
学校的理想装备
电子图书·学校专集
校园网上的最佳资源
Windows Sockets 规范及应用
—Windows 网络编程接口
BOOK
内部资料 非卖品

Windows Sockets 规范及应用

—Windows 网络编程接口

施 炜　　李 铮　　秦 颖　　编著

****

****

****

****

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、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。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。

作者声明

由于成书时间紧迫。本书不免有许多错误和不当之处，故此作者衷心希望各位读者能对本书提出宝贵意见（包括补充新的应用实例和内容），以便我们进一步修改完善此书。我们会尊重相应修订者的版权。作者也衷心希望在我们和各位读者的努力下，本书能够成为一本关于 Windows Sockets 编程的系统而又准确的免费中文参考书，为广大读者在 Windows 下网络编程提供帮助。

作者联系地址：

施炜：上海交通大学 94032 班　　（200030）

Email: weishi@fudan.ihep.ac.cn

李铮：上海交通大学自动化系　（200030）

Email: blee.bbs@captain.net.tsinghua.edu.cn

秦颖：上海交通大学 94033A 班　（200030）

Email: fluke.bbs@captain.net.tsinghua.edu.cn

作者希望每一位拿到本书的读者能以任何方式通知我们。以便我们掌握本书的应用情况。并敬请各位读者暂时不要在其他 FTP 站点散发，谢谢合作。

编著者

1996 年 5 月 20 日

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所谓“信息基础设施”。网络化是计算机技术九十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 个国家的五千多个网络连入 Internet，其中包含了数千个组织的 30 万台主机，用户数以百万计。

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四千万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。

编著者

1995 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，以使程序员能充分地利用 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，我们称之为 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 规范才能做到现 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。我们称这种应用程序为 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，通常我们指的是 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.

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

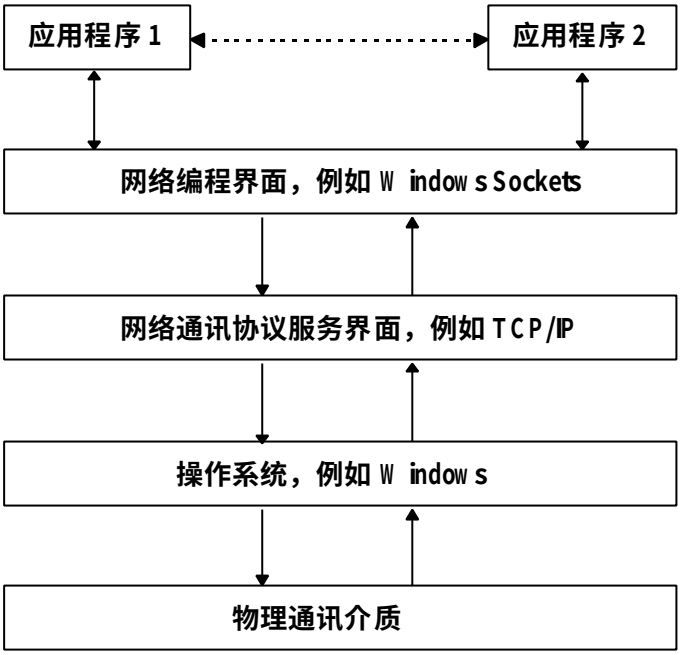

图 1-1 应用程序与 Windows Sockets 关系图

虽然我们并不反对使用这一套 API 来实现另一通讯协议栈（而且我们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我们这一份规范所规定的范围，我们在此将不作讨论。

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 也是和 4.3BSD 的要求一致的。

1.3 Microsoft Windows 和针对 Windows 的扩展

这一套 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。而我们在本书中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的。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 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和 Windows Scokets 将来的版本冲突。

* 增加了 WSAStartup()函数和 WASClearup()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四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

在这一章，我们将介绍如何使用 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，在某些方面可能会和第六、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。但这个规范中没有任何规定可以被解释成是限制多个 Windows Sockets DLL 同时存在并且被一个或者多个应用程序同时调用的。

2.2 套接口

2.2.1 基本概念

通讯的基石是套接口，一个套接口是通讯的一端。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。套接口存在于通讯域中。通讯域是为了处理一般的线程通过套接口通讯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 规范支持单一的通讯域，即 Internet 域。各种进程使用这个域互相之间用 Internet 协议族来进行通讯（Windows Sockets 1.1 以上的版本支持其他的域，例如 Windows Sockets 2）。

套接口可以根据通讯性质分类；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讯。不过只要底层的通讯协议允许，不同类型的套接口

间也照样可以通讯。

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/服务器间通讯时的非对称性。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。它必须在通讯的两头都被实现。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/相应的过程可以简单的用图 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。

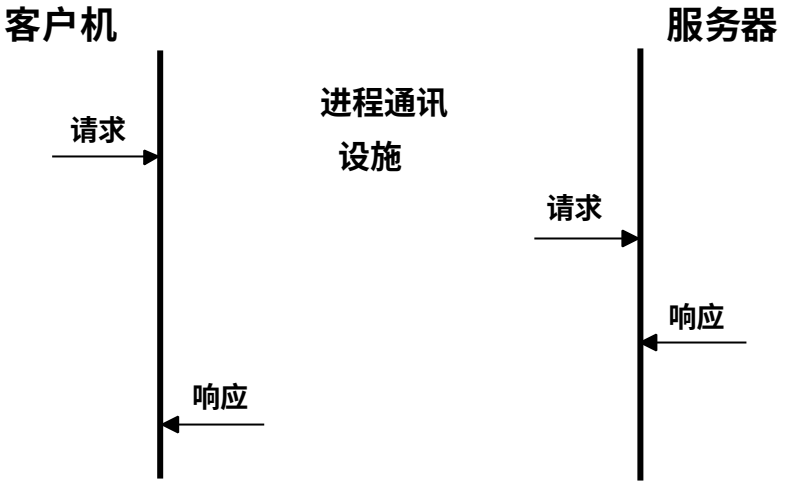

图 2-1 客户机/服务器模型

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挡（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讯协议来说（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：1．一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。2．一些重要的功能，例如路由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和 DEC VAX 处理器的字节顺序一致的。因此它与 68000 型处理器以及 Internet 的顺序是不同的，所以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和传送给 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，应用程序则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	

