TCP/IP 应用程序的通信连接模式

本文的作者通过分析 TCP/IP 程序在不同级别上采用的不同方式来向您讲述了如何设计好 TCP/IP 应用程序的通信模式以及需要注意的相关问题。

刘 光宝 (liugb@cn.ibm.com), 软件工程师, IBM

2008 年 7 月 10 日

3 评论

内容

TCP/IP 应用层与应用程序

TCP/IP 起源于二十世纪 60 年代末美国政府资助的一个分组交换网络研究项目，它是一个真正的开放协议，很多不同厂家生产各种型号的计算机，它们运行完全不同的操作系统，但 TCP/IP 协议组件允许它们互相进行通信。现在 TCP/IP 已经从一个只供一些科学家使用的小实验网成长为一个由成千上万的计算机和用户构成的全球化网络，TCP/IP 也已成为全球因特网（Internet）的基础，越来越多的 TCP/IP 互联网应用和企业商业应用正在改变着世界。

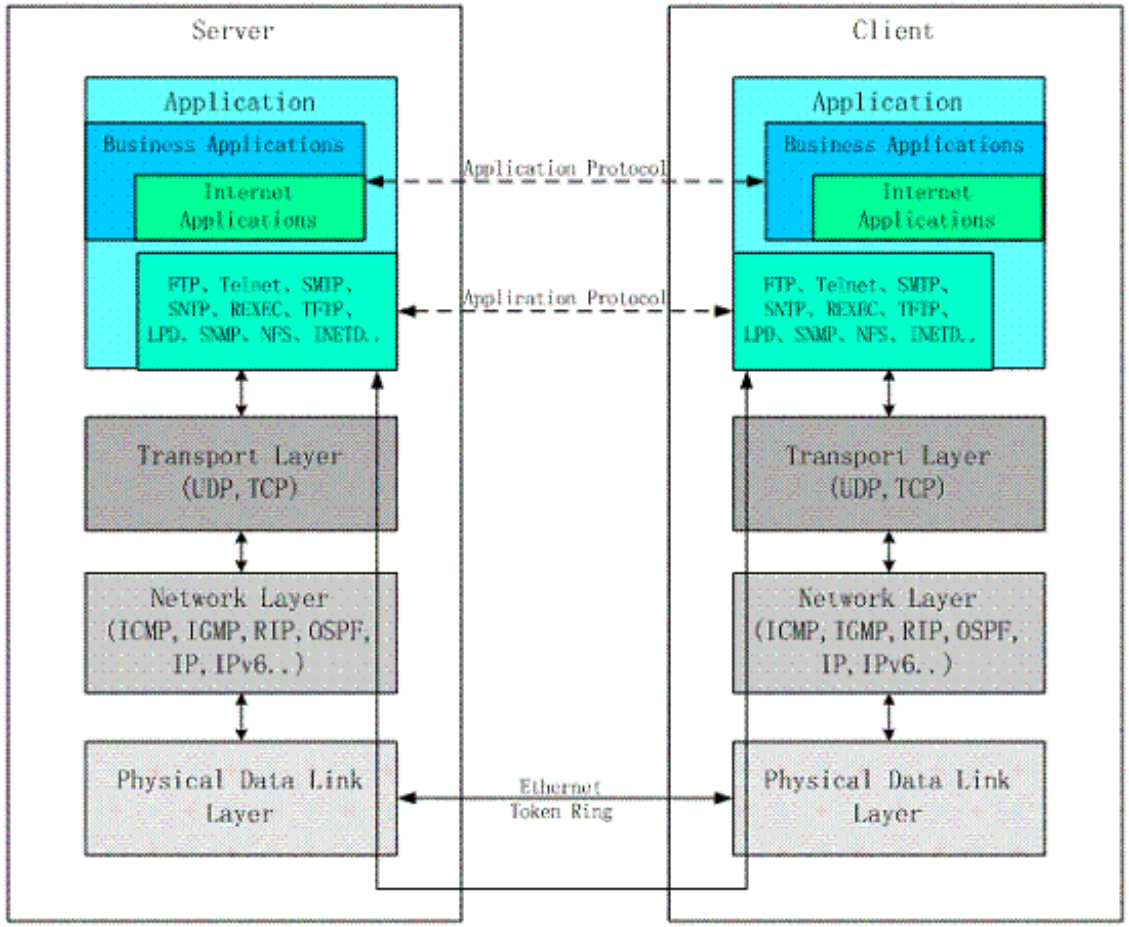
TCP/IP 通讯协议采用了四层的层级模型结构（注：这与 OSI 七层模型不相同），每一层都调用它的下一层所提供的网络任务来完成自己的需求。TCP/IP 的每一层都是由一系列协议来定义的。这 4 层分别为：

- **应用层 (Application)**：应用层是个很广泛的概念，有一些基本相同的系统级 TCP/IP 应用以及应用协议，也有许多的企业商业应用和互联网应用。
- **传输层 (Transport)**：传输层包括 UDP 和 TCP，UDP 几乎不对报文进行检查，而 TCP 提供传输保证。
- **网络层 (Network)**：网络层协议由一系列协议组成，包括 ICMP、IGMP、RIP、OSPF、IP(v4,v6) 等。
- **链路层 (Link)**：又称为物理数据网络接口层，负责报文传输。

图 1 显示了 TCP/IP 层级模型结构，应用层之间的协议通过逐级调用传输层（Transport layer）、网络层（Network Layer）和物理数据链路层（Physical Data Link）而可以实现应用层的应用程序通信互联。

应用层需要关心应用程序的逻辑细节，而不是数据在网络中的传输活动。应用层其下三层则处理真正的通信细节。在 Internet 整个发展过程中的所有思想和着重点都以一种称为 RFC（Request For Comments）的文档格式存在。针对每一种特定的 TCP/IP 应用，有相应的 RFC 文档。一些典型的 TCP/IP 应用有 FTP、Telnet、SMTP、SNTP、REXEC、TFTP、LPD、SNMP、NFS、INETD 等。RFC 使一些基本相同的 TCP/IP 应用程序实现了标准化，从而使得不同厂家开发的应用程序可以互相通信。

图 1 TCP/IP 层级模型结构



然而除了这些已经实现标准化的系统级 TCP/IP 应用程序外，在企业商业应用和互联网应用开发中，存在着大量的商业应用程序通信互联问题。如图 1 显示，其中的应用层所包含应用程序主要可以分成两类，即系统级应用和商业应用，互联网商业应用是商业应用中的主要形式之一。

不同开发商和用户在开发各自商业应用通信程序时也存在有许多不同的设计方式。关于 TCP/IP 应用层以下的技术文献与书籍早已是汗牛充栋，

但是关于 TCP/IP 应用本身，尤其是关于商业应用的通信设计模式技术讨论方面的文章还是比较少的。TCP/IP 应用通信设计模式实际上是在 TCP/IP 基础编程之上的一种应用编程设计方式，也属于一种应用层协议范畴，其可以包含有 TCP/IP 地址族模式设计、I/O 模式设计、通信连接模式设计以及通信数据格式设计等。鉴于目前讨论 TCP/IP 商业应用程序设计模式问题这方面的文章还很少见，本文尝试给出一些通信连接模式设计中共同的概念与一些典型的设计模式，在以后的文章中将继续讨论地址族模式设计、I/O 模式设计、以及通信数据格式设计等方面的模式设计实现话题。

