高级TCP/IP编程

Effective TCP/IP Programming

Jon C.Snader 著

刘江林 译

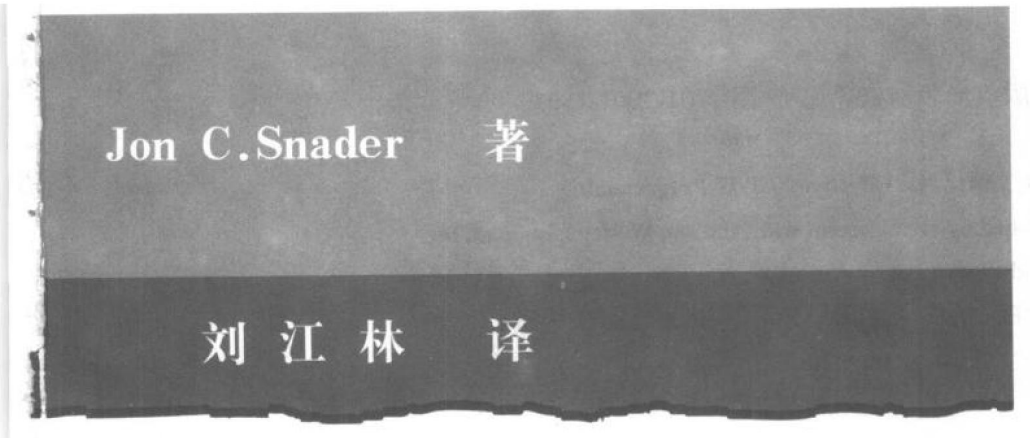
44 Tips to Improve Your Network Programs

Addison-Wesley

中国电力出版社
www.infopower.com.cn

高级TCP/IP编程

Effective TCP/IP Programming



44 Tips to Improve Your Network Programs

中国电力出版社

内 容 提 要

本书涉及了网络编程的所有细节，通过对 TCP/IP 编程精细部分的分析，帮助读者理解网络协议内部是如何与应用程序交互的。全书分为四章，提供了 44 个 TCP/IP 编程技巧，生动详实的探索了网络编程的各个方面。

本书适合中、高级网络程序员阅读，也可供专业计算机人士参考。

图书在版编目（CIP）数据

高级 TCP/IP 编程/（美）斯纳德著；刘江林译. -北京：中国电力出版社，2001.6

ISBN 7-5083-0661-9

Ⅰ.高… Ⅱ.①斯…②刘… Ⅲ.计算机网络-通信协议-程序设计 Ⅳ.TN915.04

中国版本图书馆 CIP 数据核字（2001）第 034249 号

北京版权局著作权登记号 **图字** 01-2001-1871

中国电力出版社出版、发行

（北京三里河路 6 号 100044 http://www.infopower.com.cn）

三河市实验小学印刷厂印刷

各地新华书店经售

*

2001 年 8 月第一版 2001 年 8 月北京第一次印刷

787 毫米×1092 毫米 16 开本 19.5 印张 441 千字

定价 **35.00** 元

前　言

简介

随着 Internet 爆炸性的增长，无线通信以及网络技术的发展，编写网络应用程序的程序员和工程师也在相应增加。TCP/IP 编程看起来非常简单。应用程序接口（Application Programming Interface，API）十分易懂，即使是刚刚入门的初学者也可以通过使用客户端或服务器的程序模板来编写应用程序。

然而，通常初学者在经历了最初的高效率之后，在细节面前开始停滞不前，并且发现他们的应用程序正遭受着性能或健壮性的考验。网络编程是一个充满着黑暗角落的领域，有些细节也可能会被错误地理解。本书通过对 TCP/IP 编程最精细部分的分析，让光明照亮那些角落并帮助你改正错误。

在读完本书之后，你对网络编程的许多难点会有一个透彻的理解。本书涉及了网络编程的所有细节。通过对这些小细节的理解，读者将获得相应知识，即网络协议的内部工作机制是如何跟应用程序交互的。拥有了这些知识，那些以前看起来令人困惑的程序行为就会变得很容易理解，问题的解决办法也会变得很清晰。

