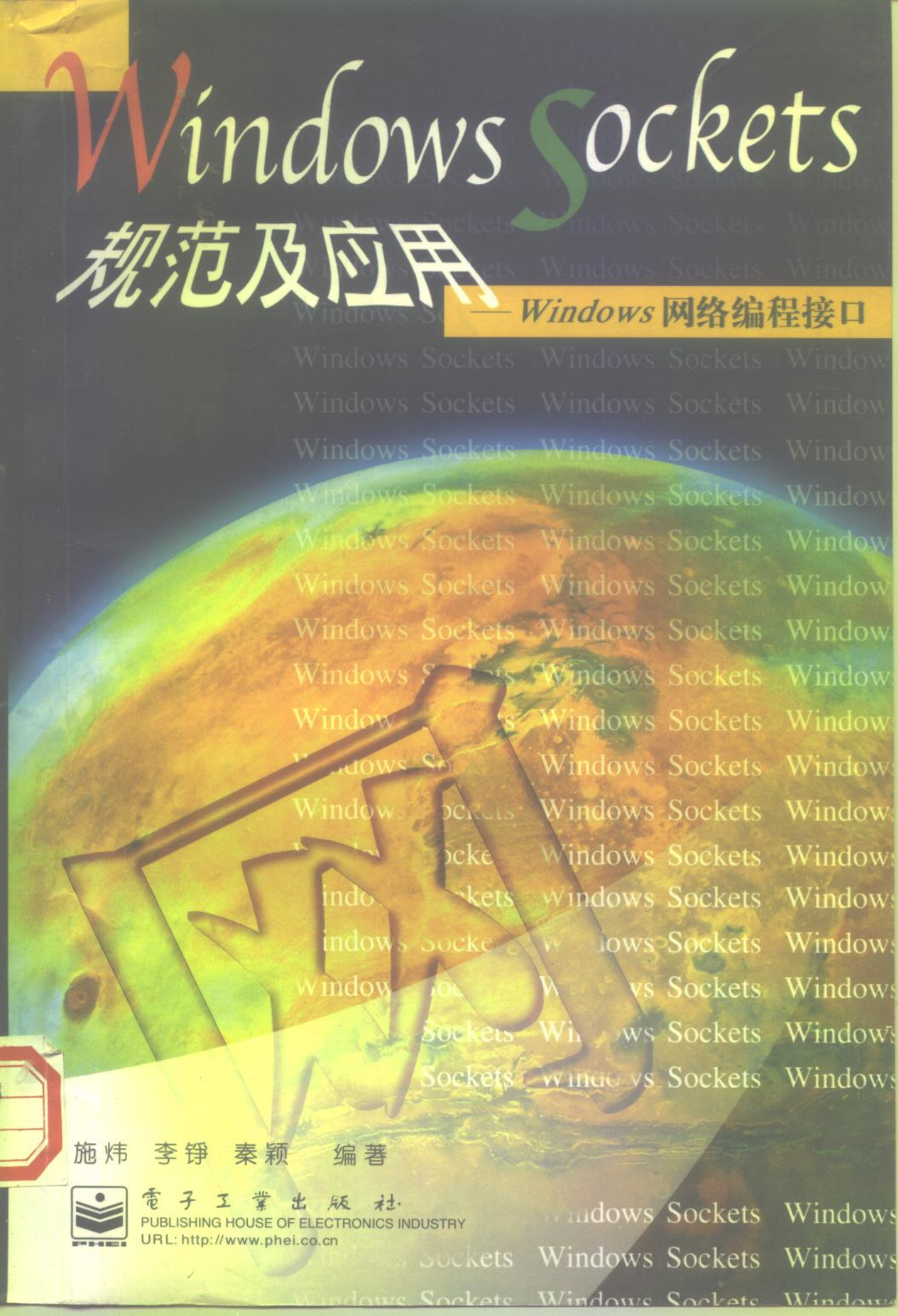
Windows Sockets 规范及应用

—Windows网络编程接口



施炜 李铮 秦颖 编著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
URL: http://www.phei.co.cn