通信连接模式设计主要考虑内容有：

- 通信两端程序建立通信方式
- 通信连接方式
- 通信报文发送与接收方式

以下内容将介绍建立通信的 Client/Server 模型，然后逐一介绍通信连接模式设计所需要考虑的这些内容。

↑回页首

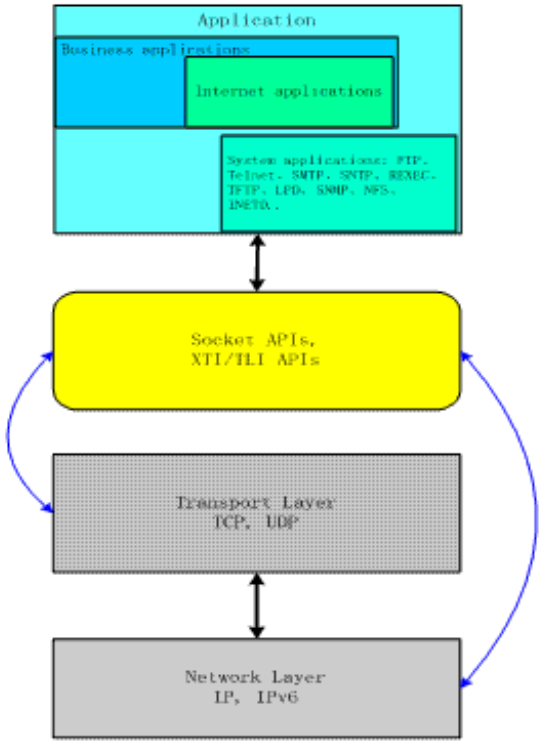
传输层接口 APIs 与 TCP/IP 应用程序 C/S 模型

传输层接口 APIs

TCP/IP 应用层位于传输层之上，TCP/IP 应用程序需要调用传输层的接口才能实现应用程序之间通信。目前使用最广泛的传输层的应用编程接口是套接字接口（Socket）。Socket APIs 是于 1983 年在 Berkeley Socket Distribution (BSD) Unix 中引进的。 1986 年 AT&T 公司引进了另一种不同的网络层编程接口 TLI（Transport Layer Interface），1988 年 AT&T 发布了一种修改版的 TLI，叫做 XTI（X/open Transport interface）。XTI/TLI 和 Socket 是用来处理相同任务的不同方法。关于 TCP/IP APIs 使用文章与书籍已相当多，本文则是侧重于如何组合使用这些 APIs 来进行 TCP/IP 应用程序连接模式设计，并归纳出几种基本应用连接模式。

如图 2 显示，应用层是通过调用传输层接口 APIs（Socket 或 XTI/TLI）来与传输层和网络层进行通信的。

图 2 传输层接口



不管是使用何种编程接口，要在两个机器或两个程序之间建立通信，通信双方必须建立互相一致的通信模式。如果双方的通信设计模式不一致就无法建立有效的通信连接。

以下是经常使用的 socket APIs，是建立 TCP/IP 应用程序的标准接口，也是影响 TCP/IP 应用程序通信方式的几个主要 APIs，不同 APIs 组合再结合系统调用可以实现不同方式的应用。Sockets 支持多种传输层和网络层协议，支持面向连接和无连接的数据传输，允许应用分布式工作。

- socket()：是用来创建一个 socket，socket 表示通信中的一个节点，其可以在一个网络中被命名，用 socket 描述符表示，socket 描述符类似于 Unix 中的文件描述符。
- bind()：是用来把本地 IP 层地址和 TCP 层端口赋予 socket。
- listen()：把未连接的 socket 转化成一个等待可连接的 socket，允许该 socket 可以被请求连接，并指定该 socket 允许的最大连接数。
- accept()：是等待一个连接的进入，连接成功后，产生一个新的 socket 描述符，这个新的描述符用来建立与客户端的连接。
- connect()：用来建立一个与服务端的连接。
- send()：发送一个数据缓冲区，类似 Unix 的文件函数 write()。另外 sendto() 是用在无连接的 UDP 程序中，用来发送自带寻址信息的数据包。
- recv()：接收一个数据缓冲区，类似 Unix 的文件函数 readl()。另外 recvfrom() 是用在无连接的 UDP 程序中，用来接收自带寻址信息的数据

包。

- close()：关闭一个连接

Client/Server 模型

Sockets 是以 Client 和 Server 交互通信方式来使用的。典型的系统配置是把 Server 放在一台机器中，而把 Client 放在另一台机器中，Client 连接到 Server 交换信息。一个 socket 有一系列典型的事件流。例如，在面向连接的 Client/Server 模型中，Server 端的 socket 总是等待一个 Client 端的请求。要实现这个请求，Server 端首先需要建立能够被 Client 使用的地址，当地址建立后，Server 等待 Client 请求服务。当一个 Client 通过 socket 连接到 Server 后，Client 与 Server 之间就可以进行信息交换。Client/Server 是通信程序设计的基本模式。从软件开发的角度讲，TCP/IP 应用程序都是基于 Client/Server 方式的。注意本篇文章以下 Client/Server 概念是针对程序内部调用 Socket API 所讲的概念，与针对整个程序甚至针对机器而讲的客户端 / 服务器概念有所不同。用 Server APIs 建立的程序可以被当作客户端使用，用 Client APIs 建立的程序也可以被用作服务器端使用。建立 Server 需要的 APIs 有 socket(), bind(), listen(), accept()，建立 Client 需要的 APIs 有 Socket(), Connect()。在实际应用开发中，同一个程序里往往同时可以有 Client 和 Server 的代码，或者多种形式的组合。在实际应用编程中，针对 Socket APIs 不同有效组合，结合系统调用可以有多种复杂的设计变化。

面向连接的应用编程存在三类基本的不同级别的设计方式范畴，根据 Socket APIs 从上到下顺序依次是：

- Client/Server 通信建立方式
- Client/Server 通信连接方式
- Client/Server 通信发送与接收方式

下面内容以面向连接的 Socket 应用编程为例来说明这几种不同通信范畴的设计实现。

回页首

Client/Server 建立方式设计概述

一个 Client 连接一个 Server

如果只有两台机器之间连接，那么一个是 Client，另一个是 Server，如下面图 3 所示。这是最简单的 TCP/IP 的应用，也是 TCP/IP 应用早期的 Peer to Peer (P2P) 概念。其流程基本如图 4 所示。

图 3 TCP/IP 应用单点 Client/Server

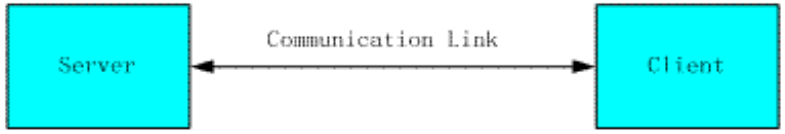
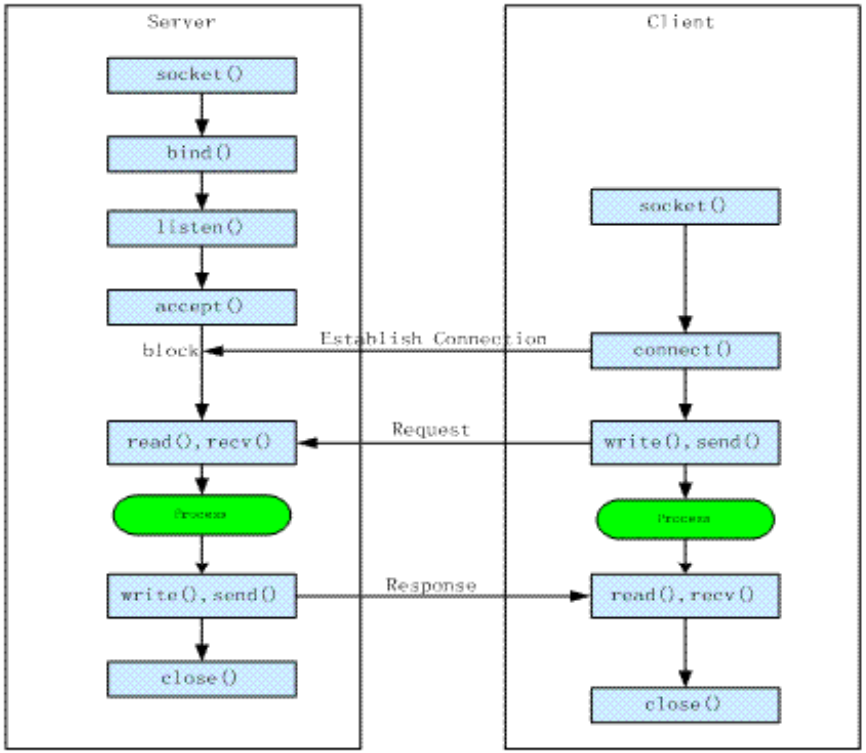


