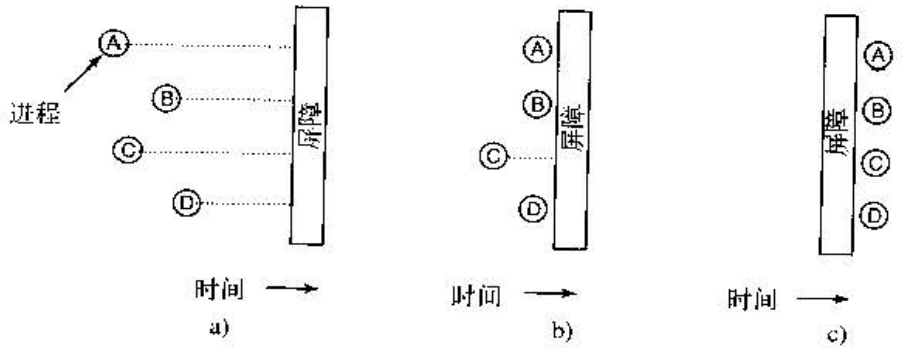

图2-37 屏障的使用：a) 进程接近屏障；b) 除了一个之外所有的进程都被屏障阻塞；c) 当最后一个进程到达屏障时，所有的进程一起通过

现在，我们设想这个矩阵非常之大（比如100万行乘以100万列），所以需要并行处理（可能在一台多处理器上）以便加速运算。各个进程工作在这个矩阵的不同部分，并且从老的矩阵按照物理定律计算新的矩阵元素。但是，除非第n次迭代已经完成，也就是说，除非所有的进程都完成了当前的工作，否则没有进程可以开始第$n+1$次迭代。实现这一目标的方法是通过编程使每一个进程在完成当前迭代部分后执行一个barrier操作。只有当全部进程完成工作之后，新的矩阵（下一次迭代的输入）才会完成，此时所有的进程会被释放而开始新的迭代过程。

2.4 调度

当计算机系统是多道程序设计系统时，通常就会有多个进程或线程同时竞争CPU。只要有两个或更多的进程处于就绪状态，这种情形就会发生。如果只有一个CPU可用，那么就必须选择下一个要运行的进程。在操作系统中，完成选择工作的这一部分称为*调度程序*（scheduler），该程序使用的算法称为*调度算法*（scheduling algorithm）。

尽管有一些不同，但许多适用于进程调度的处理方法也同样适用于线程调度。当内核管理线程的时候，调度经常是按线程级别的，与线程所属的进程基本或根本没有关联。下面我们将首先关注适用于进程与线程两者的调度问题，然后会明确地介绍线程调度以及它所产生的独特问题。第8章将讨论多核芯片的问题。

2.4.1 调度介绍

让我们回到早期以磁带上的卡片作为输入的批处理系统时代，那时的调度算法很简单：依次运行磁带上的每一个作业。对于多道程序设计系统，调度算法要复杂一些，因为经常有多个用户等候服务。有些大型机系统仍旧将批处理和分时服务结合使用，需要调度程序决定下一个运行的是一个批处理作业还是终端上的一个交互用户。（顺便提及，一个批处理作业可能需要连续运行多个程序，不过在本节中，我们假设它只是一个运行单个程序的请求。）由于在这些机器中，CPU是稀缺资源，所以好的调度程序可以在提高性能和用户的满意度方面取得很大的成果。因此，大量的研究工作都花费在创造聪明而有效的调度算法上了。

在拥有了个人计算机的优势之后，整个情形向两个方面发展。首先，在多数时间内只有一个活动进程。一个用户进入文字处理软件编辑一个文件时，一般不会同时在后台编译一个程序。在用户向文字处理软件键入一条命令时，调度程序不用做多少工作来判定哪个进程要运行——惟一的候选者是文字处理软件。

其次，同CPU是稀缺资源时的年代相比，现在计算机速度极快。个人计算机的多数程序受到的是用户当前输入速率（键入或敲击鼠标）的限制，而不是CPU处理速率的限制。即便对于编译（这是过去CPU周期的主要消耗者）现在大多数情况下也只要花费仅仅几秒钟。甚至两个实际同时运行的程序，诸如一个文字处理软件和一个电子表单，由于用户在等待两者完成工作，因此很难说需要哪一个先完成。这样的结果是，调度程序在简单的PC机上并不重要。当然，总有应用程序会实际消耗掉CPU，例如，为绘制一小时高精度视频而调整108 000帧（NTSC制）或90 000帧（PAL制）中的每一帧颜色就需要大量工业强度的计算能力。然而，类似的应用程序不在我们的考虑范围。