们再来看一下数据库的例子。假设内核使用优先级调度算法，但提供一条可供进程设置（并改变）优先级的系统调用。这样，尽管父进程本身并不参与调度，但它可以控制如何调度子进程的细节。在这里，调度机制位于内核，而调度策略则由用户进程决定。

2.4.6 线程调度

当若干进程都有多个线程时，就存在两个层次的并行：进程和线程。在这样的系统中调度处理有本质差别，这取决于所支持的是用户级线程还是内核级线程（或两者都支持）。

首先考虑用户级线程。由于内核并不知道有线程存在，所以内核还是和以前一样地操作，选取一个进程，假设为A，并给予A以时间片控制。A中的线程调度程序决定哪个线程运行，假设为A1。由于多道线程并不存在时钟中断，所以这个线程可以按其意愿任意运行多长时间。如果该线程用完了进程的全部时间片，内核就会选择另一个进程运行。

在进程A终于又一次运行时，线程A1会接着运行。该线程会继续耗费A进程的所有时间，直到它完成工作。不过，该线程的这种不合群的行为不会影响到其他的进程。其他进程会得到调度程序所分配的合适份额，不会考虑进程A内部所发生的事。

现在考虑A线程每次CPU计算的工作比较少的情况，例如，在50ms的时间片中有5ms的计算工作。于是，每个线程运行一会儿，然后把CPU交回给线程调度程序。这样在内核切换到进程B之前，就会有序列A1，A2，A3，A1，A2，A3，A1，A2，A3，A1。这种情形可用图2-43a表示。

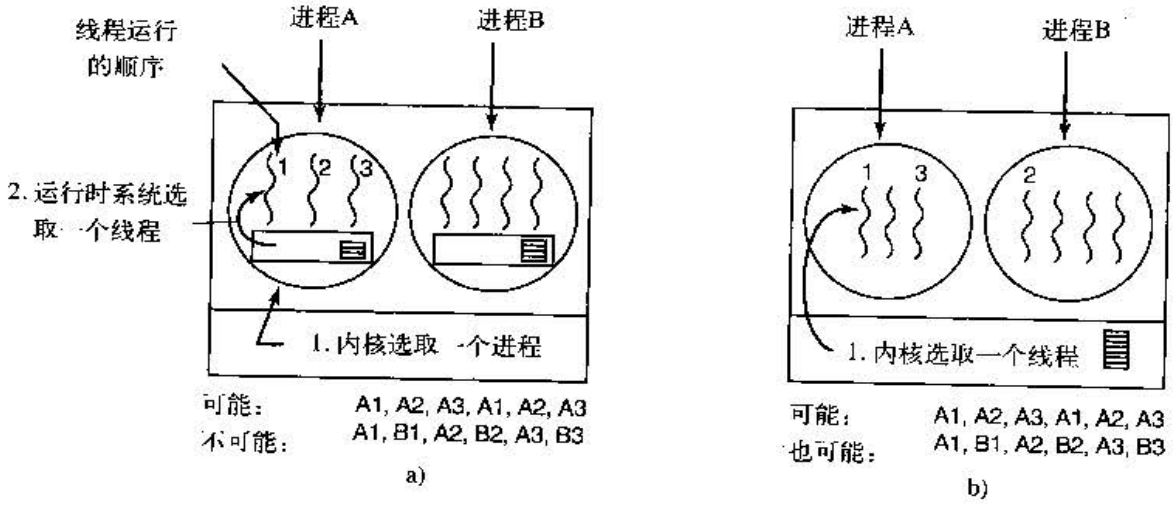

图2-43 a) 用户级线程的可能调度，有50ms时间片的进程以及每次运行5ms CPU的线程；b) 与a) 有相同特性的内核级线程的可能调度

实时系统使用的调度算法可以是上面介绍的算法中的任意一种。从实用考虑，轮转调度和优先级调度更为常用。惟一的局限是，缺乏一个时钟将运行过长的线程加以中断。

现在考虑使用内核级线程的情形。内核选择一个特定的线程运行。它不用考虑该线程属于哪个进程，不过如果有必要的话，它可以这样做。对被选择的线程赋予一个时间片，而且如果超过了时间片，就会强制挂起该线程。一个线程在50ms的时间片内，5ms之后被阻塞，在30ms的时间段中，线程的顺序会是A1，B1，A2，B2，A3，B3，在这种参数和用户线程状态下，有些情形是不可能出现的。这种情形部分通过图2-43b刻画。

用户级线程和内核级线程之间的差别在于性能。用户级线程的线程切换需要少量的机器指令，而内核级线程需要完整的上下文切换，修改内存映像，使高速缓存失效，这导致了若干数量级的延迟。另一方面，在使用内核级线程时，一旦线程阻塞在I/O上就不需要像在用户级线程中那样将整个进程挂起。

从进程A的一个线程切换到进程B的一个线程，其代价高于运行进程A的第2个线程（因为必须修改内存映像，清除内存高速缓存的内容），内核对此是了解的，并可运用这些信息做出决定。例如，给定两个在其他方面同等重要的线程，其中一个线程与刚好阻塞的线程属于同一个进程，而另一个线程属于其他的进程，那么应该倾向前者。