死锁检测算法就是基于向量的比较。我们定义向量A和向量B之间的关系为$A \leqslant B$以表明A的每一个分量要么等于要么小于和B向量相对应的分量。从数学上来说，$A \leqslant B$当且仅当且$A_i \leqslant B_i (0 \leqslant i \leqslant m)$。

每个进程起初都是没有标记过的。算法开始会对进程做标记，进程被标记后就表明它们能够被执行，不会进入死锁。当算法结束时，任何没有标记的进程都是死锁进程。该算法假定了一个最坏情形：所有的进程在退出以前都会不停地获取资源。

死锁检测算法如下：

1) 寻找一个没有标记的进程P_i，对于它而言R矩阵的第i行向量小于或等于A。

2) 如果找到了这样一个进程，那么将C矩阵的第i行向量加到A中，标记该进程，并转到第1步。

3) 如果没有这样的进程，那么算法终止。

算法结束时，所有没有标记过的进程（如果存在的话）都是死锁进程。

算法的第1步是寻找可以运行完毕的进程，该进程的特点是它有资源请求并且该请求可被当前的可用资源满足。这一选中的进程随后就被运行完毕，在这段时间内它释放自己持有的所有资源并将它们返回到可用资源库中。然后，这一进程被标记为完成。如果所有的进程最终都能运行完毕的话，就不存在死锁的情况。如果其中某些进程一直不能运行，那么它们就是死锁进程。虽然算法的运行过程是不确定的（因为进程可按任何行得通的次序执行），但结果总是相同的。

作为一个例子，在图6-7中展示了用该算法检测死锁的工作过程。这里我们有3个进程、4种资源（可以任意地将它们标记为磁带机、绘图仪、扫描仪和CD-ROM驱动器）。进程1有一台扫描仪。进程2有2台磁带机和1个CD-ROM驱动器。进程3有1个绘图仪和2台扫描仪。每一个进程都需要额外的资源，如矩阵R所示。

要运行死锁检测算法，首先找出哪一个进程的资源请求可被满足。第1个不能被满足，因为没有CD-ROM驱动器可供使用。第2个也不能被满足，由于没有打印机空闲。幸运的是，第3个可被满足，所以进程3运行并最终释放它所拥有的资源，给出

$$A = (2\ 2\ 2\ 0)$$

接下来，进程2也可运行并释放它所拥有的资源，给出

$$A = (4\ 2\ 2\ 1)$$

现在剩下的进程都能够运行，所以这个系统中不存在死锁。

假设图6-7的情况有所改变。进程2需要1个CD-ROM驱动器、2台磁带机和1台绘图仪。在这种情况下，所有的请求都不能得到满足，整个系统进入死锁。

现在我们知道了如何检测死锁（至少是在这种预先知道静态资源请求的情况下），但问题在于何时去检测它们。一种方法是每当有资源请求时去检测。毫无疑问越早发现越好，但这种方法会占用昂贵的CPU时间。另一种方法是每隔k分钟检测一次，或者当CPU的使用率降到某一域值时去检测。考虑到CPU使用效率的原因，如果死锁进程数达到一定数量，就没有多少进程可运行了，所以CPU会经常空闲。

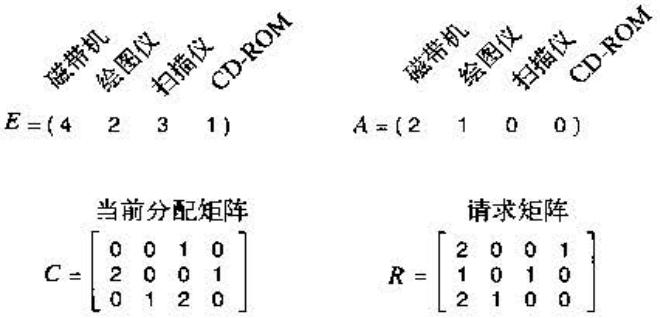

图6-7　死锁检测算法的一个例子

6.4.3　从死锁中恢复

假设我们的死锁检测算法已成功地检测到了死锁，那么下一步该怎么办？当然需要一些方法使系统重新正常工作。在本小节中，我们会讨论各种从死锁中恢复的方法，尽管这些方法看起来都不那么令人满意。

1. 利用抢占恢复

在某些情况下，可能会临时将某个资源从它的当前所有者那里转移到另一个进程。许多情况下，尤其是对运行在大型主机上的批处理操作系统来说，需要人工进行干预。

比如，要将激光打印机从它的持有进程那里拿走，管理员可以收集已打印好的文档并将其堆积在一