Windows Sockets 规范及应用

——Windows 网络编程接口

施炜　李铮　秦颖　编著

電子工業出版社
Publishing House of Electronics Industry

内 容 提 要

本书适应了 Windows、Internet 及计算机网络普及的潮流，介绍了一套在 Windows 下网络编程的规范——Windows Sockets。这套规范是 Windows 下得到广泛应用的、开放的、支持多种协议的网络编程接口。从 1991 年的 1.0 版到 1995 年的 2.0.8 版，经过不断完善并在 Intel，Microsoft，Sun，SGIInformix，Novell 等公司的全力支持下，已成为 Windows 网络编程的事实上的标准。为使读者能够充分理解和应用这套规范，本书不但对 Windows Sockets 1.1 及2.0规范作了较为详尽的介绍，还结合了作者的实际工作，给出了具有实际应用价值的程序实例。书中的内容包括：Windows Sockets 规范 1.1 版及 2.0.8 版介绍；Windows Sockets 网络编程指导和具体应用实例；Windows Sockets 规范 1.1 版及 2.0.8 版库函数参考等。

本书体系完整，文字流畅，可供从事网络应用开发的工程技术人员和大专院校师生参考。

书　　名：Windows Sockets 规范及应用——Windows 网络编程接口
编 著 者：施　炜　李　铮　秦　颖
责任编辑：焦桐顺
印 刷 者：北京李史山印刷厂
出版发行：电子工业出版社出版、发行　URL：http://www.phei.co.cn
　　北京市海淀区万寿路 173 信箱　邮编 100036　发行部电话：68214070
经　　销：各地新华书店经销
开　　本：787 x 1092 1/16　印张：13.125　字数：336 千字
版　　次：1997 年 8 月第 1 版　1997 年 8 月第 1 次印刷
书　　号：ISBN 7-5053-4276-2
　　TP·1941
定　　价：16.50 元

前　言

当今世界正处于信息时代，计算机和通信网络是这一时代的所谓“信息基础设施”。网络化是计算机技术90年代的重要发展趋势之一。目前计算机网络的新发展是：异机种网络和异网互联有较大突破。TCP/IP协议在异网互联中体现出了其强大的生命力，以它为基础组建的Internet是目前国际上规模最大的计算机网间网，到1991年底世界上已有26个国家的5千多个网络连入Internet，其中包含了数千个组织的30万台主机，用户数以千万计。

与计算机网络的普及相呼应的是Windows的广泛应用，现在在全世界各地已有超过4千万用户在使用不同版本的Windows。自1995年8月24日Windows 95正式推出以来，在短短的一个星期内销售量已超过100万份，有的零售商店不得不半夜开门，以迎接滚滚而来的抢购者。这说明以用户友好的图形界面为基础的Windows已得到用户的普遍认可，已经并将继续成为个人机平台上的事实上的操作系统标准。所以研究和开发在Windows下的网络编程技术具有普遍的应用价值。

在Windows下的各种网络编程接口中，Windows Sockets脱颖而出，越来越得到大家的重视，这是因为Windows Sockets规范是一套开放的、支持多种协议的Windows下的网络编程接口。从1991年的1.0版到1995年的2.0.8版，经过不断完善并在Intel，Microsoft，Sun，SGI，Informix，Novell等公司的全力支持下，已成为Windows网络编程的事实上的标准。

在作者利用Windows Sockets规范进行应用开发的过程中，发现这方面的资料很少，特别是缺乏一本全面而实用的专著。为了使广大用户能够充分理解和应用这套规范，我们编写了这本书。本书不但对Windows Sockets 1.1及2.0规范作了较为详尽的介绍，还结合了作者的实际工作，给出了具有实际应用价值的程序实例。希望它能对Windows Sockets规范在国内的推广和应用起到抛砖引玉的作用。读者在阅读本书的过程中，如果能对自己的学习工作有所帮助和指导，是作者的最大愿望。由于时间紧迫，作者学识有限，书中错误在所难免，偏颇和不当之处，恳请读者不吝赐教。

本书由施炜、李铮、秦颖合作完成，其中，第一、二、四、六章和5.2节由施炜编写；第七章、5.1节、3.4节由李铮编写；第5.3节、3.1～3.3节由秦颖编写。在本书的编写过程中，得到了上海交通大学的毛向辉先生的大力支持，并提供了一些最新的资料，在此谨表示衷心的谢意。

编者

1996年9月于上海交通大学

目　录

第一章　简介

1.1　什么是 Windows Sockets 规范?