本书的组织方式有点与众不同，我们在一系列的技巧中每次只探讨一个问题。在学习一个特定的难点的过程中，我们会比较深入地去探索 TCP/IP 编程的某一个方面。这样，读完本书之后，读者不仅可以确认和处理一般的问题，而且对 TCP/IP 协议如何与应用程序一起工作和交互会有一个相对全面的认识。

本书的组织方式也许会导致某些地方的脱节。为了方便读者阅读，第一章包含了一个路径图，它解释每章的内容与基本结构。目录中列出了所有的技巧，这也会加强你对文字组织方式的理解。因为每个技巧的标题是以祈使句的形式表达的，所以你也可以考虑使用目录作为一系列的网络编程规则。

另一方面，技巧的组织方式使本书更适合作为一本手册。当在日常工作中遇到问题，可以很容易地去重读相应的技巧，重新认识特定的问题。你将会发现许多主题在多个技巧中涉及到，这种重复可以帮助你巩固概念并使它们看起来更简单自然。

读者对象

本书主要是为有一定基础的初学者和中级网络程序员编写的，但是对有经验的读者

也有一定的参考价值。虽然我们假定读者对网络和基本的套接字 API 有一定的熟悉程度，但在第一章里还是包含了对基本的套接字调用的回顾，并使用它们建立了一个原始的客户端和服务器。技巧 4（开发和使用程序框架）更详细地讲解了不同的客户端和服务器模型，所以即使读者只有很少的网络编程知识，也能够理解本书并从中受益。

几乎所有的例子都是用 C 语言编写的，所以读者必须对基本的 C 语言有一个很好的理解，这样你就会从本书中的程序中获益匪浅。在技巧 31（记住世界并不全是 C 语言的）里，我们展示了用 Perl 语言编写的例子，但是我们假定你没有 Perl 语言的知识。同样，本书中有一些小的 shell 程序的例子，但是理解它们也无需有 shell 编程经验。

例子和文字试图做到与平台无关。除了少数例子外，其他的都可以在 UNIX 或 Win32 系统下编译并运行。即使那些不在 UNIX 或 Win32 系统下工作的程序员，也会很轻松地把那些例子移植到他们工作的平台上。

排版约定

在我们学习的过程中，将建立和运行许多小的程序，这些程序是为我们正在探讨的问题而举出的例子。在显示交互的输入和输出时，我们使用下面的约定：

- 我们输入的英文设置为加黑 Courier 字体。
- 系统输出的英文设置为普通 Courier 字体。
- 不作为实际输入和输出部分的注释设置为斜体。

下面是从技巧 9 中摘取的一个例子：

```
bsd: $ tcprw localhost 9000
hello
received message 1                    this is printed after a 5-second delay
                                      the server is killed here

hello again
tcprw: readline failed: Connection reset by peer (54)
bsd: $
```

注意，我们在 shell 提示符中包含了系统名称。在前面的例子里，我们看到 tcprw 运行在名称为 bsd 的主机上。

在我们介绍一个新 API 函数时，我们把自己的或标准系统调用封装到一个方框中。标准的系统调用封装到实线方框里，如下所示：

```
#include<sys/socket.h>          /* UNIX */
#include<winsock2.h>            /* Windows */

int connect(SOCKET s,const struct sockaddr *peer, int peer_len );

                        Returns:0 on success, -1(UNIX)or nonzero(Windows)on failure
```

我们自己开发的 API 函数封装到虚线方框里，如下所示：

```
#include "etcp.h"
SOCKET tcp_server( char *host, char *port );
                                    Returns:a listening socket(terminates on error)
```

附加的注释和放于脚注里的资料设置为小字体并和本段一样缩进。通常这些材料可以在第一遍阅读时忽略掉。

最后，我们用尖括号把 URL 括起来，如下所示：

<http://www.freebsd.org>

可获得的源代码和勘误表

本书中所有例子的源代码都可以从<http://www.netcom.com/~jsnader>上获得电子版本。因为本书中的例子是直接从程序中抽取出来的，你可以得到它们并在它们上面做自己的试验。框架和库代码也可以下载来供自己使用。