	FAR *		
SO_OOBINLINE	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 WSAAyncGetXByY()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 WSAAyncGetXByY()这一类例程得到的信息是一致。

2.6 与 Berkeley 套接口的不同

有一些很有限的地方，Windows Sockets API 必须与从严格地坚持 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 Win/32 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 比较的方法，或判断返回值是否为负（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。另外对于 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。这一函数在 5.3.11 中讨论。这个函数在 Windows Sockets 实现中是作为 WIN/32 函数 GetLastError()的先导函数（最终是一个别名）。这样做是为了在多线程的进程中为每一线程得到自己的错误信息提供可靠的保障。

为了保持与 BSD 的兼容性，应用程序可以加入以下一行代码：

```
#define errno WSAGetLastError()
```

这就保证了用全程的 errno 变量所写的网络程序代码在单线程环境中可以正确使用。当然，这样做有许多明显的缺点：如果一个原程序包含了一段代码对套接口和非套接口函数都用 errno 变量来检查错误，那么这种机制将无法工作。此外，一个应用程序不可能为 errno 赋一个新的值（在 Windows Sockets 中，WSASetLastError()函数可以做到这一点）。

例如：

典型的 BSD 风格：

```
r = recv(...);
if (r == -1                                /* 但请见下文 */
    && errno == EWOULDBLOCK)
   {...}
```

更优良的风格：

```
r = recv(...);
if (r == -1                                 /* 但请见下文 */
    && WSAGetLastError() == EWOULDBLOCK)
   {...}
```

虽然为了兼容性原因，错误常量与 4.3BSD 所提供的一致；应用程序应该尽可能地使用“WSA”系列错误代码定义。例如，一个更准确的上面程序片断的版本应该是：

```
r = recv(...);
if (r == -1                                    /* 但请见下文 */
    && WSAGetLastError() == WSAEWOULDBLOCK)
   {...}
```

2.6.4 指针

所有应用程序与 Windows Sockets 使用的指针都必须是 FAR 指针，为了方便应用程序开发者使用，Windows Sockets 规范定义了数据类型 LPHOSTENT。

2.6.5 重命名的函数

有两种原因 Berkeley 套接口中的函数必须重命名以避免与其他的 API 冲突：

2.6.5.1 close()和 closesocket()

在 Berkeley 套接口中，套接口出现的形式与标准文件描述字相同，所以 close()函数可以用来和关闭正规文件一样来关闭套接口。虽然在 Windows Sockets API 中，没有任何规定阻碍 Windows Sockets 实现用文件句柄来标识套接口，但是也没有任何规定要求这么做。套接口描述字并不认为是和正常文件句柄对应的,而且并不能认为文件操作，例如 read()，write()和 close()在应用于套接口后不能保证正确工作。套接口必须使用 closesocket()例程来关闭，用 close()例程来关闭套接口是不正确的，这样做的效果对于 Windows Sockets 规范说来也是未知的。

2.6.5.2 ioctl()和 iooctlsocket()

许多 C 语言的运行时系统出于与 Windows Sockets 无关的目的使用 ioctl()例程，所以 Windows Sockets 定义 ioctlsocket()例程。它被用于实现 BSD 中用 ioctl()和 fcntl()实现的功能。

2.6.6 阻塞例程和 EINPROGRESS 宏

虽然 Windows Sockets 支持关于套接口的阻塞操作，但是这种应用是被强烈反对的.如果程序员被迫使用阻塞模式（例如一个准备移植的已有的程序），那么他应该清楚地知道Windows Sockets中阻塞操作的语义。有关细节请参见4.1.1

2.6.7 Windows Sockets 支持的最大套接口数目

一个特定的 Windows Sockets 提供者所支持的套接口的最大数目是由实现确定的。任何一个应用程序都不应假设某个待定数目的套接口可用。这一点在 4.3.15 WSAStartup()中会被重申。而且一个应用程序可以真正使用的套接口的数目和某一特定的实现所支持的数目是完全无关的。

一个 Windows Sockets 应用程序可以使用的套接口的最大数目是在编译时由常量 FD_SETSIZE 决定的。这个常量在 select()函数（参见 4.1.18）中被用来组建 fd_set 结构。在 WINSOCK.H 中缺省值是 64。如果一个应用程序希望能够使用超过 64 个套接口，则编程人员必须在每一个源文件包含 WINSOCK.H 前定义确切的 FD_SET 值。有一种方法可以完成这项工作，就是在工程项目文件中的编译器选项上加入这一定义。例如在使用 Microsoft C 时加入-D FD_SETSIZE=128 作为编译命令的一个命令行参数.要强调的是：FD_SET 定义的值对 Windows Sockets 实现所支持的套接口的数目并无任何影响。

2.6.8 头文件

为了方便基于 Berkeley 套接口的已有的源代码的移植，Windows Sockets 支持许多 Berkeley 头文件。这些 Berkeley 头文件被包含在 WINSOCK.H 中。所以一个 Windows Sockets 应用程序只需简单的包含 WINSOCK.H 就足够了（这也是一种被推荐使用的方法）。

2.6.9 API 调用失败时的返回值

常量 SOCKET_ERROR 是被用来检查 API 调用失败的。虽然对这一常量的使用并不是强制性的，但这是推荐的。如下的例子表明了如何使用 SOCKET_ERROR 常量

典型的 BSD 风格：

```
r = recv(...);
if (r == -1                    /* or r < 0 */
    && errno == EWOULDBLOCK)
   {...}
```

更优良的风格：

```
r = recv(...);
if (r == SOCKET_ERROR
```

```
    && WSAGetLastError == WSAEWOULDBLOCK)
    {...}
```

2.6.10 原始套接口

Windows Sockets 规范并没有规定 Windows Sockets DLL 必须支持原始套接口－用 SOCK_RAW 打开的套接口。然而 Windows Sockets 规范鼓励 Windows Sockets DLL 提供原始套接口支持。一个 Windows Sockets 兼容的应用程序在希望使用原始套接口时应该试图用 socket()调用（参见 5.1.23 节）来打开套接口。如果这么做失败了，应用程序则应该使用其他类型的套接口或向用户报告错误。

