需要多线程的第三个原因涉及性能方面的讨论。若多个线程都是CPU密集型的，那么并不能获得性能上的增强，但是如果存在着大量的计算和大量的I/O处理，拥有多个线程允许这些活动彼此重叠进行，从而会加快应用程序执行的速度。

最后，在多CPU系统中，多线程是有益的，在这样的系统中，真正的并行有了实现的可能。我们会在第8章讨论这个主题。

通过考察一些典型例子，我们就可以更清楚地看出多线程的有益之处。作为第一个例子，考虑一个字处理软件。字处理软件通常按照出现在打印页上的格式在屏幕上精确显示文档。特别地，所有的行分隔符和页分隔符都在正确的最终位置上，这样在需要时用户可以检查和修改文档（比如，消除孤行——在一页上不完整的顶部行和底部行，因为这些行不甚美观）。

假设用户正在写一本书。从作者的观点来看，最容易的方法是把整本书作为一个文件，这样一来，查询内容、完成全局替换等都非常容易。另一种方法是，把每一章都处理成单独一个文件。但是，在把每个小节和子小节都分成单个的文件之后，若必须对全书进行全局的修改时，那就真是麻烦了，因为有成百个文件必须一个个地编辑。例如，如果所建议的某个标准xxxx正好在书付印之前被批准了，于是“标准草案xxxx”一类的字眼就必须改为“标准xxxx”。如果整本书是一个文件，那么只要一个命令就可以完成全部的替换处理。相反，如果一本书分成了300个文件，那么就必须分别对每个文件进行编辑。

现在考虑，如果有一个用户突然在一个有800页的文件的第一页上删掉了一个语句之后，会发生什么情形。在检查了所修改的页面并确认正确后，这个用户现在打算接着在第600页上进行另一个修改，并键入一条命令通知字处理软件转到该页面（可能要查阅只在那里出现的一个短语）。于是字处理软件被强制对整个书的前600页重新进行格式处理，这是因为在排列该页前面的所有页面之前，字处理软件并不知道第600页的第一行应该在哪里。而在第600页的页面可以真正在屏幕上显示出来之前，计算机可能要拖延相当一段时间，从而令用户不甚满意。

多线程在这里可以发挥作用。假设字处理软件被编写成含有两个线程的程序。一个线程与用户交互，而另一个在后台重新进行格式处理。一旦在第1页中的语句被删除掉，交互线程就立即通知格式化线程对整本书重新进行处理。同时，交互线程继续监控键盘和鼠标，并响应诸如滚动第1页之类的简单命令，此刻，另一个线程正在后台疯狂地运算。如果有点运气的话，重新格式化会在用户请求查看第600页之前完成，这样，第600页页面就立即可以在屏幕上显示出来。

如果我们已经做到了这一步，那么为什么不再进一步增加一个线程呢？许多字处理软件都有每隔若干分钟自动在磁盘上保存整个文件的特点，用于避免由于程序崩溃、系统崩溃或电源故障而造成用户一整天的工作丢失的情况。第三个线程可以处理磁盘备份，而不必干扰其他两个线程。拥有三个线程的情形，如图2-7所示。

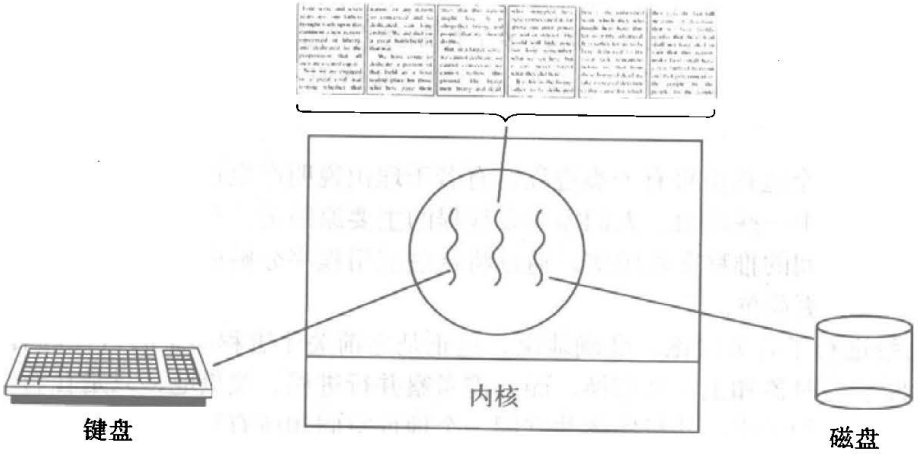

图2-7 有三个线程的字处理软件

如果程序是单线程的，那么在进行磁盘备份时，来自键盘和鼠标的命令就会被忽略，直到备份工作完成为止。用户当然会认为性能很差。另一个方法是，为了获得好的性能，可以让键盘和鼠标事件中断