Windows Sockets 规范以 U.C. Berkeley 大学 BSD UNIX 中流行的 Socket 接口为范例定义了一套 Micosoft Windows 下的网络编程接口。它不仅包含了人们所熟悉的 Berkeley Socket 风格的库函数,也包含了一组针对 Windows 的扩展库函数,以使程序员能充分利用 Windows 消息驱动机制进行编程。

Windows Sockets 规范本意在于提供给应用程序开发者一套简单的 API,并让各家网络软件供应商共同遵守。此外,在一个特定版本 Windows 的基础上,Windows Sockets 也定义了一个二进制接口(ABI),以此来保证应用 Windows Sockets API 的应用程序能够在任何网络软件供应商的符合 Windows Sockets 协议的实现上工作。因此这份规范定义了应用程序开发者能够使用、并且网络软件供应商能够实现的一套库函数调用和相关语义。

遵守这套 Windows Sockets 规范的网络软件,称之为 Windows Sockets 兼容的;而 Windows Sockets 兼容实现的提供者,我们称之为 Windows Sockets 提供者。一个网络软件供应商必须百分之百地实现 Windows Sockets 规范,才能做到与 Windows Sockets 兼容。

任何能够与 Windows Sockets 兼容实现协同工作的应用程序,就被认为是具有 Windows Sockets 接口,这种应用程序称为 Windows Sockets 应用程序。

Windows Sockets 规范定义并记录了如何使用 API 与 Internet 协议族(IPS,通常指的是 TCP/ IP)连接,尤其要指出的是所有的 Windows Sockets 实现都支持流套接口和数据报套接口。

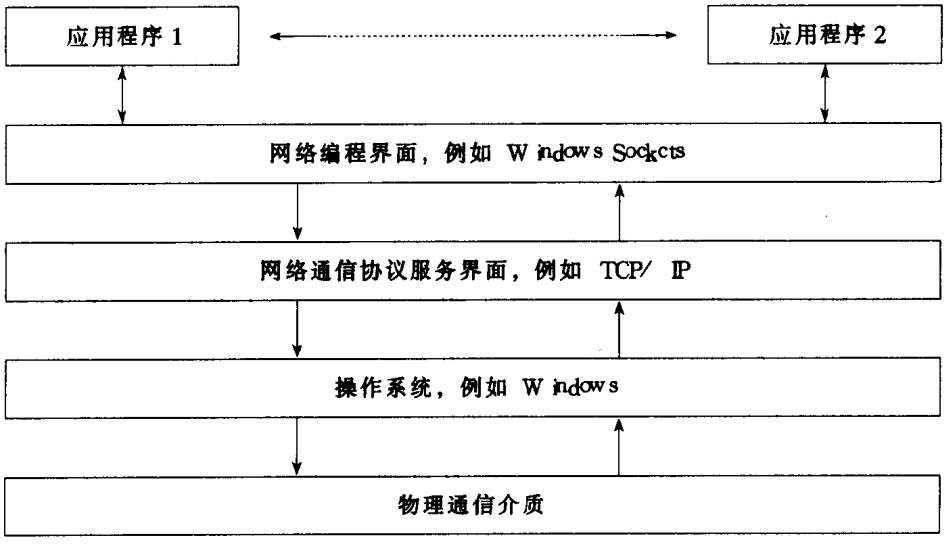


图 1- 1　应用程序与W indows Sockets关系图

应用程序调用 Windows Sockets 的 API 实现相互之间的通信。Windows Sockets 又利用下层的网络通信协议功能和操作系统调用实现实际的通信工作。它们之间的关系如图 1－1 所示。