2.7 在多线程 Windows 版本中的 Windows Sockets

Windows Sockets 接口被设计成既能够在单线程的 Windows 版本（例如 Windows 3.1）又能够在占先的多线程 Windows 版本（例如 Windows NT）中使用，在多线程环境中，套接口接口基本上是不变的。但多线程应用程序的作者必须知道，在线程之间同步对套接口的使用是应用程序的责任，而不是 Windows Sockets 实现的责任。这一点在其他形式的 I/O 中管理，例如文件 I/O 中是一样的。没有对套接口调用进行同步将导致不可预测的结果。例如，如果有两个线程同时调用同一套接口进行 send()，那么数据发送的先后顺序就无法保证了。

在一个线程中关闭一个未完成的阻塞的套接口将会导致另一个线程使用同一套接口的阻塞调用出错（WSAEINTER）返回，就象操作被取消一样。这也同样适用于某一个 select()调用未完成时，应用程序关闭了其中的一个被选择的套接口。

在占先的多线程 Windows 版本中，并没有缺省的阻塞钩子函数。这是因为如果一个单一的应用程序在等待某一操作结束时并不会调用 PeekMessage()或 GetMessage()这些会使应用程序产生一个非占先窗口的函数。因此机器在这种情况下不会被阻塞。然而，为了向后的兼容性，在多线程 Windows 版本中，WSASetBlockingHook()函数也被实现了。任何使用缺省阻塞钩子的应用程序可以安装它们自己的阻塞钩子函数来覆盖缺省的阻塞钩子函数。

第三章 Windows Sockets 1.1 应用实例

在本章中，作者的实际工作为背景，给出了一个使用 Windows Sockets 1.1 编程的具体例子。并对这个例子作了详细的分析。这个例子在 Windows 3.1、Windows Sockets 1.1 和 BSD OS for PC 2.0（BSD UNIX 微机版）环境下调试通过。

3.1 套接口网络编程原理

套接口有三种类型:流式套接口,数据报套接口及原始套接口.

流式套接口定义了一种可靠的面向连接的服务,实现了无差错无重复的顺序数据传输.数据报套接口定义了一种无连接的服务,数据通过相互独立的报文进行传输,是无序的,并且不保证可靠,无差错.原始套接口允许对低层协议如IP或ICMP直接访问,主要用于新的网络协议实现的测试等.