图 4 显示了 TCP/IP 应用编程最基本的 Client/Server 模式，显示了基本的 Client/Server 通信所需要调用的 Socket APIs 以及顺序。

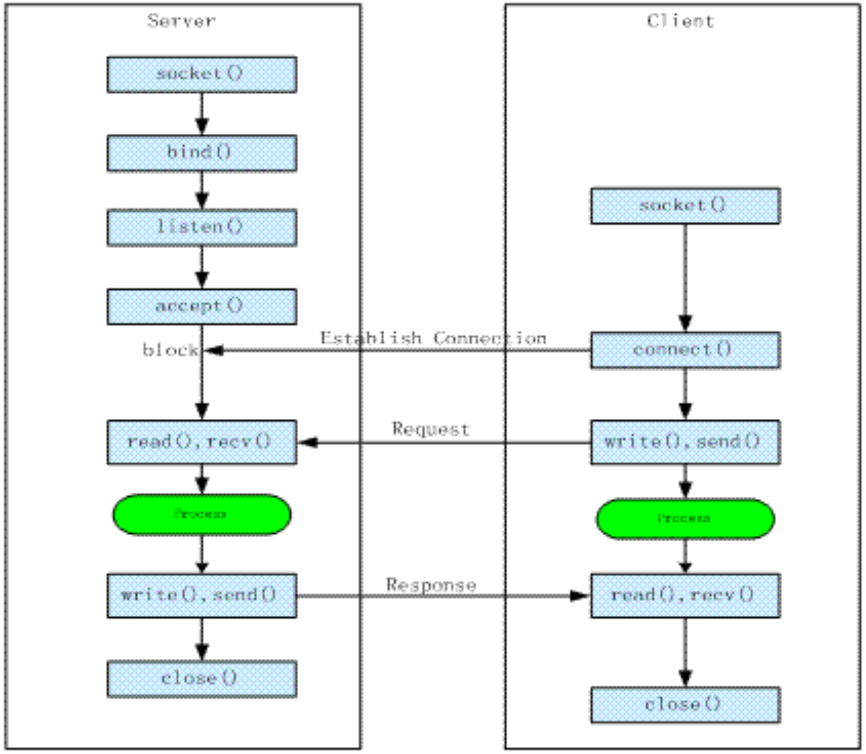
图 4 TCP/IP 应用编程基本 Client/Server 模式



多个 Client 连接一个 Server

多个 Client 同时连接一个 Server 是 TCP/IP 应用的主流形式，如图 5 所示，其中 Client 连接数可以从几个到成千上万。

图 5 TCP/IP 应用多 Client 端的 Client/Server

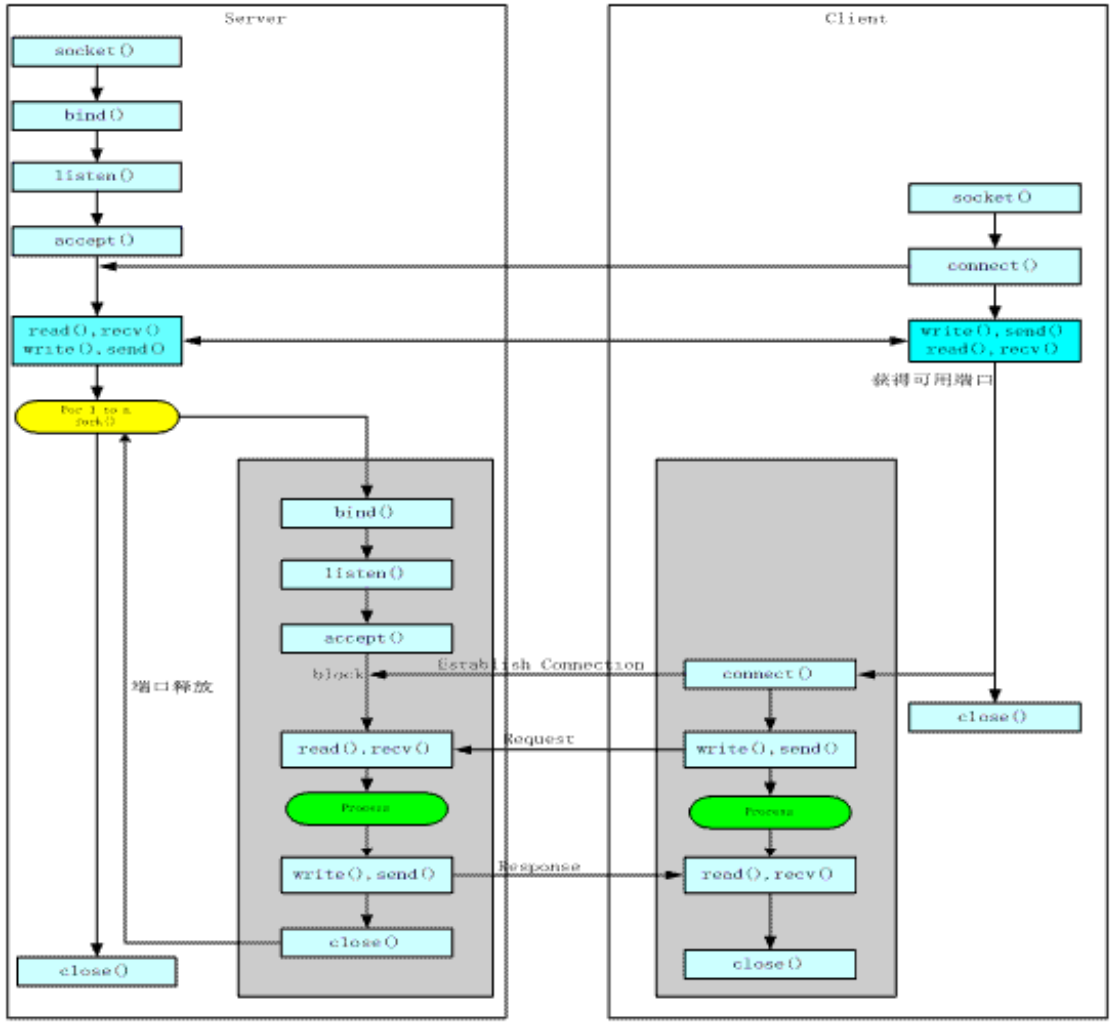


由于 socket APIs 缺省方式下都是阻塞方式的，实现多个 Client 同时连接一个 Server 就需要特别的设计。其实现方式可以有多种不同的设计，这其中也涉及 I/O 模式设计。下面将展开介绍其中几种设计形式。