在同一 Web 站点还可以获得勘误表。

致谢

通常，作者都要感谢他们的家庭成员在写书过程中给予的帮助，现在我知道他们为什么要这样做了。如果没有我的妻子 Maria 的帮助，本书的文字也许不会令人感到满意。如果没有她承担的比我的“50%”大得多的校对，也许我就不会有足够的时间来完成本书。对此她所干的额外家务和忍受的孤独夜晚，任何感激的话都不为过。

作者的另一个宝贵的资源是那些苦苦读完早期草稿的审阅者。本书的技术审阅者找出了很多错误。既有技术错误也有排版错误，他们改正我错误的理解，建议我采用新的方法，告诉我以前不知道的东西，有时甚至给我鼓励。在此我感谢 Chris Cheeland、Bob Gilligan（FreeGate 公司）、Peter HaverLock（北电网络）、S. Lee Henry（Web Publishing，公司）、Mukesh Kacker（Sun 公司）、Uri Raz 和 Rich Stevens，感谢他们的辛勤工作和建议。本书因为他们的帮助而做得更好。

最后，我要感谢我的编辑 Karen Gettman、项目编辑 Mary Hart、产品协调员 Tyrrell Albaugh 和拷贝编辑 Cat Ohala。和他们一起工作充满了乐趣，他们对我这个第一次写书的作者的帮助很大。

欢迎读者把评论、建议和校正反馈给我。

Jon C. Snader
jsnader@ix.netcom.com
http://www.netcom.com/~jsnader

Tampa, Florida
1999 年 12 月

目　　录

第一章

简　　介

本书的目的是帮助有一定基础的初学者和中级网络程序员成为一个熟练的网络编程人员，并且最终成为大师。向大师级网络程序员的转变在很大程度上靠的是经验和特定知识的积累，当然这些东西有时是很模糊的概念。除了时间和实践之外没有别的东西能够提供经验，而本书可以在知识的积累方面帮助你。

当然，网络编程是一个很大的领域，采用网络技术在两台或更多的机器之间进行通信时，存在很多的障碍。这些障碍从简单的如串行链接到复杂的如 IBM 的系统网络体系结构（System Network Architecture），应有尽有。今天，我们清楚地知道 TCP/IP 协议是构造网络的首选技术。这个选择很大程度上是因为 Internet 以及它最流行的应用程序：万维网（World Wide Web）。

> 虽然 Web 使用了应用程序（Web 浏览器和服务器）和协议（如 HTTP）。但实际上它并不是一个应用程序也不是一个协议。我们的意思是，Web 是 Internet 上的最流行的可视的网络技术应用程序。

即使在 Web 诞生之前，TCP/IP 也是一种很流行的构架网络的方法。因为它是一个开放的标准并且可以在不同供应商的机器之间进行互连，所以人们越来越多地使用它来建设网络和网络应用程序。到 1990 年底，TCP/IP 已经成为网络的主导技术，而且这种趋势似乎将在一段时间内保持下去。因此，本书把精力集中在 TCP/IP 以及运行 TCP/IP 的网络上。

如果希望掌握网络编程，就必须首先汲取一些背景知识，这些知识对更全面地理解和评价本书的程序是很有用的。首先介绍初学者通常会遇到的几个问题，这些问题大都是由于一些误解造成的，其中包括对 TCP/IP 某些方面和用于在机器之间进行通信的 API 的不完全理解。所有这些问题都是客观存在的，它们永远是困惑的源泉，而且在网络新闻组里也是经常涉及到的主题。

一　些　约　定

本书中的文本材料和程序，除了一些显而易见的之外，都试图做到可以在 UNIX（32 位或 64 位）和 Microsoft Win32 API 之间移植。我们对 16 位的 Windows 应用程序不予考虑。也就是说，几乎所有的材料和大多数的程序在其他的环境里也是适用的。

这个易移植的愿望在代码示例中导致了一些不便。UNIX 程序员也许对套接字描述符的概念斜眼相看，因为套接字描述符是用 SOCKET 而不是用 int 类型定义的；而 Windows 程序员也会注意到我们严格的提交控制台应用程序。这些约定将在技巧 4 中讲述。