无连接服务器一般都是面向事务处理的,一个请求一个应答就完成了客户程序与服务程序之间的相互作用。若使用无连接的套接口编程，程序的流程可以用图3-1表示。

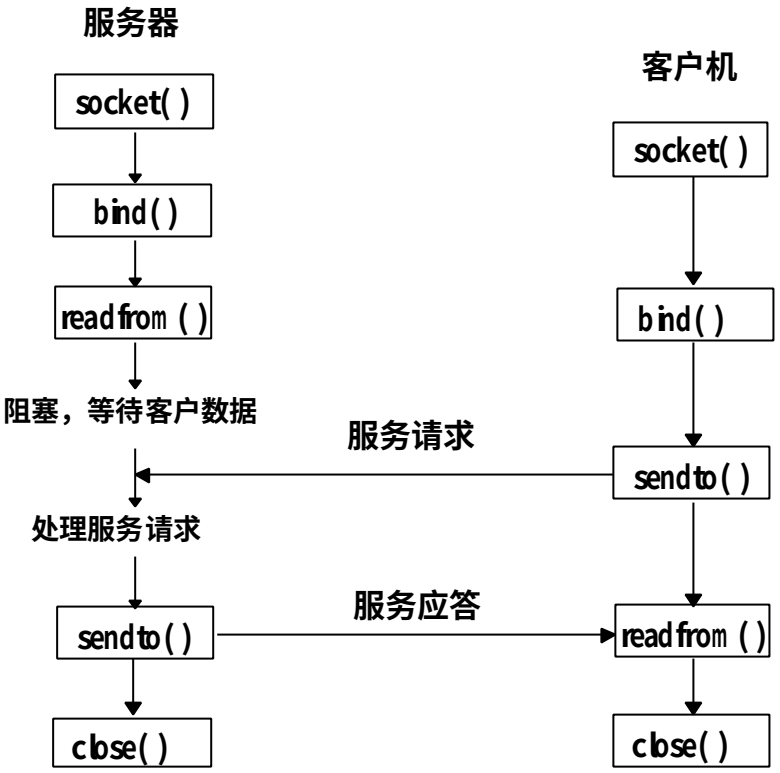

图 3-1 无连接套接口应用程序时序图

面向连接服务器处理的请求往往比较复杂，不是一来一去的请求应答所能解决的，而且往往是并发服务器。使用面向连接的套接口编程，可以通过图3-1来表示：其时序。

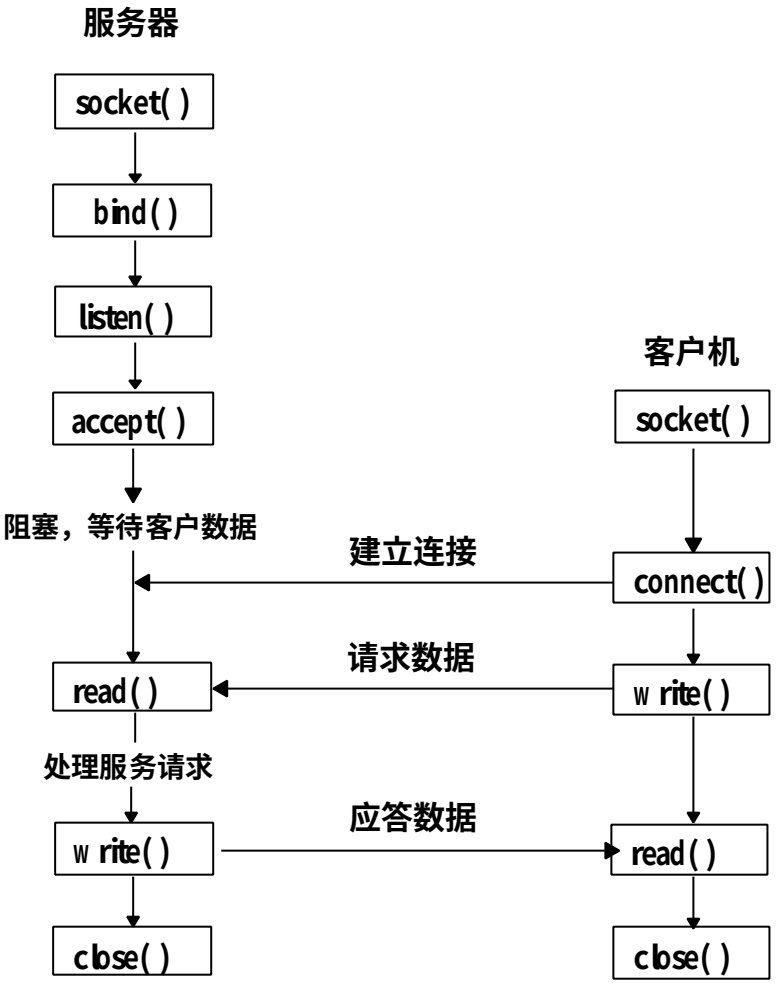

图 3-2 面向连接套接口应用程序时序图

套接口工作过程如下:服务器首先启动,通过调用socket()建立一个套接口,然后调用bind()将该套接口和本地网络地址联系在一起,再调用listen()使套接口做好侦听的准备,并规定它的请求队列的长度,之后就调用accept()来接收连接.客户在建立套接口后就可调用connect()和服务器建立连接.连接一旦建立,客户机和服务器之间就可以通过调用read()和write()来发送和接收数据.最后,待数据传送结束后,双方调用close()关闭套接口.

3.2 Windows Sockets 编程原理

由于Windows的基于消息的特点,WINSOCK和BSD套接口相比,有如下一些新的扩充:

1.异步选择机制

异步选择函数WSAAsyncSelect()允许应用程序提名一个或多个感兴趣的网络事件,如FD_READ,FD_WRITE,FD_CONNECT,FD_ACCEPT等等代表的网络事件.当

被提名的网络事件发生时,Windows应用程序的窗口函数将收到一个消息.这样就可以实现事件驱动了.

2.异步请求函数

异步请求函数允许应用程序用异步方式获得请求的信息,如WSAAsyncGetXByY()类函数. 这些函数是对BSD标准函数的扩充.函数WSACancelAsyncRequest()允许用户中止一个正在执行的异步请求.

3.阻塞处理方法

WINSOCK提供了"钩子函数"负责处理Windows消息,使Windows的消息循环能够继续.WINSOCK提供了两个函数(WSASetBlockingHook()和WSAUnhookBlockingHook())让应用程序设置或取消自己的"钩子函数".函数WSAIsBlocking()可以检测是否阻塞,函数WSACancelBlockingCall()可以取消一个阻塞的调用.

4.错误处理

WINSOCK提供了两个WSAGetLastError()和WSASetLastError()来获取和设置最近错误号.

5.启动和终止

由于Windows Sockets的服务是以动态连接库WINSOCK.DLL形式实现的,所以必须要先调用WSAStartup()函数对Windows Sockets DLL进行初始化,协商WINSOCK的版本支持,并分配必要的资源.在应用程序关闭套接口后,还应调用WSACleanup()终止对Windows Sockets DLL的使用,并释放资源,以备下一次使用.

在这些函数中,实现Windows网络实时通信的关键是异步选择函数WSAAsyncSelect()的使用. 用法及详细说明参见第5.3.7.

3.3 Windows Sockets 与 UNIX 套接口编程实例

下面是一个简单的基于连接的点对点实时通信程序.它由两部分组成,服务器在主机UNIX下直接运行, 客户机在Windows下运行.

3.3.1 SERVER 介绍

由于SERVER是在UNIX下运行的,它对套接口的使用都是BSD的标准函数,程序也比较简单, 只有一段程序,下面简要解释一下.

首先,建立自己的套接口.在互连网的进程通信中,全局标识一个进程需要一个被称为"半相关"的三元组(协议,本地主机地址,本地端口号)来描述,而一个完整的进程通信实例则需要一个被称为"相关"的五元组(协议, 本地主机地址,本地端口号,远端主机地址,远端端口号)来描述.

```
s=socket(AF_INET, SOCK_STREAM, 0)
```

该函数建立指定地址格式,数据类型和协议下的套接口,地址格式为AF_INET(唯一支持的格式),数据类型SOCK_STREAM表示建立流式套接口,参数三为0,即协议缺省.

```
bind(s, (struct sockaddr *)&server, sizeof(server))
```

该函数将建立服务器本地的半相关,其中,server是sockaddr_in结构,其成员描述了本地端口号和本地主机地址,经过bind()将服务器进程在网上标识出来.

然后,建立连接.先是调用listen()函数表示开始侦听.再通过accept()调用等待接收连接.

listen(s,1)表示连接请求队列长度为1,即只允许有一个请求,若有多个请求,则出现错误,给出错误代码WSAECONNREFUSED.

```
ns = accept(s, (struct sockaddr *)&client, &namelen))
```

accept()阻塞(缺省)等待请求队列中的请求,一旦有连接请求来,该函数就建立一个和s有相同属性的新的套接口.client也是一个sockaddr_in结构,连接建立时填入请求连接的套接口的半相关信息.

接下来,就可以接收和发送数据了.

```
recv(ns,buf,1024,0)
send(ns,buf,pktlen,0)
```

上面两个函数分别负责接收和发送数据,recv从ns(建立连接的套接口)接收数据放入buf中,send则将buf中数据发送给ns.至于第四个参数,表示该函数调用方式,可选择MSG_DONTROUTE和MSG_OOB, 0表示缺省.

最后,关闭套接口.

```
close(ns);
close(s);
```

3.3.2 CLIENT 介绍

客户端是在Windows上运行的,使用了一些Windows Sockets的扩展函数,稍微复杂一些.包括了.RC和.C两个文件,其中的主窗口函数ClientProc()是程序的主要部分,下面简单解释一下.

首先,是在WinMain()中建立好窗口后,即向主窗口函数发一条自定义的WM_USER消息, 做相关的准备工作.在主窗口函数中,一接收到WM_USER消息,首先调用WSAStartup()函数初始化Windows Sockets DLL,并检查版本号.如下:

```
Status = WSAStartup(VersionReqd, lpmyWSAData);
```

其中,VersionReqd描述了WINSOCK的版本(这里为1.1版),lpmyWSAData指向一个WSADATA结构,该结构描述了Windows Sockets的实现细节.