虽然人们并不反对使用这一套 API 来实现另一通信协议栈(而且期望在将来规范的修改中能够讨论这个问题),但这种用法已经超出了这一份规范所规定的范围,本书在此将不作讨论。

1.2 Bekeley 套接口

Windows Sockets 规范是建立在 Bekeley 套接口模型上的。这个模型现在已是 TCP/IP 网络的标准。它提供了习惯于 UNIX 套接口编程的程序员极为熟悉的环境,并且简化了移植现有的基于套接口的应用程序源代码的工作。Windows Sockets API 也是与 4.3BSD 的要求相一致的。

1.3 Microsoft Windows 和针对 Windows 的扩展

Windows Sockets API 能够在所有 3.0 以上版本的 Windows 和所有 Windows Scokets 实现上使用,所以它不仅为 Windwos Sockets 实现和 Windows Sockets 应用程序提供了 16 位操作环境,而且也提供了 32 位操作环境。

Windows Sockets 也支持多线程的 Windows 进程。一个进程包含了一个或多个同时执行的线程。在 Windows 3.1 非多线程版本中,一个任务对应了一个仅具有单个线程的进程。本书所提到的线程均是指在多线程 Windows 环境中的真正意义的线程。在非多线程环境中(例如 Windows 3.0),这个术语是指 Windows Sockets 进程。

Windows Sockets 规范中的针对 Windows 的扩展部分,为应用程序开发者提供了开发具有 Windows 应用软件的功能。它有利于使程序员写出更加稳定并且更加高效的程序,也有助于在非占先 Windows 版本中使多个应用程序在多任务情况下更好地运作。除了 WSAStartup()和 WSACleanup()两个函数除外,其他的 Windows 扩展函数的使用不是强制性的。

1.4 这份规范的地位

Windows Sockets 是一份独立的规范。它的产生和存在是为了造益于应用程序开发者、网络软件供应商和广大计算机用户。这份规范的每一份正式出版的版本(非草稿)实际上代表了为网络软件供应商实现所需和应用程序开发者所用的一整套 API。关于这套规范的讨论和改进还正在进行之中。这样的讨论主要是通过 Internet 上的一个电子邮件论坛 - winsock@microdyne.com 进行的,同时也有不定期的会议举行。会议的具体内容会在电子邮件论坛上发表。

1.5 曾经作过的修改

1.5.1 Windows Sockets 1.0

Windows Sockets 1.0 代表了网络软件供应商和用户协会细致周到的工作的结晶。Windows Sockets 1.0 规范的发布是为了让网络软件供应商和应用程序开发者能够开始建立各自的符合 Windows Sockets 标准的实现和应用程序。

1.5.2 Windows Sockets 1.1

Windows Sockets 1.1 继承了 Windows Sockets 1.0 的准则和结构,并且仅在一些绝对必要的地方作了改动。这些改动都是基于不少公司在创作 Windows Sockets 1.0 实现时的经验和教训。Windows Scokets 1.1 包含了一些更加清晰的说明和对 Windows Sockets 1.0 的小改动。此外 1.1 版还包含了如下重大的变更:

* 加入了 gethostname()这个常规调用,以便更加简单地得到主机名字和地址。

* 定义 DLL 中小于 1000 的序数为 Windows Sockets 保留,而对大于 1000 的序数则没有限制。这使 Windows Sockets 供应商可以在 DLL 中加入自己的界面,而不用担心所选择的序数会与 Windows Scokets 将来的版本相冲突。

* 增加了 WSAStartup()函数和 WASCleanup()函数之间的关联,要求两个函数互相对应。这使得应用程序开发者和第三方 DLL 在使用 Windows Sockets 实现时不需要考虑其他程序对这套 API 的调用。