利用一个 Client 连接一个 Server 形式实现多 Client 连接

从程序设计角度讲，只要 Client 和 Server 端口是一对一形式，那么就属于一个 Client 连接一个 Server 形式。在处理多个 Client 端连接时，Server 端轮流使用多个端口建立多个 Client-Server 连接，连接关闭后，被释放端口可以被循环使用。在这种多连接形式中需要谨慎处理 Client 端如何获取使用 Server 端的可用端口。比如图 6 显示 Server 有一个服务于所有进程的进程可以先把 Server 端的可用端口发送给 Client 端，Client 端再使用该端口建立连接来处理业务。Server 针对每一个 Client 连接用一个专门的进程来处理。由于可用端口数有限，Server 用一个有限循环来处理每一个可用的端口连接。由于新端口需要用 bind() 来绑定，所以需要从 bind() 开始到 close() 结束都需要包含在循环体内。

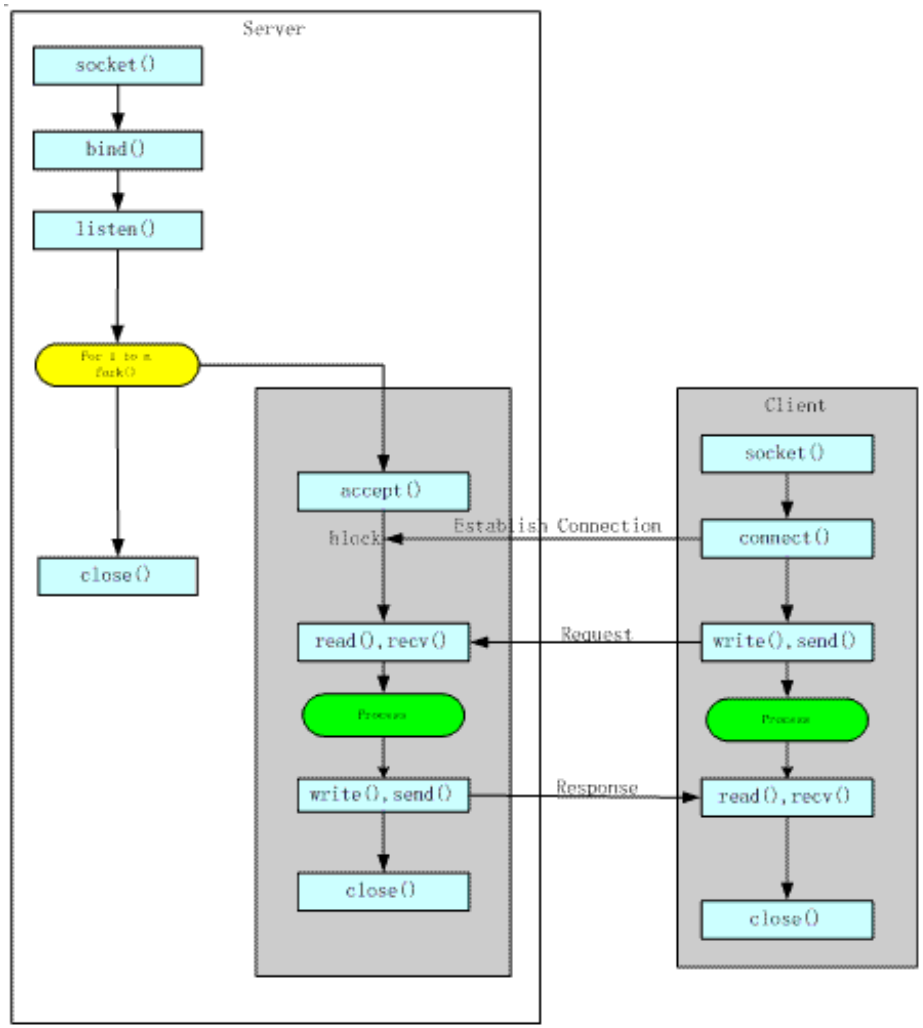
图 6 利用一对一 Client-Server 模式实现多 Client 连接



使用多个 **accept()** 实现多 **Client** 连接

多进程 Server 一般有一个专注进程是服务于每一个连接的。当 Client 端完成连接后，专注进程可以循环被另外的连接使用。使用多个 accept() 也可以实现处理多 Client 连接。多 accept() 的 Server 也只有一个 socket()，一个 bind()，一个 listen()，这与通常情况一样。但是它建立许多工作子进程，每一个工作子进程都有 accept()，这样可以为每一个 Client 建立 socket 描述符。如图 7 所示，由于 accept() 连接成功后，会产生一个新的 socket 描述符，这样通过循环多进程利用 accept() 产生的多 socket 描述符就可以与多个 Client 进行连接通信。循环体是从 accept() 开始到 close() 结束的。

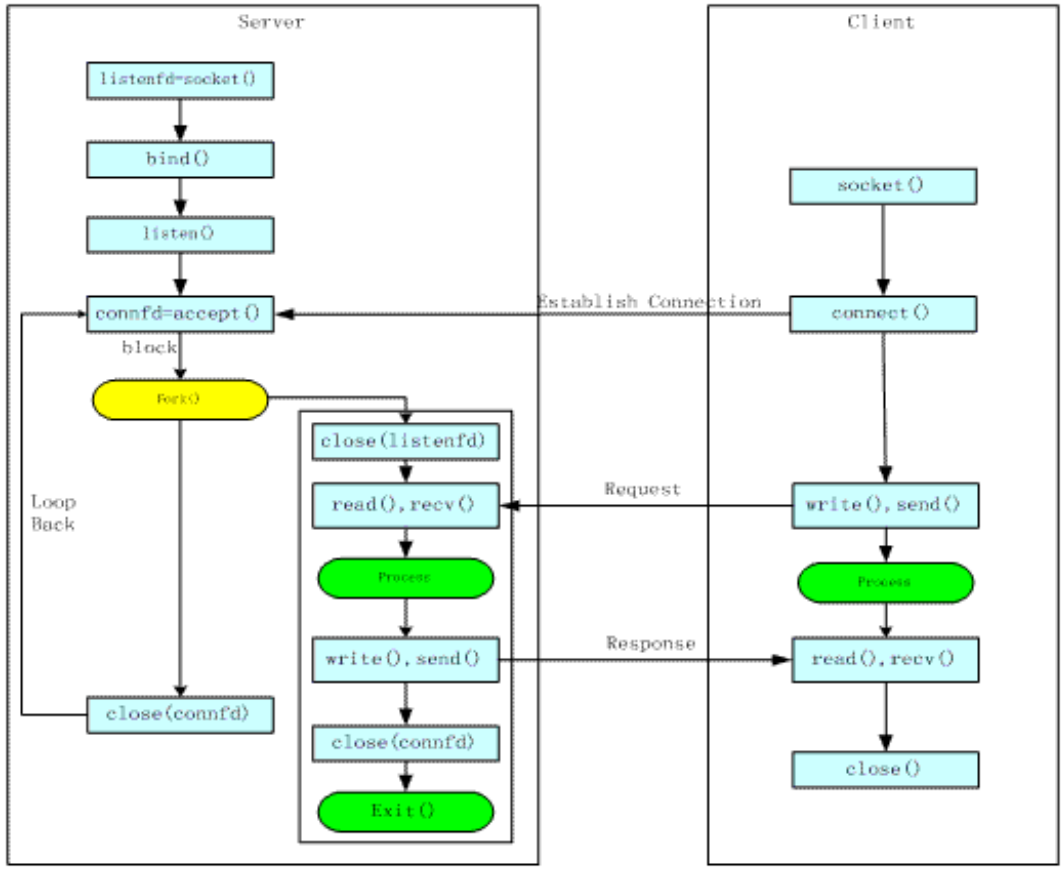
图 7 使用多 **accept()** 实现多 **Client** 连接



使用并发 **Server** 模式实现多 **Client** 连接

并发服务器模式曾经是 TCP/IP 的主流应用程序设计模式，得到广泛使用，目前互联网上仍有相当多的应用使用此种模式。其设计思路是在 accept 之后 fork 出一个子进程。因为 socket 会产生监听 socket 描述符 listenfd，accept 会产生连接 socket 描述符 connfd。连接建立后，子进程继承连接描述符服务于 Client，父进程则继续使用监听描述符等待另外一个 Client 的连接请求，以产生另外一个连接 socket 描述符和子进程。如图 8 所示，accept() 接收到一个 Client 连接后，产生一个新的 socket 描述符，通过 fork() 系统调用，用一个子进程来处理该 socket 描述符的连接服务。而父进程可以立即返回到 accept()，等待一个新的 Client 请求，这就是典型的并发服务器模式。并发服务器模式同时处理的最大并发 Client 连接数由 listen() 的第二个参数来指定。