WSAStartup()之后,进程通过主机名(运行时命令行参数传入)获取主机地址,如下:

```
hostaddr = gethostbyname(server_address);
```

hostaddr指向hostent结构,内容参见5.2.1.

然后,进程就不断地消息循环,等待用户通过菜单选择"启动".这时,通过调用Client()来启动套接口.在Client()中,首先也是调用socket()来建立套接口.如下:

```
if ((s = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
{
    AlertUser(hWnd, "Socket Failed");
    return (FALSE);
}
```

紧接着,调用WSAAsyncSelect()函数提名FD_CONNECT网络事件,如下:

```
if (!SetSelect(hWnd, FD_CONNECT))
    return (FALSE);
```

SetSelect()主要就是调用WSAASyncSelect(),让Windows Sockets DLL在侦

测到连接建立时,就发送一条UM_SOCK的自定义消息,使消息循环继续下去.如下:

```
BOOL SetSelect(HWND hWnd, long lEvent)
{
    if (WSAAsyncSelect(s, hWnd, UM_SOCK, lEvent) == SOCKET_ERROR)
    {
        AlertUser(hWnd, "WSAAsyncSelect Failure.");
        return (FALSE);
    }
    return (TRUE);
}
```

为建立连接,必须马上调用connect()如下,由于先调用了WSAASyncSelect(),connect()便是非阻塞调用.进程发出连接请求后就不管了,当连接建立好后,WINSOCK DLL自动发一条消息给主窗口函数,以使程序运行下去.

```
connect(s, (struct sockaddr FAR *)&dst_addr, sizeof(dst_addr));
```

窗口函数在收到UM_SOCK消息后,判断是由哪个网络事件引起的,第一次,必然是由连接事件引起的,这样,就会执行相应的程序段,同样调用SetSelect()来提名FD_WRITE事件.希望在套接口可发送数据时接到消息.在收到FD_WRITE消息时,先调用send()发送数据,再调用SetSelect()来提名FD_READ事件, 希望在套接口可接收数据是接到消息.在收到FD_READ消息时,先调用recv()来接收数据再提名FD_WRITE事件,如此循环下去.直到发生连接关闭的事件FD_CLOSE,这时就调用WSAAsyncSelect(s,hWnd,0,0)来停止异步选择.在窗口函数接到WM_DESTROY消息时(即关闭窗口之前),先调用closesocket()(作用同UNIX 中的close())来关闭套接口,再调用WSACleanup()终止Windows Sockets DLL,并释放资源.

3.3.3 源程序清单

程序1:CLIENT.RC

```
ClientMenu MENU
BEGIN
    POPUP "&Server"
    BEGIN
        MENUITEM "&Start...", 101
        MENUITEM "&Exit",  102
    END
END
```