* 把函数 intr_addr()的返回类型,从结构 in_addr 改为了无符号长整型。这个改变是为了适应 Microsoft C 编译器和 Borland C 编译器对返回类型为 4 字节结构的函数的不同处理方法。

* 把 WSAAsyncSelect()函数语义从"边缘触发"改为"电平触发"。这种方式大大地简化了一个应用程序对这个函数的调用。

* 改变了 ioctlsocket()函数中 FIONBIO 的语义。如果套接口还有未完成的 WSAAsyncSelect()调用,该函数将失败返回。

* 为了符合 RFC 1122,在套接口选项中加入了 TCP_NODELAY 这一条。

所有 Windows Sockets 1.1 对于 Windows Sockets 1.0 的改动在以下都作了记号。

第二章　使用 Windows Sockets 1.1 编程

本章介绍如何使用 Windows Sockets 1.1 编程，并讨论了使用 Windows Sockets 1.1 编程的一些细节问题。本章的讨论均是基于 Windows Sockets 1.1 规范的，在某些方面可能会与第六、七章对 Windows Sockets 2 的讨论不一致，请读者注意这一区别。

2.1　Windows Sockets 协议栈安装检查

任何一个与 Windows Sockets Import Library 联接的应用程序只需简单地调用 WSAStartup()函数便可检测系统中有没有一个或多个 Windows Sockets 实现。而对于一个稍微聪明一些的应用程序来说，它会检查 PATH 环境变量来寻找有没有 Windows Sockets 实现的实例(Windows Sockets.DLL)。对于每一个实例，应用程序可以发出一个 LoadLibrary()调用并且用 WSAStartup()函数来得到这个实现的具体数据。

这一版本的 Windows Sockets 规范并没有试图明确地讨论多个并发的 Windows Sockets 实现共同工作的情况。但这个规范中没有限制多个 Windows Sockets DLL 同时存在，并且被一个或者多个应用程序同时调用。

2.2　套接口

2.2.1　基本概念

通信的基石是套接口，一个套接口是通信的一端。在这一端上你可以找到与其对应的一个名字。一个正在被使用的套接口都有它的类型和与其相关的进程。套接口存在于通信域中。通信域是为了处理一般的线程通过套接口通信而引进的一种抽象概念。套接口通常和同一个域中的套接口交换数据(数据交换也可能穿越域的界限，但这时一定要执行某种解释程序)。Windows Sockets 规范支持单一的通信域，即 Internet 域。各种进程使用这个域互相之间用 Internet 协议族来进行通信(Windows Sockets 1.1 以上的版本支持其他的域，例如 Windows Sockets 2)。

套接口可以根据通信性质分类，这种性质对于用户是可见的。应用程序一般仅在同一类的套接口间通信。不过只要底层的通信协议允许，不同类型的套接口间也照样可以通信。

用户目前可以使用两种套接口，即流套接口和数据报套接口。流套接口提供了双向的、有序的、无重复并且无记录边界的数据流服务。数据报套接口支持双向的数据流，但并不保证是可靠、有序、无重复的。也就是说，一个从数据报套接口接收信息的进程有可能发现信息重复了，或者和发出时的顺序不同。数据报套接口的一个重要特点是它保留了记录边界。对于这一特点，数据报套接口采用了与现在的许多包交换网络(例如以太网)非常类似的模型。

2.2.2 客户机/服务器模型

一个在建立分布式应用时最常用的范例便是客户机/服务器模型。在这种方案中客户应用程序向服务器程序请求服务。这种方式隐含了在建立客户机/服务器间通信时的非对称性。客户机/服务器模型工作时要求有一套为客户机和服务器所共识的惯例来保证服务能够被提供(或被接受)。这一套惯例包含了一套协议。它必须在通信的两头都被实现。根据不同的实际情况,协议可能是对称的或是非对称的。在对称的协议中,每一方都有可能扮演主从角色;在非对称协议中,一方被不可改变地认为是主机,而另一方则是从机。一个对称协议的例子是 Internet 中用于终端仿真的 TELNET。而非对称协议的例子是 Internet 中的 FTP。无论具体的协议是对称的或是非对称的,当服务被提供时必然存在"客户进程"和"服务进程"。

