磁盘备份，但这样却引入了复杂的中断驱动程序设计模型。如果使用三个线程，程序设计模型就很简单了。第一个线程只是和用户交互；第二个线程在得到通知时进行文档的重新格式化；第三个线程周期性地将RAM中的内容写到磁盘上。

很显然，在这里用三个不同的进程是不能工作的，这是因为三个线程都需要在同一个文件上进行操作。通过让三个线程代替三个进程，三个线程共享公共内存，于是它们都可以访问同一个正在编辑的文件。

许多其他的交互式程序中也存在类似的情形。例如，电子表格是允许用户维护矩阵的一种程序，矩阵中的一些元素是用户提供的数据；另一些元素是通过所输入的数据运用可能比较复杂的公式而得出的计算结果。当用户改变一个元素时，许多其他元素就必须重新计算。通过一个后台线程进行重新计算的方式，交互式线程就能够在进行计算的时候，让用户从事更多的工作。类似地，第三个线程可以在磁盘上进行周期性的备份工作。

现在考虑另一个多线程发挥作用的例子：一个万维网服务器。对页面的请求发给服务器，而所请求的页面发回给客户机。在多数Web站点上，某些页面较其他页面相比，有更多的访问。例如，对Sony主页的访问就远远超过对深藏在页面树里的任何特定摄像机的技术说明书页面的访问。利用这一事实，Web服务器可以把获得大量访问的页面集合保存在内存中，避免到磁盘去调入这些页面，从而改善性能。这样的一种页面集合称为*高速缓存*（cache），高速缓存也运用在其他许多场合中。例如在第1章中介绍的CPU缓存。

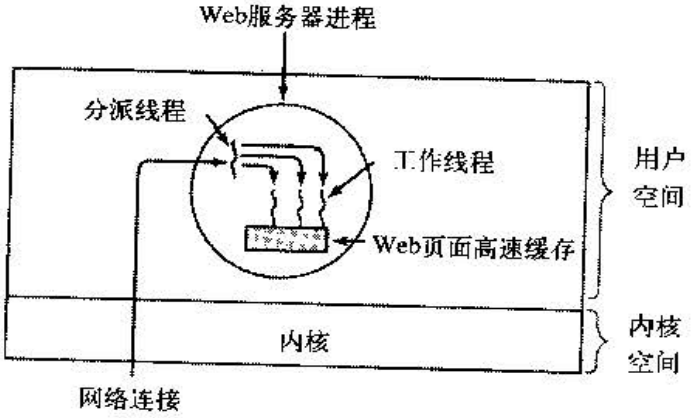

图2-8 一个多线程的Web服务器

一种组织Web服务器的方式如图2-8所示。在这里，一个称为*分派程序*（dispatcher）的线程从网络中读入工作请求。在检查请求之后，分派线程挑选一个空转的（即被阻塞的）*工作线程*（worker thread），提交该请求，通常是在每个线程所配有的某个专门字中写入一个消息指针。接着分派线程唤醒睡眠的工作线程，将它从阻塞状态转为就绪状态。

在工作线程被唤醒之后，它检查有关的请求是否在Web页面高速缓存之中，这个高速缓存是所有线程都可以访问的。如果没有，该线程开始一个从磁盘调入页面的read操作，并且阻塞直到该磁盘操作完成。当上述线程阻塞在磁盘操作上时，为了完成更多的工作，分派线程可能挑选另一个线程运行，也可能把另一个当前就绪的工作线程投入运行。

这种模型允许把服务器编写为顺序线程的一个集合。在分派线程的程序中包含一个无限循环，该循环用来获得工作请求并且把工作请求派给工作线程。每个工作线程的代码包含一个从分派线程接收请求，并且检查Web高速缓存中是否存在所需页面的无限循环。如果存在，就将该页面返回给客户机，接着该工作线程阻塞，等待一个新的请求。如果没有，工作线程就从磁盘调入该页面，将该页面返回给客户机，然后该工作线程阻塞，等待一个新的请求。

图2-9给出了有关代码的大致框架。如同本书的其他部分一样，这里假设TRUE为常数1。另外，buf和page分别是保存工作请求和Web页面的相应结构。

```
while (TRUE) {
    get_next_request(&buf);
    handoff_work(&buf);
}
```

a)

```
while (TRUE) {
    wait_for_work(&buf)
    look_for_page_in_cache(&buf, &page);
    if (page_not_in_cache(&page))
        read_page_from_disk(&buf, &page);
    return_page(&page);
}
```

b)

图2-9 对应图2-8的代码概要：a) 分派线程；b) 工作线程