栈，用来记录执行历史，其中每一帧保存了一个已调用的但是还没有从中返回的过程。尽管线程必须在某个进程中执行，但是线程和它的进程是不同的概念，并且可以分别处理。进程用于把资源集中到一起，而线程则是在CPU上被调度执行的实体。

线程给进程模型增加了一项内容，即在同一个进程环境中，允许彼此之间有较大独立性的多个线程执行。在同一个进程中并行运行多个线程，是对在同一台计算机上并行运行多个进程的模拟。在前一种情形下，多个线程共享同一个地址空间和其他资源。而在后一种情形中，多个进程共享物理内存、磁盘、打印机和其他资源。由于线程具有进程的某些性质，所以有时被称为轻量级进程（lightweight process）。多线程这个术语，也用来描述在同一个进程中允许多个线程的情形。正如我们在第1章中看到的，一些CPU已经有直接硬件支持多线程，并允许线程切换在纳秒级完成。

在图2-11a中，可以看到三个传统的进程。每个进程有自己的地址空间和单个控制线程。相反，在图2-11b中，可以看到一个进程带有三个控制线程。尽管在两种情形中都有三个线程，但是在图2-11a中，每一个线程都在不同的地址空间中运行，而在图2-11b中，这三个线程全部在相同的地址空间中运行。

当多线程进程在单CPU系统中运行时，线程轮流运行。在图2-1中，我们已经看到了进程的多道程序设计是如何工作的。通过在多个进程之间来回切换，系统制造了不同的顺序进程并行运行的假象。多线程的工作方式也是类似的。CPU在线程之间的快速切换，制造了线程并行运行的假象，好似它们在一个比实际CPU慢一些的CPU上同时运行。在一个有三个计算密集型线程的进程中，线程以并行方式运行，每个线程在一个CPU上得到了真实CPU速度的三分之一。

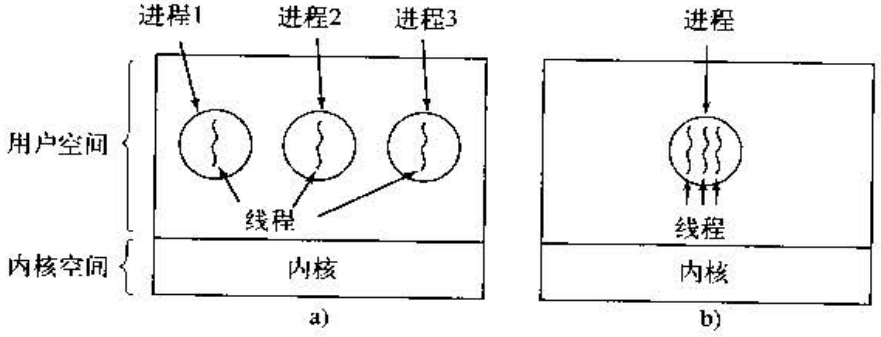

图2-11 a) 三个进程，每个进程有一个线程；b) 一个进程带三个线程

进程中的不同线程不像不同进程之间那样存在很大的独立性。所有的线程都有完全一样的地址空间，这意味着它们也共享同样的全局变量。由于各个线程都可以访问进程地址空间中的每一个内存地址，所以一个线程可以读、写或甚至清除另一个线程的堆栈。线程之间是没有保护的，原因是1）不可能，2）也没有必要。这与不同进程是有差别的。不同的进程会来自不同的用户，它们彼此之间可能有敌意，一个进程总是由某个用户所拥有，该用户创建多个线程应该是为了它们之间的合作而不是彼此间争斗。除了共享地址空间之外，所有线程还共享同一个打开文件集、子进程、报警以及相关信号等，如图2-12所示。这样，对于三个没有关系的线程而言，应该使用图2-11a的结构，而在三个线程实际完成同一个作业，并彼此积极密切合作的情形中，图2-11b则比较合适。

每个进程中的内容	每个线程中的内容
地址空间	程序计数器
全局变量	寄存器
打开文件	堆栈
子进程	状态
即将发生的报警	
信号与信号处理程序	
账户信息	

图2-12 第一列给出了在一个进程中所有线程共享的内容，第二列给出了每个线程自己的内容

图2-12中，第一列表项是进程的属性，而不是线程的属性。例如，如果一个线程打开了一个文件，