一个服务程序通常在一个众所周知的地址监听对服务的请求,也就是说,服务进程一直处于休眠状态,直到一个客户对这个服务的地址提出了连接请求。在这个时刻,服务程序被"惊醒"并且为客户提供服务——对客户的请求作出适当的反应。这一请求/响应的过程可以简单地用图 2-1 表示。虽然基于连接的服务是设计客户机/服务器应用程序时的标准,但有些服务也是可以通过数据报套接口提供的。

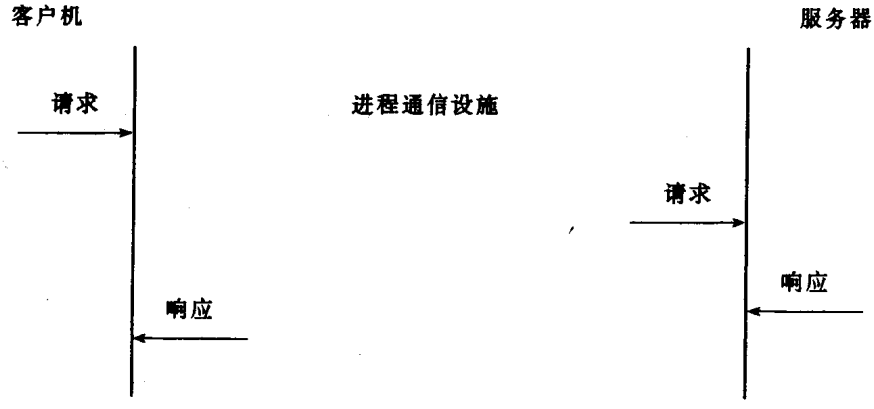


图 1-2 客户机/服务器模型

2.2.3 带外数据

注意:以下对于带外数据(也称为 TCP 紧急数据)的讨论,都是基于 BSD 模型而言的。用户和实现者必须注意,目前有两种互相矛盾的关于 RFC 793 的解释,也就是在这基础上,带外数据这一概念才被引入的。而且 BSD 对于带外数据的实现并没有符合 RFC 1122 定下的主机的要求,为了避免互操作时的问题,应用程序开发者最好不要使用带外数据,除非是与某一既成事实的服务互操作时所必须的。Windows Sockets 提供者也必须提供他们的产品对于带外数据实现的语义的文档(采用 BSD 方式或者是 RFC 1122 方式)。规定一个特殊的带外数据语义集已经超出了 Windows Sockets 规范的讨论范围。

流套接口的抽象中包括了带外数据这一概念,带外数据是相连的每一对流套接口间一个逻辑上独立的传输通道。带外数据是独立于普通数据传送给用户的,这一抽象要求带外数据设备必须支持每一时刻至少一个带外数据消息被可靠地传送。这一消息可能包含至少一个字节;并且在任何时刻仅有一个带外数据信息等候发送。对于仅支持带内数据的通信协议来说(例如紧急数据是与普通数据在同一序列中发送的),系统通常把紧急数据从普通数据中分离

出来单独存放。这就允许用户可以在顺序接收紧急数据和非顺序接收紧急数据之间作出选择(非顺序接收时可以省去缓存重叠数据的麻烦)。在这种情况下,用户也可以"偷看一眼"紧急数据。

某一个应用程序也可能喜欢线内处理紧急数据,即把其作为普通数据流的一部分。这可以靠设置套接口选项中的 SO_OOBINLINE 来实现(参见 5.1.21 节,setsockopt())。在这种情况下,应用程序可能希望确定未读数据中的哪一些是"紧急的("紧急"这一术语通常应用于线内带外数据)。为了达到这个目的,在 Windows Sockets 的实现中就要在数据流保留一个逻辑记号来指出带外数据从哪一点开始发送,一个应用程序可以使用 SIOCATMARK ioctlsocket()命令(参见 5.1.12 节)来确定在记号之前是否还有未读入的数据。应用程序可以使用这一记号与其对方进行重新同步。

