用银行家算法。另一个问题是这种方法的资源利用率不是最优的。例如，有一个进程先从输入磁带上读取数据，进行一小时的分析，最后会写到输出磁带上，同时会在绘图仪上绘出。如果所有资源都必须提前请求，这个进程就会把输出磁带机和绘图仪控制住一小时。

不过，一些大型机批处理系统要求用户在所提交的作业的第一行列出它们需要多少资源。然后，系统立即分配所需的全部资源，并且直到作业完成才回收资源。虽然这加重了编程人员的负担，也造成了资源的浪费，但这的确防止了死锁。

另一种破坏占有和等待条件的略有不同的方案是，要求当一个进程请求资源时，先暂时释放其当前占用的所有资源，然后再尝试一次获得所需的全部资源。

6.6.3 破坏不可抢占条件

破坏第三个条件（不可抢占）也是可能的。假若一个进程已分配到一台打印机，且正在进行打印输出，如果由于它需要的绘图仪无法获得而强制性地把它占有的打印机抢占掉，会引起一片混乱。但是，一些资源可以通过虚拟化的方式来避免发生这样的情况。假脱机打印机向磁盘输出，并且只允许打印机守护进程访问真正的物理打印机，这种方式可以消除涉及打印机的死锁，然而却可能带来由磁盘空间导致的死锁。但是对于大容量磁盘，要消耗完所有的磁盘空间一般是不可能的。

然而，并不是所有的资源都可以进行类似的虚拟化。例如，数据库中的记录或者操作系统中的表都必须被锁定，因此存在出现死锁的可能。

6.6.4 破坏环路等待条件

现在只剩下一个条件了。消除环路等待有几种方法。一种是保证每一个进程在任何时刻只能占用一个资源，如果要请求另外一个资源，它必须先释放第一个资源。但假若进程正在把一个大文件从磁带机上读入并送到打印机打印，那么这种限制是不可接受的。

另一种避免出现环路等待的方法是将所有资源统一编号，如图6-13a所示。现在的规则是：进程可以在任何时刻提出资源请求，但是所有请求必须按照资源编号的顺序（升序）提出。进程可以先请求打印机后请求磁带机，但不可以先请求绘图仪后请求打印机。

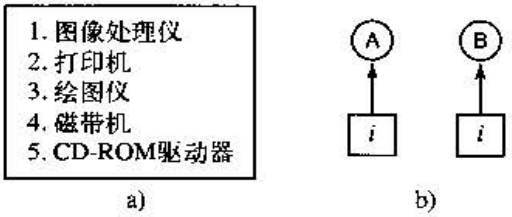

图6-13 a) 对资源排序编号；
b) 一个资源分配图

若按此规则，资源分配图中肯定不会出现环。让我们看看在有两个进程的情形下为何可行，参看图6-13b。只有在A请求资源j且B请求资源i的情况下会产生死锁。假设i和j是不同的资源，它们会具有不同的编号。若$i>j$，那么A不允许请求j，因为这个编号小于A已有资源的编号；若$i<j$，那么B不允许请求i，因为这个编号小于B已有资源的编号。不论哪种情况都不可能产生死锁。

对于多于两个进程的情况，同样的逻辑依然成立。在任何时候，总有一个已分配的资源是编号最高的。占用该资源的进程不可能请求其他已分配的各种资源。它或者会执行完毕，或者最坏的情形是去请求编号更高的资源，而编号更高的资源肯定是可用的。最终，它会结束并释放所有资源，这时其他占有最高编号资源的进程也可以执行完。简言之，存在一种所有进程都可以执行完毕的情景，所以不会产生死锁。

该算法的一个变种是摈弃必须按升序请求资源的限制，而仅仅要求不允许进程请求比当前所占有资