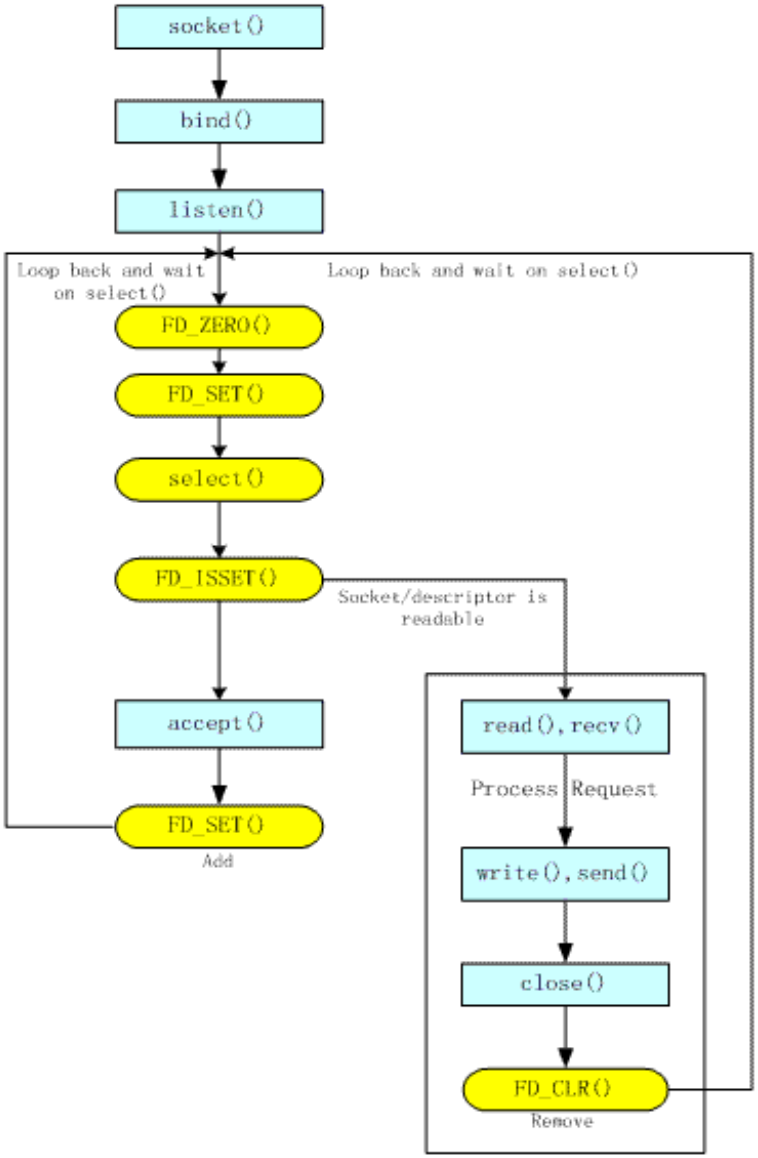
图 8 **TCP/IP** 应用并发 **Server**



使用 I/O 多路技术实现多 Client 连接

以上三种连接设计，多 Server 端口、多 accept() 和并发服务器模式，都是通过 fork() 系统调用产生多进程来实现多 Client 连接的。使用 I/O 多路技术也可以同时处理多个输入与输出问题，即用一个进程同时处理多个文件描述符。I/O 多路技术是通过 select() 或 poll() 系统调用实现的。poll() 与 select() 功能完全相同，但是 poll() 可以更少使用内存资源以及有更少的错误发生。select() 调用需要与操作文件描述符集的 APIs 配合使用。select() 系统调用可以使一个进程检测多个等待的 I/O 是否准备好，当没有设备准备好时，select() 处于阻塞状态中，其中任一设备准备好后，select() 函数返回调用。select() API 本身也有一个超时时间参数，超时时间到后，无论是否有设备准备好，都返回调用。其流程如图 9 所示。在 socket APIs listen() 和 accept() 之间插入 select() 调用。使用这三个宏 FD_ZERO()、FD_CLR() 和 FD_SET()，在调用 select() 前设置 socket 描述符屏蔽位，在调用 select() 后使用 FD_ISSET 来检测 socket 描述符集中对应于 socket 描述符的位是否被设置。 FD_ISSET() 就相当通知了一个 socket 描述符是否可以被使用，如果该 socket 描述符可用，则可对该 socket 描述符进行读写通信操作。通常，操作系统通过宏 FD_SETSIZE 来声明在一个进程中 select() 所能操作的文件或 socket 描述符的最大数目。更详细的 I/O 多路技术实现，可以参考其他相关文献。

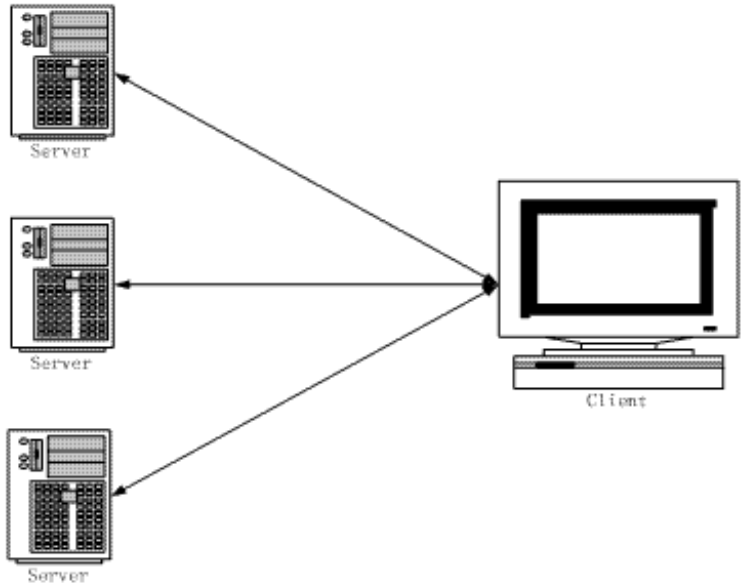
图 9 **I/O 多路技术实现多连接的 Server**



一个 **Client** 连接多个 **Server**

一个 Client 连接多个 Server 这种方式很少见，主要用于一个客户需要向多个服务器发送请求情况，比如一个 Client 端扫描连接多个 Server 端情况。如图 １０ 所示。此种方式设计主要是 Client 端应用程序的逻辑设计，通常需要在 Client 端设计逻辑循环来连接多个 Server，在此不做更多描述。