同样地，我们尽量避免在套接字上使用 read 和 write 函数，这是因为 Win32 API 不支持在套接字上调用这些系统函数。我们经常谈到读或写一个套接字，但是我们只是一般地说说而已。我们用 recv、recvfrom 或 recvmsg 来表示“读”，而用 send、sendto 或 sendmsg 来表示“写”。在特定地表示读系统调用时，我们把它设置为 Courier 字体的 read，该约定对写也一样适用。

是否在本书中包含 IPv6 的材料是最难做出的选择之一，IPv6 在不久的将来会取代 Internet 协议（IP）的现有版本（IPv4）。最后，我们决定不涉及 IPv6。做出这个决定有很多原因，包括：

- 几乎所有本书中的东西都是正确的，不管它是使用 IPv4 还是 IPv6。
- 确实存在的差异在 API 的地址部分本地化了。
- 本书在很大程度上提供的是熟练网络程序员的共同的经验和知识，而且我们确实仍然没有 IPv6 的经验，这是因为 IPv6 仅仅在近期才广泛地得以实现。

因此，如果我们谈到没有限定的 IP 时，指的就是 IPv4。在确实提到 IPv6 的地方，我们会十分小心地明白地指出它是 IPv6。

最后，本书以 8 位数据单元为一个字节，而在网络社区里通常称这些数据为 octets，这是历史原因造成的。过去一个字节的大小是依赖于平台而定的，即使是它大小标准也没有统一一致。为了避免混淆，早期的网络书籍杜撰术语 octet 来表示字节的大小。今天，人们普遍认同一个字节就是 8 位长[Kernighan and Pike 1999]，而使用 octet 似乎显得有些书生气了。

本 书 结 构

在本章的其余的部分中，将介绍基本的套接字 API 和在编写 TCP/IP 应用程序时使用的客户端-服务器体系结构。这是掌握网络编程所必须建立的基础。

第二章讨论了一些有关 TCP/IP 和网络的基本事实和误解。例如，在本章将讲述基于连接协议和无连接协议之间的差异。我们将研究 IP 寻址和子网的一些经常令人困惑的主题，classless interdomain routing（无级域间路由选择，CIDR）和 network address translation（网络地址转换法，NAT）。我们可以了解到 TCP 实际上不保证数据的递交，认识到我们必须为对等方（peer）和用户的错误操作做准备，以及我们的应用程序在广域网（WAN）内运行的结果可能和在局域网（LAN）内不一样。

我们知道 TCP 是一个流协议，也知道这对作为程序员的我们来说意味着什么。同时，我们将了解到 TCP 不自动检测连接的丢失，了解到为什么不检测是好事情，也将了解到我们可以在它上面做些什么。

我们将了解为什么套接字 API 应该总是更喜欢建立在 Transport Layer Interface/X/Open

Transport Interface（TLI/XTI）之上，了解我们为什么不应把开放系统互联模型（OSI）看得很重要，也将了解 TCP 是一个具有显著效率、优秀性能的协议，了解经常使用 UDP 来重现它的功能是不明智的。

在第二章中，将为几个 TCP/IP 应用程序模型开发框架代码，并使用它来构造一个经常使用的函数的类库。这些框架和类库都是很重要的，因为它们使我们在编写应用程序的时候不担心例行的琐事如地址转换、连接管理等等。因为可以使用这些框架，所以我们就不会被捷径所诱惑，如固定地址和端口号，或忽略错误返回值。

我们在本书中重复地使用这些框架和类库来建立测试例子、示例代码，甚至使用独立工作的应用程序。我们经常通过在自己的框架中增加几行代码来建立一个特殊用途的应用程序或测试例子。

在第三章中，我们比较深入地讨论了几个看起来很琐碎的主题。例如，我们首先讨论了 TCP 的一些操作以及该操作做了一些什么事情。开始看起来这似乎很简单：我们写入 n 个字节，然后 TCP 把它们发送到对等方。然而，正如我们将会看到的，实际情况并非如此。TCP 有一个复杂的规则集决定它是否可以把写入的数据立即发送出去。如果可以的话，代价又是多少呢？如果要编写既健壮又高效的程序，那么理解这些规则以及它们是如何跟我们的应用程序交互的就很必要了。