WSAAsyncSelect()函数可以用于处理对带外数据到来的通知。

2.2.4 广播

数据报套接口可以用来向许多系统支持的网络发送广播数据包。要实现这种功能,网络本身必须支持广播功能,因为系统软件并不提供对广播功能的任何模拟。广播信息将会给网络造成极重的负担,因为它们要求网络上的每台主机都为它们服务,所以发送广播数据包的能力被限制于那些用显式标记了允许广播的套接口中。广播通常是为了如下两个原因而使用的:一个应用程序希望在本地网络中找到一个资源,而应用程序对该资源的地址又没有任何先验的知识;一些重要的功能,例如,要求把它们的信息发送给所有可以找到的邻机。

被广播信息的目的地址取决于这一信息将在何种网络上广播。Internet 域中支持一个速记地址用于广播 - INADDR_BROADCAST。由于使用广播以前必须捆绑一个数据报套接口,所以所有收到的广播消息都带有发送者的地址和端口。

某些类型的网络支持多种广播的概念。例如 IEEE802.5 令牌环结构便支持链接层广播指示,它用来控制广播数据是否通过桥接器发送。Windows Sockets 规范没有提供任何机制用来判断某个应用程序是基于何种网络之上的,而且也没有任何办法来控制广播的语义。

2.3 字节顺序

Intel 处理器的字节顺序是与 DEC VAX 处理器的字节顺序相一致的。因此它与 68000 型处理器以及 Internet 的顺序是不同的,用户在使用时要特别小心以保证正确的顺序。

任何从 Windows Sockets 函数对 IP 地址和端口号的引用与传送给 Windows Sockets 函数的 IP 地址和端口号均是按照网络顺序组织的,这也包括了 sockaddr_in 结构这一数据类型中的 IP 地址域和端口域(但不包括 sin_family 域)。

考虑到一个应用程序通常用与"时间"服务对应的端口来和服务器连接,而服务器提供某种机制来通知用户使用另一端口。因此 getservbyname()函数返回的端口号已经是网络顺序了,可以直接用来组成一个地址,而不需要进行转换。然而如果用户输入一个数,而且指定使用这一端口号,应用程序就必须在使用它建立地址以前,把它从主机顺序转换成网络顺序(使用 htons()函数)。相应地,如果应用程序希望显示包含于某一地址中的端口号(例如从 getpeername()函数中返回的),这一端口号就必须在被显示前从网络顺序转换到主机顺序(使用 ntohs()函数)。

由于 Intel 处理器和 Internet 的字节顺序是不同的，上述的转换是无法避免的，应用程序的编写者应该使用作为 Windows Sockets API 一部分的标准的转换函数，而不要使用自己的转换函数代码。因为将来的 Windows Sockets 实现有可能在主机字节顺序与网络字节顺序相同的机器上运行。因此只有使用标准的转换函数的应用程序是可移植的。

2.4 套接口属性选项

Windows Sockets 规范支持的套接口属性选项都列在对 setsockopt()函数和 getsockopt()函数的叙述中。任何一个 Windows Sockets 实现必须能够识别所有这些属性选项，并且对每一个属性选项都返回合理的数值。每一个属性选项的缺省值列在下表中：

选 项	类 型	含 义	缺省值	注意事项
SO_ACCEPTCON	BOOL	套接口正在监听	FALSE	
SO_BROADCAST	BOOL	套接口被设置为可以发送广播数据	FALSE	
SO_DEBUG	BOOL	允许 Debug	FALSE	(*)
SO_DONTLINGER	BOOL	如果为真，SO_LINGER 选项被禁止	TRUE	
SO_DONTROUTE	BOOL	路由被禁止	FALSE	(*)
SO_ERROR	int	得到并且清除错误状态	0	
SO_KEEPALIVE	BOOL	活跃信息正在被发送	FALSE	
SO_LINGER	struct linger	返回目前的 linger 信息	l_onoff 为 0	
SO_OOBINLINE	FAR * BOOL	带外数据正在普通数据流中被接收	FALSE	
SO_RCVBUF	int	接收缓冲区大小	决定于实现	(*)
SO_REUSEADDR	BOOL	该套接口捆绑的地址是否可被其他人使用	FALSE	
SO_SNDBUF	int	发送缓冲区大小	决定于实现	(*)
SO_TYPE	int	套接口类型(如 SOCK_STREAM)	和套接口被创建时一致	
TCP_NODELAY	BOOL	禁止采用 Nagle 进行合并传送	决定于实现	