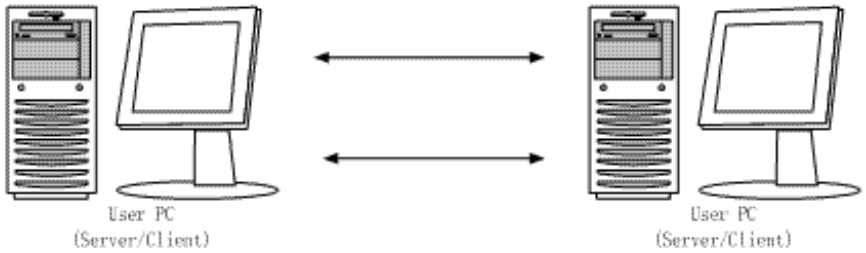
图 １０ 单 **Client** 对多 **Server**



复杂 Client/Server 设计与现代 P2P

最近几年，对等网络技术 (Peer-to-Peer，简称 P2P) 迅速成为计算机界关注的热门话题之一，以及影响 Internet 未来的科技之一。与早期点对点 (Peer to Peer) 的 Client/Server 模式不同，现在的 P2P 模式是指每个结点既可充当服务器，为其他结点提供服务，同时也可作为客户端享用其他结点提供的服务。实际上 P2P 模式仍然是基于 Client/Server 模式的，每个通信节点都既是 Server，又是 Client，P2P 是基于复杂 Client/Server 设计的 TCP/IP 应用。图 1 1 显示 P2P 模式下两个用户 PC 之间的对等连接。

图 1 1 **P2P** 模式



在技术上，P2P 本身是基于 TCP/IP Client/Server 技术的一种设计模式思想， P2P 也属于网络应用层技术，与 Web 和 FTP 等应用是并列的。只是 P2P 应用在设计实现上更要复杂的多。P2P 技术实现的协同工作是无需专门的服务器支持的 (Serverless)，这里的服务器概念与 Client/Server 中的 Server 概念是不一样的。在传统意义上中心服务器机器上往往运行的是 TCP/IP 应用的 Server 端程序，所以传统意义上的 Server 概念在机器与应用上是重合的。如果更改 TCP/IP 的应用设计，使应用程序既可做 Server 又可做 Client，就可以实现无中心服务器的 P2P 模式。

在设计模式上，P2P 模式实现了网络终端用户不依赖中心服务器或者服务商而直接进行信息和数据交换的可能，因此 P2P 正在改变着整个互联网的一些基础应用，从而极大地增加了用户之间的信息沟通和交流能力。目前互联网的 P2P 应用与网络都正在飞速发展，一些典型的 P2P 应用程序比如有 BitTorrent, eDonkey 等，另外一些即时通信（IM）类软件比如 MSN、QQ 等也正在向无中心服务器模式转变。无中心服务器的 Internet 应用程序大大降低应用提供商的运营成本，而且减少人们对于 Server 稳定性的依赖。

回页首

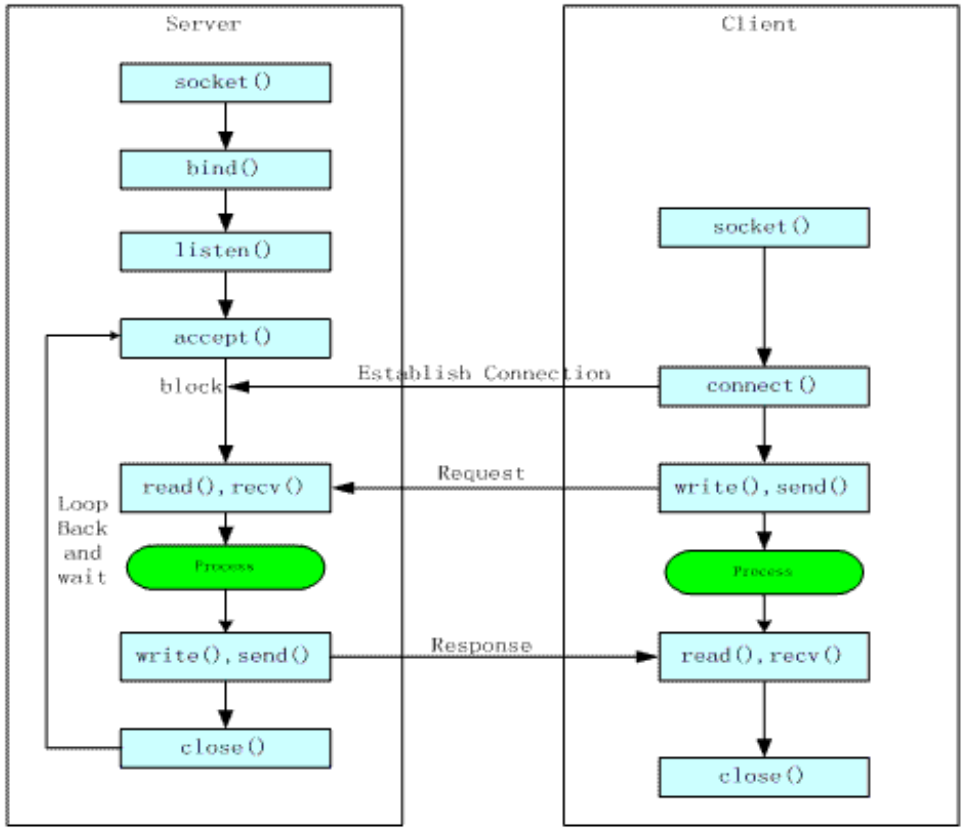
Client/Server 通信连接方式设计

Client/Server 通信方式建立后，下一步就需要考虑通信连接的方式，主要有两种方式的连接，即长连接通信与短连接通信。通信连接方式涉及到的 APIs 主要是 connect() 和 accept()。要实现某种 Client/Server 方式，就必须考虑用某种特定的连接方式。

短连接通信

短连接通信是指 Client 方与 Server 方每进行一次通信报文收发交易时才进行通讯连接，交易完毕后立即断开连接。此种方式常用于多个 Client 连接一个 Server 情况，常用于机构与用户之间通信，比如 OLTP（联机事务处理）类应用。在短连接情况下，Client 端完成任务后，就关闭连接并退出。在 Server 端，可以通过循环 accept()，使 Server 不会退出，并连续处理 Client 的请求。图 1 2 显示了一般情况下短连接通信模式的 Socket 事件流，不同设计的连接多 Client 的 Server 有不同的循环流程。