读数据和连接终止需要考虑同样的事情。我们将分析这些操作并学会如何执行一个有序的终止过程，如何保证数据不会丢失。我们也将分析 connect 操作并学会如何指定超时，以及如何在 UDP 应用程序中使用它。

我们将会学习 UNIX 超级服务器端口监视程序（inetd）的使用，以及了解到它是如何极大地减少编写具有网络功能的应用程序必须做的工作。同时，我们也会了解到我们是如何使用 tcpmux 来减少分配已知端口给服务器时的风险。我们将展示 tcpmux 是如何工作的以及如何在没有该功能的系统上建造自己版本的 tcpmux。

本书还将深入地讨论下列不易理解的主题：TIME-WAIT 状态、Nagle 算法、选择缓冲区大小以及 SO_REUSEADDR 套接字选项的正确用法。我们将了解如何使我们的程序成为应用程序事件驱动的程序以及如何才能为每一个事件提供独立的定时器。我们也会分析一些即使是高级的网络程序员也会犯的错误，并且将会学习一些可以用来提高应用程序性能的技术。

最后，我们介绍了网络和脚本语言。通过使用一些 Perl 脚本，说明了使用 Perl 是可以很容易地编写有用的网络实用程序和测试驱动程序的。

第四章介绍了两个领域。首先分析了几个工具，这些工具是每一个网络程序员必须知道的东西。开始介绍的是经典的 ping 实用程序，并说明了它是如何可以用在一些基本的疑难解答上面的。接着泛泛地分析了网络监听工具并特别分析了 tcpdump。在整个第四章中，我们将会看到运用 tcpdump 来诊断应用程序问题和困惑的几个例子。也会学习 traceroute 以及应用它来研究 Internet 的一个很小部分的形状。

ttcp 实用程序（该程序是 ping 程序的作者 Mike Muuss 的合作者）是一个研究网络性能和某些 TCP 参数对性能影响的很有用的工具。我们使用它来演示一些诊断技术。另一个公

共域工具 lsof 在匹配网络并连接到打开这些连接的进程时是很有用的。在很多时候，lsof 提供了一些很有用的信息，这些信息在没有 lsof 时不花大力气是不可获得的。

我们将详细地分析 netstat 实用程序，以及它提供的很多不同类型的信息。我们也会分析系统调用跟踪程序如 ktrace 和 truss。

我们在第四章中编写一个分析工具来结束该章第一部分的讨论，这个工具可以拦截并显示 Internet Control Message Protocol（网际控制报文协议，ICMP）数据报。它不仅为我们的工具箱里增加了一个有用的工具，同时也是一个使用原始套接字（raw sockets）的例子。

在第四章的第二部分中我们介绍了一些可用来加深我们的知识、加深对 TCP/IP 和网络的理解的资源。我们将会了解 Rich Stevens 编写的著名书籍、我们可以研究和学习的网络代码、可以从 Internet Engineering Task Force（因特网特别工作组，IETF）获得的 request for comments（请求评议，因特网提议的标准，RFC）集合以及 Usenet 新闻组。

客户端-服务器体系结构

尽管我们经常说起客户端和服务器，但在一般的情况下，一个特定的程序到底是客户端还是服务器不总是很清楚。程序经常更像对等的双方，它们之间互相交换信息，无法看出给客户端提供的信息。通过 TCP/IP 来看，区别就明显了。服务器监听 TCP 连接或客户端主动提供的 UDP 数据报。从客户端的角度来看，我们可以说客户端是首先“说话”的一方。