程序2:CLIENT.C

```
#define USERPORT 10001
#define IDM_START 101
#define IDM_EXIT  102
#define UM_SOCK WM_USER + 0X100

#include <alloc.h>
#include <mem.h>
#include <windows.h>
```

```
#include <winsock.h>
#define MAJOR_VERSION 1
#define MINOR_VERSION 2
#define WSA_MAKEWORD(x,y)  ((y)*256+(x))

HANDLE hInst;
char server_address[256] = {0};
char buffer[1024];
char FAR * lpBuffer = &buffer[0];
SOCKET s = 0;
struct sockaddr_in dst_addr;
struct hostent far *hostaddr;
struct hostent hostnm;
struct servent far *sp;
int count = 0;

BOOL InitApplication(HINSTANCE hInstance);
long FAR PASCAL ClientProc(HWND hWnd, unsigned message, UINT wParam, LONG
lParam);
void AlertUser(HWND hWnd, char *message);

BOOL Client(HWND hWnd);
BOOL ReceivePacket(HWND hWnd);
BOOL SetSelect(HWND hWnd, long lEvent);
BOOL SendPacket(HWND hWnd, int len);

int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow)
{
    HWND hWnd;
    MSG msg;

    lstrcpy((LPSTR)server_address, lpCmdLine);
    if (!hPrevInstance)
        if (!InitApplication(hInstance))
            return (FALSE);
    hInst = hInstance;
    hWnd = CreateWindow("ClientClass", "Windows ECHO Client",
WS_OVERLAPPEDWINDOW,\
        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL,
NULL,\
        hInstance, NULL);
    if (!hWnd)
        return (FALSE);
    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);
```

```
    PostMessage(hWnd, WM_USER, (WPARAM)0, (LPARAM)0);
    while (GetMessage(&msg, NULL, NULL, NULL))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return (msg.wParam);
}

BOOL InitApplication(HINSTANCE hInstance)
{
     WNDCLASS WndClass;

    char *szAppName = "ClientClass";
     // fill in window class information

    WndClass.lpszClassName = (LPSTR)szAppName;
    WndClass.hInstance     = hInstance;
    WndClass.lpfnWndProc   = ClientProc;
    WndClass.hCursor       = LoadCursor(NULL, IDC_ARROW);
    WndClass.hIcon         = LoadIcon(hInstance, NULL);
    WndClass.lpszMenuName  = "ClientMenu";
    WndClass.hbrBackground = GetStockObject(WHITE_BRUSH);
    WndClass.style         = CS_HREDRAW | CS_VREDRAW;
    WndClass.cbClsExtra    = 0;
    WndClass.cbWndExtra    = 0;

     // register the class

     if (!RegisterClass(&WndClass))
        return(FALSE);

     return(TRUE);
}

long FAR PASCAL ClientProc(HWND hWnd, unsigned message, UINT wParam, LONG
lParam)
{
    int length, i;
    WSADATA wsaData;

    int Status;

    switch (message)
    {
        case WM_USER:
```

```
        {
            WORD   wMajorVersion, wMinorVersion;
            LPWSADATA  lpmyWSAData;
            WORD       VersionReqd;
            int        ret;

            wMajorVersion = MAJOR_VERSION;
            wMinorVersion = MINOR_VERSION;
            VersionReqd = WSA_MAKEWORD(wMajorVersion,wMinorVersion);

            lpmyWSAData = (LPWSADATA)malloc(sizeof(WSADATA));
            Status = WSAStartup(VersionReqd, lpmyWSAData);
            if (Status != 0)
            {
                AlertUser(hWnd, "WSAStartup() failed\n");
                PostQuitMessage(0);
            }
            hostaddr = gethostbyname(server_address);
            if (hostaddr == NULL)
            {
                AlertUser(hWnd, "gethostbyname ERROR!\n");
                WSACleanup();
                PostQuitMessage(0);
            }
            _fmemcpy(&hostnm, hostaddr, sizeof(struct hostent));
        }
            break;
        case WM_COMMAND:
            switch (wParam)
            {
                case IDM_START:
                    if (!Client(hWnd))
                    {
                        closesocket(s);
                        AlertUser(hWnd, "Start Failed");
                    }
                    break;
                case IDM_EXIT:
//                  WSACleanup();
                    PostQuitMessage(0);
                    break;
            }
            break;
        case UM_SOCK:
            switch (lParam)
            {
```

```
        case FD_CONNECT:
            if (!SetSelect(hWnd, FD_WRITE))
               closesocket(s);
            break;
        case FD_READ:
            if (!ReceivePacket(hWnd))
            {
               AlertUser(hWnd, "Receive Packet Failed.\n");
               closesocket(s);
               break;
            }
            if (!SetSelect(hWnd, FD_WRITE))
               closesocket(s);
            break;
        case FD_WRITE:
            for (i = 0; i < 1024; i ++)
               buffer[i] = (char)'A' + i % 26;
            length = 1024;
            if (!SendPacket(hWnd, length))
            {
               AlertUser(hWnd, "Packet Send Failed!\n");
               closesocket(s);
               break;
            }
            if (!SetSelect(hWnd, FD_READ))
               closesocket(s);
            break;
        case FD_CLOSE:
            if (WSAAsyncSelect(s, hWnd, 0, 0) == SOCKET_ERROR)
               AlertUser(hWnd, "WSAAsyncSelect Failed.\n");
            break;
        default:
            if (WSAGETSELECTERROR(lParam) != 0)
            {
               AlertUser(hWnd, "Socket Report Failure.");
               closesocket(s);
               break;
            }
            break;
      }
    break;
case WM_DESTROY:
    closesocket(s);
    WSACleanup();
    PostQuitMessage(0);
    break;
```

```
        default:
            return (DefWindowProc(hWnd, message, wParam, lParam));
    }
    return(NULL);
}

void AlertUser(HWND hWnd, char *message)
{
    MessageBox(hWnd, (LPSTR)message, "Warning", MB_ICONEXCLAMATION);
    return;
}

BOOL Client(HWND hWnd)
{
    memset(&dst_addr,'\0', sizeof (struct sockaddr_in));
    _fmemcpy((char  FAR *)&dst_addr.sin_addr,(char  FAR
*)hostnm.h_addr,hostnm.h_length);
    dst_addr.sin_family = hostnm.h_addrtype;
    dst_addr.sin_port = htons(USERPORT);
    if ((s = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
    {
        AlertUser(hWnd, "Socket Failed");
        return (FALSE);
    }
    if (!SetSelect(hWnd, FD_CONNECT))
        return (FALSE);
    connect(s, (struct sockaddr FAR *)&dst_addr, sizeof(dst_addr));
    return (TRUE);
}


BOOL ReceivePacket(HWND hWnd)
{
    HDC hDc;
    int length;
    int i1,i2,i3;
    char line1[255], line2[255], line3[255];

    count ++;
    if ((length = recv(s, lpBuffer, 1024, 0)) == SOCKET_ERROR)
        return (FALSE);
    if (length == 0)
        return (FALSE);
    if (hDc = GetDC(hWnd))
    {
        i1 = wsprintf((LPSTR)line1, "TCP Echo Client No.%d", count);
```

```
        i2 = wsprintf((LPSTR)line2, "Receive %d bytes",length);
        i3 = wsprintf((LPSTR)line3, "Those are:%c, %c, %c, %c, %c,
%c",buffer[0],buffer[1],buffer[2],buffer[100],buffer[1000],buffer[102
3]);
        TextOut(hDc, 10, 2, (LPSTR)line1, i1);
        TextOut(hDc, 10, 22, (LPSTR)line2, i2);
        TextOut(hDc, 10, 42, (LPSTR)line3, i3);
        ReleaseDC(hWnd, hDc);
    }
    return (TRUE);
}

BOOL SetSelect(HWND hWnd, long lEvent)
{
    if (WSAAsyncSelect(s, hWnd, UM_SOCK, lEvent) == SOCKET_ERROR)
    {
        AlertUser(hWnd, "WSAAsyncSelect Failure.");
        return (FALSE);
    }
    return (TRUE);
}

BOOL SendPacket(HWND hWnd, int len)
{
    int length;

    if ((length = send(s, lpBuffer, len, 0)) == SOCKET_ERROR)
        return (FALSE);
    else
    if (length != len)
    {
        AlertUser(hWnd, "Send Length NOT Match!");
        return (FALSE);
    }
    return (TRUE);
}
```

程序3:SERVER.C

```
#include <sys/types.h>
#include <sys/mntent.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#define USERPORT 10001
#define HOST_IP_ADDR "192.1.1.2"
```

```
main(int argc, char **argv)
{
    char buf[1024];
    struct sockaddr_in client;
    struct sockaddr_in server;

    int s;
    int ns;
    int namelen;
    int pktlen;

    if ((s=socket(AF_INET, SOCK_STREAM, 0))<0)
    {
        perror("Socket()");
        return;
    }
    bzero((char *)&server,sizeof(server));
    server.sin_family = AF_INET;
    server.sin_port = htons(USERPORT);
    server.sin_addr.s_addr = INADDR_ANY;
    if (bind(s, (struct sockaddr *)&server, sizeof(server))<0)
    {
        perror("Bind()");
        return;
    }
    if (listen(s,1)!=0)
    {
        perror("Listen()");
        return;
    }
    namelen = sizeof(client);
    if ((ns = accept(s, (struct sockaddr *)&client, &namelen)) ==-1)
    {
        perror("Accept()");
        return;
    }
    for (;;)
    {
        if ((pktlen = recv(ns,buf,1024,0))<0)
        {
            perror("Recv()");
            break;
        }
        else
        if (pktlen == 0)
        {
```

```
            printf("Recv():return FAILED,connection is shut down!\n");
            break;
        }
        else
            printf("Recv():return SUCCESS,packet length = %d\n",pktlen);
        sleep(1);
        if (send(ns,buf,pktlen,0)<0)
        {
            perror("Send()");
            break;
        }
        else
            printf("Send():return SUCCESS,packet length = %d\n",pktlen);
    }
    close(ns);
    close(s);
    printf("Server ended successfully\n");
}
```