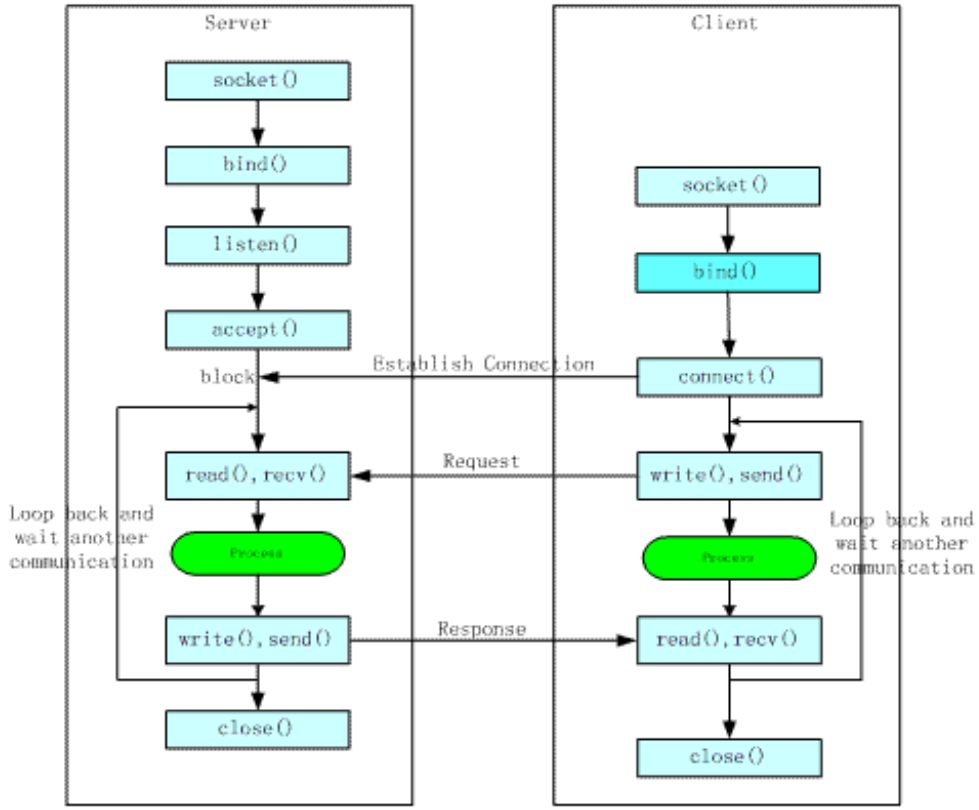
图 1 2 短连接模式通信



长连接通信

长连接通信是指 Client 方与 Server 方先建立通讯连接，连接建立后不会断开，然后再进行报文发送和接收，报文发送与接收完毕后，原来连接不会断开而继续存在，因此可以连续进行交易报文的发送与接收。这种方式下由于通讯连接一直存在，其 TCP/IP 状态是 Established，可以用操作系统的命令 netstat 查看连接是否建立。由于在长连接情况下，Client 端和 Server 端一样可以固定使用一个端口，所以长连接下的 Client 也需要使用 bind() 来绑定 Client 的端口。在长连接方式下，需要循环读写通信数据。为了区分每一次交易的通信数据，每一次交易数据常常需要在数据头部指定该次交易的长度，接收 API 需要首先读出该长度，然后再按该长度读出指定长度的字节。长连接方式常用于一个 Client 端对一个 Server 端的通讯，一般常用于机构与机构之间的商业应用通信，以处理机构之间连续的大量的信息数据交换。或者说可用于两个系统之间持续的信息交流情况。通常为了加快两个系统之间的信息交流，通常还需要建立几条长连接的并行通信线路。图 1 3 显示了一般情况下长连接通信模式的 socket 事件流，可见其最大特点是 Client 和 Server 都有循环体，而且循环体只包含读写 APIs。

图 1 3 长连接模式通信



回页首

Client/Server 通信发送与接收方式设计

在通信数据发送与接收之间也存在不同的方式，即同步和异步两种方式。这里的同步和异步与 I/O 层次的同异步概念不同。主要涉及 socket APIs recv() 和 send() 的不同组合方式。

同步发送与接收

从应用程序设计的角度讲，报文发送和接收是同步进行的，既报文发送后，发送方等待接收方返回消息报文。同步方式一般需要考虑超时问题，即报文发出去后发送方不能无限等待，需要设定超时时间，超过该时间后发送方不再处于等待状态中，而直接被通知超时返回。同步发送与接收经常与短连接通信方式结合使用，称为同步短连接通信方式，其 socket 事件流程可如上面的图 1 2 所示。

异步发送与接收

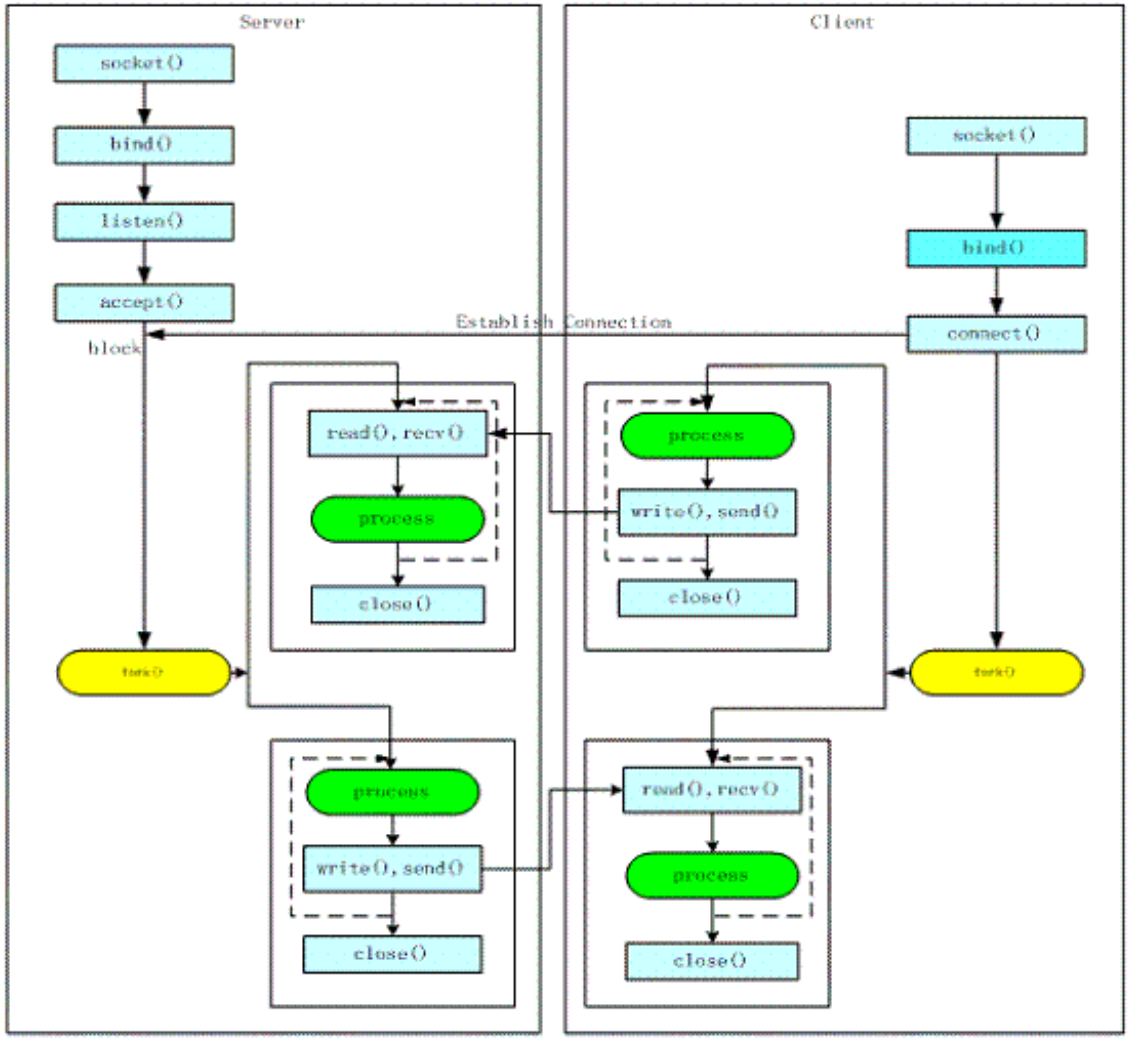
从应用程序设计的角度讲，发送方只管发送数据，不需要等待接收任何返回数据，而接收方只管接收数据，这就是应用层的异步发送与接收方式。要实现异步方式，通常情况下报文发送和接收是用两个不同的进程来分别处理的，即发送与接收是分开的，相互独立的，互不影响。异步

发送与接收经常与长连接通信方式结合使用，称为异步长连接通信方式。从应用逻辑角度讲，这种方式又可分双工和单工两种情况。

异步双工

异步双工是指应用通信的接收和发送在同一个程序中，而有两个不同的子进程分别负责发送和接收，异步双工模式是比较复杂的一种通信方式，有时候经常会出现在不同机构之间的两套系统之间的通信。比如银行与银行之间的信息交流。它也可以适用在现代 P2P 程序中。如图 1 4 所示，Server 和 Client 端分别 fork 出两个子进程，形成两对子进程之间的连接，两个连接都是单向的，一个连接是用于发送，另一个连接用于接收，这样方式的连接就被称为异步双工方式连接。