在本书中我们考虑三种典型的客户端-服务器的情形，如图 1.1 所示。在第一种情形中，

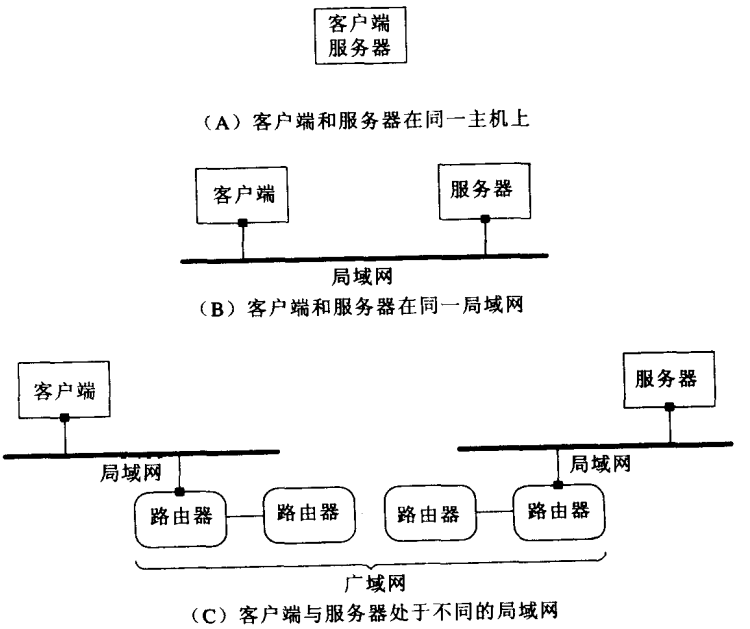


图 1.1 典型的客户端-服务器结构的几种情形

客户端和服务器运行在同一台机器上，如图 1.1（A）所示。因为没有涉及到物理网络，所以这是最简单的一种情形。输出的数据照常沿着 TCP/IP 栈下行，但是它不放在网络设备输出队列上，而是在内部返回，沿着 TCP/IP 栈上行作为输入。

在开发时，这种设置有很多优点，即使客户端和服务器最终是运行在不同的机器上。首先，可以很容易粗略地判断客户端和服务器应用程序的性能，因为在这种情况下没有网络延时；其次，这种方法提供了一种理想化的实验环境，包不会丢失、延迟或不按照顺序来递交。

> 正如我们将会在技巧 7 里了解的那样，至少是在大多数的时间里，我们必须强调，即使在这种环境下也可能导致 UDP 数据报的丢失。

最后，如果我们在同一台机器上调试的话，那么开发工作就会经常变得更容易而且更方便。

当然，客户端和服务器即使是在成为产品的环境下也可以运行在同一台机器上。请参阅技巧 26，了解基于这种情况的一个例子。

在第二种客户端-服务器设置中，如图 1.1（B）所示，客户端和服务器运行在同一个局域网内的不同机器上。这个环境涉及到真正的网络，但是这个环境仍然还是近乎理想化的。数据包几乎不丢失，实际上也不会乱序到达。这是一个通常的产品环境，在很多情况下应用程序就是专为这种环境设计的，而不会运行在其他的环境下。

基于这种情况的一个常见的例子就是打印服务器。一个小的局域网中可能为几台主机只配置一个打印机。其中一个主机（或者打印机内置的 TCP/IP 栈）充当服务器，接收来自其他主机的客户端的打印请求，并把这些数据放到缓冲区中等候打印机打印。

第三种类型客户端-服务器的情形涉及到被广域网如图 1.1（C）分割的客户端和服务器。广域网可以是 Internet 或者是公司的 intranet，但是最关键的是两个应用程序不在同一个局域网内，而且从一个应用程序发出到另一个应用程序的 IP 数据报必须通过一个或几个路由器。

这种环境比前两种复杂得多。随着广域网内流量的增加，路由器队列经常临时存储数据包直到它们可以转发出去才开始填充。当路由器的队列空间不够时，它们就开始丢弃数据包，这就会导致重传，进而导致重复和乱序传递数据包。这些问题都不是理论上的，而且都比较常见，如同我们将会在技巧 38 里看到的那样。

我们将会在技巧 12 里详细地讨论局域网环境和广域网环境之间的差异，但是目前我们仅仅只能说它们的表现几乎完全不同。

基本的套接字 API 回顾

在本节中，我们将回顾基本的套接字 API 并使用它来创建初步的客户端和服务器应用程序。尽管这些应用程序上是“没有血肉的”，但它们确实说明了 TCP 客户端和服务器的本质特性。