3.4 另一个精巧的应用程序实例－wshout

在本节中，我们通过一个经过精心选择的例子，进一步讨论一下 Windows Sockets 编程技术。例如如何编制客户机或服务器程序，如何应用 TCP 有连接服务（流式套接口）或 UDP 无连接服务（数据报套接口），如何进行阻塞或非阻塞方式的套接口操作等等，这些都是经常碰到的问题。接下来要介绍的 wshout 程序，可以通过灵活地设置不同选项来达到上述应用情况的任意组合，从而基本覆盖了应用 Windows Sockets 编程所可能碰到的问题，具有很好的研究参考价值。

由于该程序思路清晰，结构精良，所以我们不打算很详细地剖析每一个语句，而只是简要介绍一下整个程序的逻辑结构，并在源程序中加入适当的注释。我们相信，任何具有基本 C 语言和 Windows 编程经验的读者，都能很轻松地读懂绝大部分内容。经过仔细咀嚼和推敲后，更能得到一些编写优质程序的灵感。

该程序在 FTP 公司的 PCTCP 支撑环境下调试通过，不过只要读者拥有任何符合 Windows Sockets 1.1 规范的实现，也能顺利执行该程序。

3.4.1 源程序目录

1. wshout.c wshout 主程序
2. wshout.h wshout 头文件
3. wshout.rc wshout 资源文件

4. ushout.c UDP 客户机程序
5. ulisten.c UDP 服务器程序
6. tshout.c TCP 客户机程序
7. tlisten.c TCP 服务器程序
8. errno.c 获取 WSAE*错误描述字符串程序
9. resolve.c 客户机/服务器启动程序

在编译本程序时，笔者用的是 BC3.1，只需做一个 PRJ 工程文件，将上述.c 文件及 winsock.lib 包括进来就行了。请注意 winsock.h 应在 include 目录或当前目录中，winsock.lib 可利用 winsock.dll 通过 implib 工具来建立。如果读者使用其他的编译器，可自行作相应的调整，在此不再赘述。

