一个进程能够完成。而且，在A请求其他资源实例前，A可能先释放一个资源实例，这就可以让C先完成，从而避免了死锁。因而，安全状态和不安全状态的区别是：从安全状态出发，系统能够保证所有进程都能完成；而从不安全状态出发，就没有这样的保证。

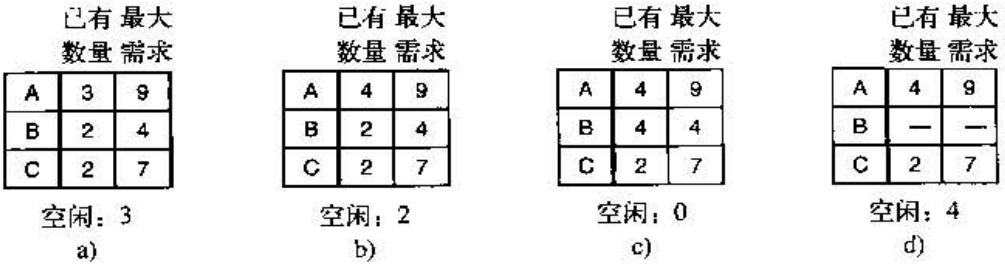

图6-10 说明b中的状态为不安全状态

6.5.3 单个资源的银行家算法

Dijkstra（1965）提出了一种能够避免死锁的调度算法，称为*银行家算法*（banker's algorithm），这是6.4.1节中给出的死锁检测算法的扩展。该模型基于一个小城镇的银行家，他向一群客户分别承诺了一定的贷款额度。算法要做的是判断对请求的满足是否会导致进入不安全状态。如果是，就拒绝请求；如果满足请求后系统仍然是安全的，就予以分配。在图6-11a中我们看到4个客户A、B、C、D，每个客户都被授予一定数量的贷款单位（比如1单位是1千美元），银行家知道不可能所有客户同时都需要最大贷款额，所以他只保留10个单位而不是22个单位的资金来为客户服务。这里将客户比作进程，贷款单位比作资源，银行家比作操作系统。

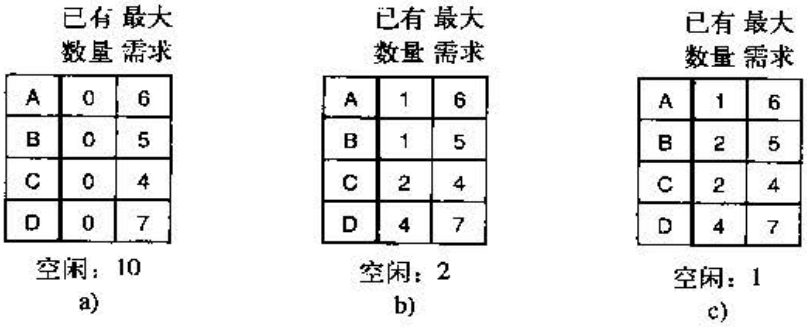

图6-11 三种资源分配状态：a) 安全；b) 安全；c) 不安全

客户们各自做自己的生意，在某些时刻需要贷款（相当于请求资源）。在某一时刻，具体情况如图6-11b所示。这个状态是安全的，由于保留着2个单位，银行家能够拖延除了C以外的其他请求。因而可以让C先完成，然后释放C所占的4个单位资源。有了这4个单位资源，银行家就可以给D或B分配所需的贷款单位，以此类推。

考虑假如向B提供了另一个他所请求的贷款单位，如图6-11b所示，那么我们就有如图6-11c所示的状态，该状态是不安全的。如果忽然所有的客户都请求最大的限额，而银行家无法满足其中任何一个的要求，那么就会产生死锁。不安全状态并不一定引起死锁，由于客户不一定需要其最大贷款额度，但银行家不敢抱这种侥幸心理。

银行家算法就是对每一个请求进行检查，检查如果满足这一请求是否会达到安全状态。若是，那么就满足该请求；若否，那么就推迟对这一请求的满足。为了看状态是否安全，银行家看他是否有足够的资源满足某一个客户。如果可以，那么这笔投资认为是能够收回的，并且接着检查最接近最大限额的一个客户，以此类推。如果所有投资最终都被收回，那么该状态是安全的，最初的请求可以批准。

6.5.4 多个资源的银行家算法

可以把银行家算法进行推广以处理多个资源。图6-12说明了多个资源的银行家算法如何工作。

在图6-12中我们看到两个矩阵。左边的矩阵显示出为5个进程分别已分配的各种资源数，右边的矩