让我们以简单的客户端所必须的 API 调用来开始。图 1.2 是每个客户端必须使用的基本套接字调用的示意图。如图 1.2 所示，对等方（peer）的地址在 sockaddr_in 结构里被指定。并传递到 connect。

```
#include <sys/socket.h>        /* UNIX */
#include <winsock2.h>          /* Windows */

SOCKET socket( int domain, int type, int protocol);
```

Returns:Socket descriptor on success, -1(UNIX) or INVALID_SOCKET (Windows) on failure

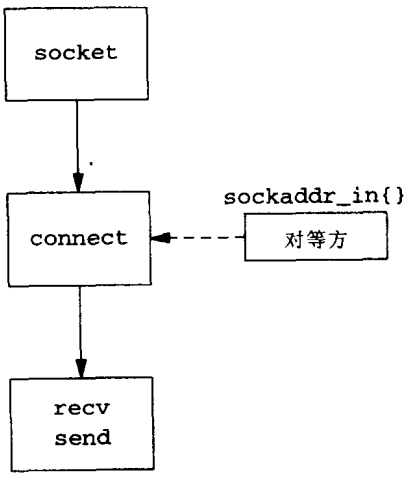


图 1.2　客户端的基本套接字调用

通常我们必须做的第一件事情是为连接获得一个套接字，通过 socket 系统调用来完成。

套接字 API 是与协议无关的，并且它可以支持几个不同的通信域（communication domain）。domain 参数是一个常数，表示所希望的通信域。

两个最普通的域是 AF_INET（或 Internet）域和 AF_LOCAL（或 AF_UNIX）域。在本书中我们只考虑 AF_INET 域。AF_LOCAL 域用于同一机器上的进程间通信（interprocess communication，IPC）。

> 关于 domain 常数应当为 AF_*还是 PF_*的争论一直存在。PF_*的支持者指出 PF_*曾用于 4.1c/2.8/2.9BSD 的至今已经不存在的 socket 调用版本中，并指出 PF 代表的是 protocol family 这一事实。AF_*的支持者指出核心套接字代码和 AF_*常数的 domain 参数相匹配。因为这两套常数是同样地定义的——实际上，其中一个是根据另一个定义的——所以我们无论使用哪一个都没有应用上的差别。

type 参数指示将要创建的套接字的类型。最常用的值以及我们在本书中使用的值列出如下：

- SOCK_STREAM——这些套接字提供了一个可靠的、全双工的面向连接的字节流。在 TCP/IP 中，它指的是 TCP。
- SOCK_DGRAM——这些套接字提供了一个不可靠的、效率高的数据报服务。在 TCP/IP 中，它指的是 UDP。
- SOCK_RAW——这些套接字允许访问 IP 层中的一些数据报。它们用于特殊的用途，如监听 ICMP 消息。

protocol 域指示对套接字应当使用哪一个协议。对于 TCP/IP，这通常用套接字类型来显式地指定，并且该参数设置为 0。在一些情况下，如 raw 套接字，存在好几个可能的协议，你必须指定其中一个为将要使用的协议。我们将会在技巧 40 里看到这样的一个例子。

对于最简单的 TCP 客户端，另外一个 socket API 要求我们建立一个同我们对等方的会话是 connect，它用于建立连接：

```
#include <sys/socket.h>       /* UNIX */
#include <winsock2.h>         /* Windows */
int connect( SOCKET s, const struct sockaddr *peer, int peer_len );

                    Returns:0 on success, -1 (UNIX) or nonzero (Windows) on failure
```

s 参数是 socket 调用返回的套接字描述符。peer 参数指向一个地址结构，它保存着对等方的地址和其他信息。对于 AF_INET 域来说，这是一个 sockaddr_in 结构。让我们花费几分钟来看一下那个简单的例子。peer_len 参数是 peer 指向的结构的大小。

