（如浏览器）向该服务器发送报文段时，所有报文段的目的端口都将为80。特别是，初始连接建立报文段和承载HTTP请求的报文段都有80的目的端口。如我们刚才描述的那样，该服务器能够根据源IP地址和源端口号来区分来自不同客户的报文段。

图3-5显示了一台Web服务器为每条连接生成一个新进程。如图3-5所示，每个这样的进程都有自己的连接套接字，通过这些套接字可以收到HTTP请求和发送HTTP响应。然而，我们要提及的是，连接套接字与进程之间并非总是有着一一对应的关系。事实上，当今的高性能Web服务器通常只使用一个进程，但是为每个新的客户连接创建一个具有新连接套接字的新线程。（线程可被看作是一个轻量级的子进程。）如果做了第2章的第一个编程作业，你所构建的Web服务器就是这样工作的。对于这样一台服务器，在任意给定的时间内都可能有（具有不同标识的）许多连接套接字连接到相同的进程。

如果客户与服务器使用持续HTTP，则在整条连接持续期间，客户与服务器之间经由同一个服务器套接字交换HTTP报文。然而，如果客户与服务器使用非持续HTTP，则对每一对请求/响应都创建一个新的TCP连接并在随后关闭，因此对每一对请求/响应创建一个新的套接字并在随后关闭。这种套接字的频繁创建和关闭会严重地影响一个繁忙的Web服务器的性能（尽管有许多操作系统技巧可用来减轻这个问题的影响）。读者若对与持续和非持续HTTP有关的操作系统问题感兴趣的话，可参见［Nielsen 1997，Nahum 2002］。

既然我们已经讨论过了运输层多路复用与多路分解问题，下面我们就继续讨论因特网运输层协议之一，即UDP。在下一节中，我们将看到UDP无非就是对网络层协议增加了一点（多路）复用/（多路）分解服务而已。

3.3 无连接运输：UDP

在本节中，我们要仔细地研究一下UDP，看它是怎样工作的，能做些什么。我们鼓励你回过来看一下2.1节的内容，其中包括了UDP服务模型的概述，再看看2.7.1节，其中讨论了UDP上的套接字编程。

为了激发我们讨论UDP的热情，假如你对设计一个不提供不必要服务的最简化的运输层协议感兴趣。你将打算怎样做呢？你也许会首先考虑使用一个无所事事的运输层协议。特别是在发送方一侧，你可能会考虑将来自应用进程的数据直接交给网络层；在接收方一侧，你可能会考虑将从网络层到达的报文直接交给应用进程。而正如我们在前一节所学的，我们必须做一点点事，而不是什么都不做！运输层最低限度必须提供一种复用/分解服务，以便在网络层与正确的应用级进程之间传递数据。

由［RFC 768］定义的UDP只是做了运输协议能够做的最少工作。除了复用/分解功能及少量的差错检测外，它几乎没有对IP增加别的东西。实际上，如果应用程序开发人员选择UDP而不是TCP，则该应用程序差不多就是直接与IP打交道。UDP从应用进程得到数据，附加上用于多路复用/分解服务的源和目的端口号字段，以及两个其他的小字段，然后将形成的报文段交给网络层。网络层将该运输层报文段封装到一个IP数据报中，然后尽力而为地尝试将此报文段交付给接收主机。如果该报文段到达接收主机，UDP使用目的端口号将报文段中的数据交付给正确的应用进程。值得注意的是，使用UDP时，在发送报文段之前，发送方和接收方的运输层实体之间没有握手。正因为如此，UDP被称为是*无连接的*。

DNS是一个通常使用UDP的应用层协议的例子。当一台主机中的DNS应用程序想要