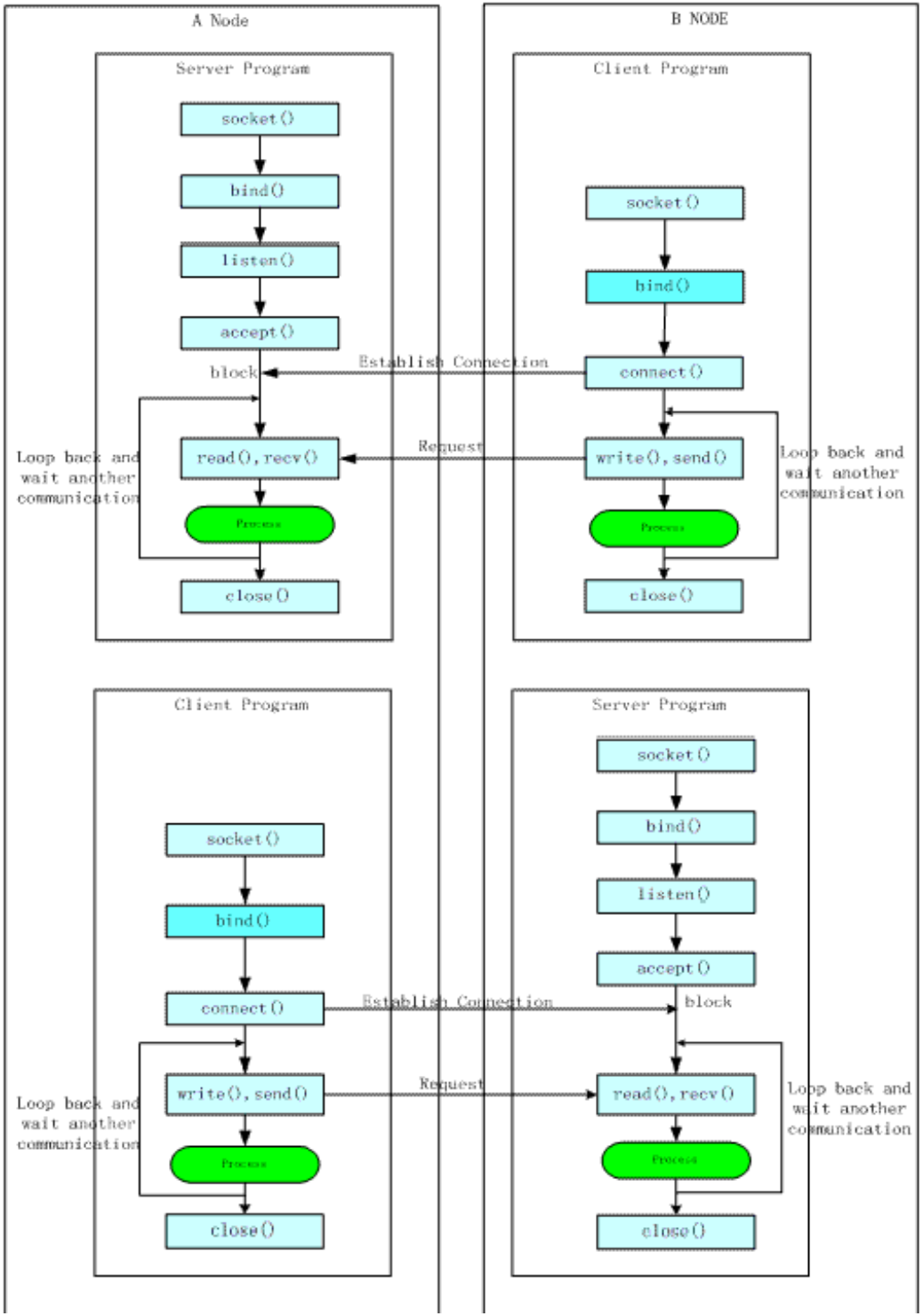
图 1 4 长连接异步双工模式



异步单工

应用通信的接收和发送是用两个不同的程序来完成，这种异步是利用两对不同程序依靠应用逻辑来实现的。图 1 5 显示了长连接方式下的异步单工模式，在通信的 A 和 B 端，分别有两套 Server 和 Client 程序，B 端的 Client 连接 A 端的 Server，A 端的 Server 只负责接收 B 端 Client 发送的报文。A 端的 Client 连接 B 端的 Server，A 端 Client 只负责向 B 端 Server 发送报文。

图 1 5 长连接异步单工模式



回页首

典型通信连接模式

综上所述，在实际 TCP/IP 应用程序设计中，就连接模式而言，我们需要考虑 Client/Server 建立方式、Client/Server 连接方式、Client/Server 发送与接收方式这三个不同级别的设计方式。实际 TCP/IP 应用程序连接模式可以是以上三类不同级别 Client/Server 方式的组合。比如一般 TCP/IP 相关书籍上提供的 TCP/IP 范例程序大都是同步短连接的 Client/Server 程序。有的组合是基本没有实用价值的，比较常用的有价值的组合是以下几种：

- 同步短连接 Server/Client
- 同步长连接 Server/Client
- 异步短连接 Server/Client
- 异步长连接双工 Server/Client
- 异步长连接单工 Server/Client

其中异步长连接双工是较为复杂的一种通信方式，有时候经常会出现在不同银行或不同城市之间的两套系统之间的通信，比如国家金卡工程。由于这几种通信方式比较固定，所以可以预先编制这几种通信方式的模板程序。

回页首

总结

本文探讨了 TCP/IP 应用程序中连接模式的设计。在以后的文章中还将继续讨论 TCP/IP 应用程序设计中的其他方面的设计话题，包括地址族模式设计、I/O 模式设计、以及通信数据格式设计等。

参考资料

- AIX V5.3 中 IPv4 和 IPv6 的网络接口操作：通过本文，您将了解更多关于套接字 I/O 控制 (ioctl) 命令的内容，以及如何使用它们完成各种网络相关的操作 . 操作系统为套接字、路由表、ARP 表、全局网络参数和接口提供了相应的控制操作方式。
- Unix Network Programming Volume 1: 帮助您全面地了解 Unix 网络编程的知识。
- AIX and UNIX 专区：developerWorks 的“AIX and UNIX 专区”提供了大量与 AIX 系统管理的所

有方面相关的信息，您可以利用它们来扩展自己的 UNIX 技能。

- AIX and UNIX 新手入门：访问“AIX and UNIX 新手入门”页面可了解更多关于 AIX 和 UNIX 的内容。
- AIX and UNIX 专题汇总：AIX and UNIX 专区已经为您推出了很多的技术专题，为您总结了很多热门的知识点。我们在后面还会继续推出很多相关的热门专题给您，为了方便您的访问，我们在这里为你把本专区的所有专题进行汇总，让您更方便的找到你需要的内容。
- 获取 本专区的 RSS Feed。（了解关于 RSS 的更多信息。）

条评论

请 登录 或 注册 后发表评论。

添加评论:

注意：评论中不支持 HTML 语法

有新评论时提醒我

剩余 1000 字符

发布

共有评论 **(3)**

学习了 非常感谢

由 Jakub Fong 于 2015年07月08日发布

报告滥用

网上到处转载
原来原作在此
学习了 前辈

由 dong_dong 于 2012年08月03日发布 报告滥用

不错，好好学习一下

由 weichangcheng 于 2012年07月26日发布 报告滥用

回页首