(*) Windows Sockets 实现有可能在用户调用 setsockopt()函数时忽略这些属性，并且在用户调用 getsockopt()函数时返回一个没有变化的值；或者它可能在 setsockopt()时接受某个值，并且在 getsockopt()时返回相应的数值，但事实上并没有在任何地方使用它。

2.5 数据库文件

getXbyY()和 WSASyncGetXByY()这一类的例程是用来得到某种特殊的网络信息的。getXbyY()例程最初(在第一版的 BERKELY UNIX 中)是被设计成一种在文本数据库中查询信息的机制。虽然 Windows Sockets 实现可能用不同的方式来得到这些信息，但 Windows

Sockets 应用程序要求通过 getXbyY()或 WSASyncGetXByY()这一类例程得到的信息是一致的。

2.6 与 Berkeley 套接口的不同

在很有限的一些地方,Windows Sockets API 必须从严格地坚持 Berkeley 传统风格中解放出来。通常这么做是因为在 Windows 环境中实现的难度。

2.6.1 套接口数据类型和错误数值

Windows Sockets 规范中定义了一个新的数据类型 SOCKET,这一类型的定义对于将来 Windows Sockets 规范的升级是必要的。例如在 Windows NT 中把套接口作为文件句柄来使用。这一类型的定义也保证了应用程序向 Win32 环境的可移植性。因为这一类型会自动地从 16 位升级到 32 位。

在 UNIX 中所有句柄包括套接口句柄,都是非负的短整数,而且一些应用程序把这一假设视为真理。Windows Sockets 句柄则没有这一限制,除了 INVALID_SOCKET 不是一个有效的套接口外,套接口可以取从 0 到 INVALID_SOCKET - 1 之间的任意值。

因为 SOCKET 类型是 unsigned,所以编译已经存在于 UNIX 环境中的应用程序的源代码可能会导致 signed/unsigned 数据类型不匹配的警告。这还意味着,在 socket()例程和 accept()例程返回时,检查是否有错误发生就不应该再使用把返回值和 - 1 比较或判断返回值是否为负的方法(这两种方法在 BSD 中都是很普通、很合法的途径)。取而代之的是,一个应用程序应该使用常量 INVALID_SOCKET,该常量已在 WINSOCK.H 中定义。

例如:典型的 BSD 风格:

```
s = socket(...);
if (s == -1)        /* of s<0 */
    {...}
```

更优良的风格:

```
s = socket(...);
if (s == INVALID_SOCKET)
    {...}
```

2.6.2 select()函数和 FD_*宏

由于一个套接口不再表示了 UNIX 风格的小的非负的整数,select()函数在 Windows Sockets API 中的实现有一些变化:每一组套接口仍然用 fd_set 类型来代表,但是它并不是一个位掩码。整个组的套接口是用了一个套接口的数组来实现的。为了避免潜在的危险,应用程序应该坚持用 FD_XXX 宏来设置,初始化、清除和检查 fd_set 结构。

2.6.3 错误代码——errno,h_errno,WSAGetLastError()

Windows Sockets 实现所设置的错误代码是无法通过 errno 变量得到的。对于 getXbyY()这一类的函数,错误代码无法从 h_errno 变量得到,错误代码可以使用 WSAGetLastError()调用得到。这一函数将在 5.3.11 节中讨论。这个函数在 Windows Sockets 实现中是作为 Win32