一旦建立了一个连接，我们就可以传输数据了。在 UNIX 平台下，我们可以简单地使用套接字描述符来调用 read 和 write 函数，这与我们使用文件描述符是一样的。正如我们前面提到的一样，不幸的是，Windows 并没有在套接字环境下重载这些系统调用，我们必须使用 recv 和 send 来代替它们。除了有一个额外的参数，这些调用是与 read 和 write 一样的：

```
#include <sys/socket.h>        /* UNIX */
#include <winsock2.h>          /* Windows */

int recv( SOCKET s, void *buf, size_t len, int flags );

int send( SOCKET s, const void *buf, size_t len, int flags );

                    Returns:number of bytes transferred on success, -1 on failure
```

s、buf 和 len 参数是与 read 和 write 中的参数一样的。flags 参数可以采用的值通常是跟系统有关的，但是 UNIX 和 Windows 都支持下列值：

- MSG_OOB——如果设置为该值，这个标志指示就要发送或读取紧急的数据。
- MSG_PEEK——该标志用于读取进来的数据，但不把数据从接收缓冲区里删除。调用该函数后，数据仍然可以被以后的读操作获得。

- MSG_DONTROUTE——该标志指示系统内核忽略通常的路由功能。它通常仅用于路由程序中或者用于诊断。

在处理 TCP 时，这些调用通常是必需的。然而，对于 UDP 的使用，recfrom 和 sendto 调用是很有用的。这些调用与 reve 和 send 是亲兄弟，但是它们允许我们在发送一个 UDP 数据报时指定目的地址以及在读取一个 UDP 数据报时取出源地址：

```
#include <sys/socket.h>          /* UNIX */
#include <winsock2.h>            /* Windows */

int recv from( SOCKET s, void *buf, size_t len, int flags );
          struct sockaddr *from, int *fromlen );

int send to( SOCKET s, const void *buf, size_t len, int flags );
          const struct sockaddr *to, int tolen );
```

Returns:number of bytes transferred on success, -1 on failure

前面的 4 个参数——s、buf、len 和 flags——与在 recv 和 send 调用中是一样的。recfrom 调用中的 from 参数指向一个套接字地址结构，系统内核在这个结构中存储进来的数据报的源地址信息。该地址的长度存储在 fromlen 指向的整数中。应当注意的是 fromlen 是一个指向整数的指针。

同理，在 Sendto 调用中 to 参数指向一个套接字地址结构，它包含数据报的目的地，tolen 参数是此地址结构的长度，注意 tolen 仅仅是一个整数，不是一个指针。

现在我们看一个简单的 TCP 客户端（图 1.3）。

simplec.c

```
1  #include <sys/types.h>
2  #include <sys/socket.h>
3  #include <netinet/in.h>
4  #include <arpa/inet.h>
5  #include <stdio.h>

6  int main( void )
7  {
8    struct sockaddr_in peer;
9    int s;
10   int rc;
11   char buf[ 1 ];

12   peer.sin_family = AF_INET;
```

```
13  peer.sin_port = htons( 7500 );
14  peer.sin_addr.s_addr = inet_addr( "127.0.0.1" );

15  s = socket( AF_INET, SOCK_STREAM, 0 );
16  if ( s < 0 )
17  {
18      perror( "socket call failed" );
19      exit( 1 );
20  }

21  rc = connect( s, ( struct sockaddr * )&peer, sizeof( peer ) );
22  if ( rc )
23  {
24      perror( "connect call failed" );
25      exit( 1 );
26  }
27  rc = send( s, "1", 1, 0 );
28  if ( rc <= 0 )
29  {
30      perror( "send call failed" );
31      exit( 1 );
32  }
33  rc = recv( s, buf, 1, 0 );
34  if ( rc <= 0 )
35      perror( "recv call failed" );
36  else
37      printf( "%c\n", buf[ 0 ] );
38  exit( 0 );
39 }
```

simplec.c

图 1.3 一个简单的 TCP 客户端

我们把图 1.3 的代码编写成一个 UNIX 程序，以后再处理复杂的可移植代码和 Windows WSAStartup 逻辑性。正如我们将在技巧 4 中看到的一样，我们可以在一个头文件中隐含大部分的繁杂代码，但是在这之前我们必须首先建立一些方法。而在这一切之前，我们将只使用相对简单的 UNIX 模式。