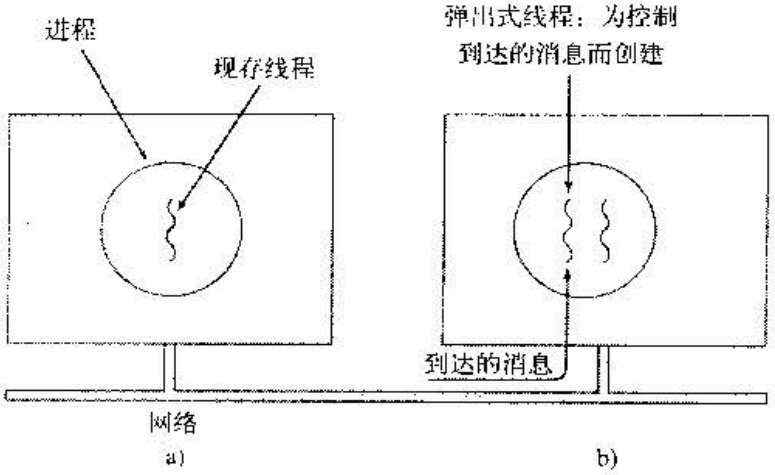

图2-18 在消息到达时创建一个新的线程：a) 消息到达之前；b) 消息到达之后

在使用弹出式线程之前，需要提前进行计划。例如，哪个进程中的线程先运行？如果系统支持在内核上下文中运行线程，线程就有可能在那里运行（这是图2-18中没有画出内核的原因）。在内核空间中运行弹出式线程通常比在用户空间中容易且快捷，而且内核空间中的弹出式线程可以很容易访问所有的表格和I/O设备，这些也许在中断处理时有用。而另一方面，出错的内核线程会比出错的用户线程造成更大的损害。例如，如果某个线程运行时间太长，又没有办法抢占它，就可能造成进来的信息丢失。

2.2.9 使单线程代码多线程化

许多已有的程序是为单线程进程编写的。把这些程序改写成多线程需要比直接写多线程程序更高的技巧。下面我们考察一些其中易犯的错误。

先考察代码，一个线程的代码就像进程一样，通常包含多个过程，会有局部变量、全局变量和过程参数。局部变量和参数不会引起任何问题，但是有一个问题是，对线程而言是全局变量，并不是对整个程序也是全局的。有许多变量之所以是全局的，是因为线程中的许多过程都使用它们（如同它们也可能使用任何全局变量一样），但是其他线程在逻辑上和这些变量无关。

作为一个例子，考虑由UNIX维护的errno变量。当进程（或线程）进行系统调用失败时，错误码会放入errno。在图2-19中，线程1执行系统调用access以确定是否允许它访问某个特定文件。操作系统把返回值放到全局变量errno里。当控制权返回到线程1之后，并在线程1读取errno之前，调度程序确认线程1此刻已用完CPU时间，并决定切换到线程2。线程2执行一个open调用，结果失败，导致重写errno，于是给线程1的返回值会永远丢失。随后在线程1执行时，它将读取错误的返回值并导致错误操作。

对于这个问题有各种解决方案。一种解决方案是全面禁止全局变量。不过这个想法不一定合适，因为它同许多已有的软件冲突。另一种解决方案是为每个线程赋予其私有的全局变量，如图2-20所示。在这个方案中，每个线程有自己的errno以及其他全局变量的私有副本，这样就避免了冲突。在效果上，这

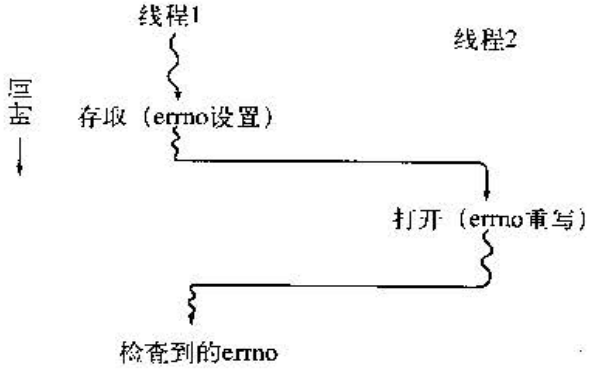

图2-19 线程使用全局变量所引起的错误

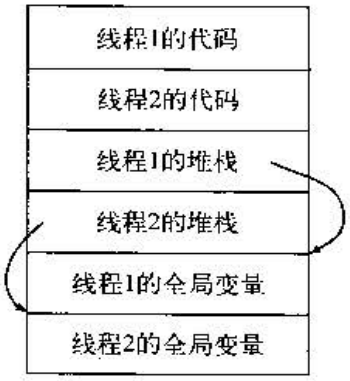

图2-20 线程可拥有私有的全局变量