3.4.2 程序逻辑结构

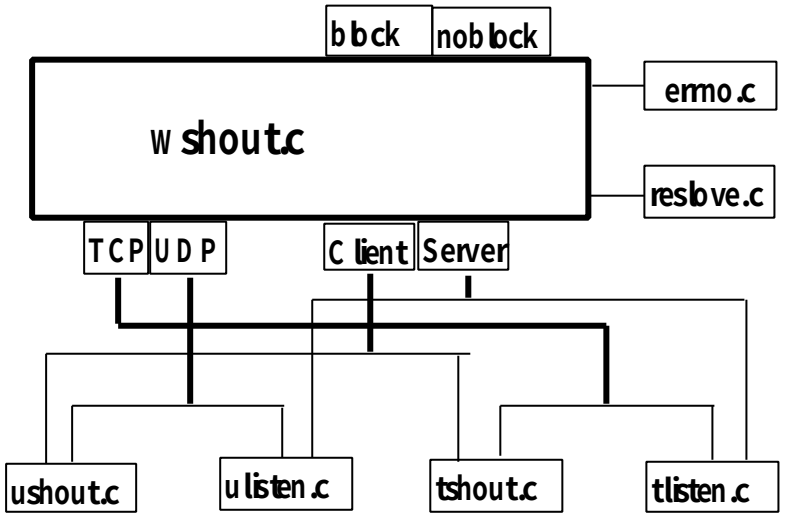

图 3-3 wshout 程序逻辑结构图

3.4.3 源程序清单及注释

3.4.3.1 wshout.c 清单

```
/*
```

```
 * 文件名: WSHOUT.C
 */

/* MSC Include files: */
#include <stdio.h>
#include <io.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

#include "wshout.h"

#define MAJOR_VERSION   1
#define MINOR_VERSION   2
#define WSA_MAKEWORD(x,y)   ((y) * 256 + (x)) /* HI:Minor, LO:Major */

HANDLE hInst;         /* 进程实例句柄 */
HWND   hOurWnd;       /* 主窗口句柄 */
HWND   hMainDlg;      /* 主对话框句柄 */

int ret;          /* 工作变量 */
char   prbuf[PRBUF_LEN]; /* 用于显示文本的工作缓冲区 */

SOCKET sd;        /* 用于监听连接的套接口描述字 */
long temporary_option   = 0L;   /* 缺省为阻塞模式 */
long blocking_option    = 0L;   /* 阻塞模式的全局标识 */
int run_cancelled = 0;    /* 指示何时按下了取消按钮 */
int len = 1024;           /* 一次写的字节数 */
BOOL   running = FALSE;  /* 程序的运行状态 */
const int iTCP = 1;       /* 指定为 TCP Shout */
const int iUDP = 2;       /* 指定为 UDP Shout */
int iProto = 1;       /* 缺省为 TCP Shout */
int iPortNo   = SOCK_SHOUT;
int temporary_protocol  = 1;    /* 在 Settings()中使用 */
int iShout  = 1;
int iListen = 2;
int iClientOrServer = 1; /* 缺省为 Shout（客户机）     */
int tClientOrServer = 1; /* 在 Settings()中使用 */
```

```
char HostModeBuf[20];/* 保存模式字符串 */
WORD VersionReqd;
LPWSADATA lpmyWSAData;

int PASCAL
WinMain (HANDLE hInstance,HANDLE hPrevInstance,LPSTR lpCmdLine,int
nCmdShow)
{

    HWND hWnd;
    MSG msg;
    BOOL InitApp(HANDLE);

    if (!hPrevInstance)
    if (!InitApp(hInstance))
        return (NULL);

    hInst = hInstance;

    hWnd = CreateWindow("MainMenu",
   "Windows Shout",
   WS_OVERLAPPEDWINDOW | WS_SYSMENU | WS_MINIMIZEBOX,
   CW_USEDEFAULT,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        NULL,
        NULL,
        hInstance,
        NULL);

    if (!hWnd)
   return (NULL);

    hOurWnd = hWnd;

    while (GetMessage(&msg, NULL, NULL, NULL)) {
   TranslateMessage(&msg); /* 翻译虚拟键码 */
```

```
    DispatchMessage(&msg);
    }

    return (msg.wParam);
}

BOOL InitApp(HANDLE hInstance)
{
    HANDLE hMemory;
    PWNDCLASS pWndClass;
    BOOL bSuccess;

    hMemory = LocalAlloc(LPTR, sizeof(WNDCLASS));
    pWndClass = (PWNDCLASS) LocalLock(hMemory);
    pWndClass->hCursor = LoadCursor(NULL, IDC_ARROW);
    pWndClass->hIcon = LoadIcon(hInstance, (LPSTR) "SHOUT");
    pWndClass->lpszMenuName = (LPSTR) "MainMenu";
    pWndClass->lpszClassName = (LPSTR) "MainMenu";
    pWndClass->hbrBackground = GetStockObject(WHITE_BRUSH);
    pWndClass->hInstance = hInstance;
    pWndClass->style = NULL;
    pWndClass->lpfnWndProc = ShoutWndProc;

    bSuccess = RegisterClass(pWndClass);

    LocalUnlock(hMemory);
    LocalFree(hMemory);
    return (bSuccess);
}

long FAR PASCAL ShoutWndProc(HWND hWnd, WORD message,WORD wParam, LONG
lParam)
{
    FARPROC lpDialogBoxProc;

    switch (message){
    case WM_CREATE:
```

```
        /* Put up the dialog box */
        lpDialogBoxProc = MakeProcInstance(DialogProc, hInst);
        DialogBox (hInst, (LPSTR) "MainDialog", hWnd, lpDialogBoxProc) ;
        FreeProcInstance(lpDialogBoxProc);
        PostMessage(hWnd, WM_DESTROY, 0, 0L);
        break;

    case WM_DESTROY:
        PostQuitMessage(0);
        break;

    default:
        return(DefWindowProc(hWnd, message, wParam, lParam));
    }
    return NULL;

}


BOOL FAR PASCAL DialogProc(HWND hOurDlg, WORD message, WORD wParam, LONG
lParam)
{
    FARPROC lpProcAbout;
    FARPROC lpProcSettings;
    long lret;
    WORD wMajorVersion, wMinorVersion;
    char hostnm[64];    /* 包含主机名的工作缓冲区 */

    switch (message) {
    case WM_INITDIALOG:
        /* 选择缺省主机 */
        SetDlgItemText(hOurDlg, IDD_HNAME, "");
        SendDlgItemMessage(hOurDlg,    /* 对话框句柄 */
        IDD_HNAME,        /* 向何处发送 msg */
        EM_SETSEL,        /* 选择字符 */
        NULL,             /* 附加信息 */
        MAKELONG(0, 0x7fff));       /* 全部内容 */
        SetFocus(GetDlgItem(hOurDlg, IDD_HNAME));
```

```
    /* 初始化 */
    hMainDlg = hOurDlg;  /* 保存自己的窗口句柄 */
    SetDlgItemText(hOurDlg, IDD_COHOST,"Shout to:");
    wMajorVersion = MAJOR_VERSION;
    wMinorVersion = MINOR_VERSION;
    VersionReqd=WSA_MAKEWORD(wMajorVersion, wMinorVersion);

    lpmyWSAData = (LPWSADATA)_calloc(1, sizeof(WSADATA));

    ret = WSAStartup(VersionReqd, lpmyWSAData);

    if (ret != 0){
   wshout_err (hOurDlg, WSAGetLastError(), "WSAStartup()");
    }

    return (TRUE);

case WM_CLOSE:
    PostMessage(hOurDlg, WM_COMMAND, IDM_EXIT, 0L);
    break;

case WM_SYSCOMMAND:
    SendMessage(hOurWnd, message, wParam, lParam);
    break;

case WM_COMMAND:
    switch (wParam) {
    case IDD_CONNECT:       /* 按下连接按钮 */
    case IDM_START:     /* 选择了 Start 菜单项 */
   run_cancelled = FALSE;
   /* 不能重入 */
   if (running){
       MessageBox(hOurWnd,"Shout is already running !",
      "Shout", MB_OK | MB_APPLMODAL | MB_ICONEXCLAMATION);
       return FALSE;
   }
   ClearBoxes(hOurDlg);
```

```
running = TRUE;

if (iClientOrServer == iShout) {
   /* 确保有主机名 */
   if (GetDlgItemText (hOurDlg, IDD_HNAME, hostnm, 80) < 2) {
       MessageBeep(0);
       SetDlgItemText(hOurDlg,
                  IDD_COMMENT,"No hostname specified");
       running = FALSE;
       break;
   }
  sd = ResolveAndConnectHost((char FAR *)hostnm,hOurDlg,iProto,
                              iPortNo);
   if (sd == SOCKET_ERROR) {/* 无法创建套接口 */
       running = FALSE;
       break;
   }
    }
else {
   sd = GetSocketAndBind(hOurDlg, iProto, iPortNo);
   if (sd == SOCKET_ERROR) {
    running = FALSE;
    break;
   }
    }

/* Set the I/O mode of the socket */
if (blocking_option) {
    lret = 1L;  /* 非阻塞模式 */
    ioctlsocket(sd, FIONBIO, (u_long FAR *) &lret);
}
else {
    lret = 0L; /* 阻塞模式 */
    ioctlsocket(sd, FIONBIO, (u_long FAR *) &lret);
}

if (iClientOrServer == iShout) {  /* SHOUT */
    /* 产生数据并写入套接口 */
```

```
    if (iProto == iTCP)
    lret = TWriteData(sd, hOurDlg, len);
    else /* UDP */
    lret = UWriteData(sd, hOurDlg, len);
}
else { /* LISTEN */
    if (iProto == iTCP)
    lret = TReadData(sd,hOurDlg, len);
    else /* UDP */
    lret = UReadData(sd,hOurDlg, len);
}

closesocket(sd);
running = FALSE;
break;

 case IDD_CANCEL:
if (running) {
    /* 停止 */
    ret = WSACancelBlockingCall();
    run_cancelled = TRUE;
    if (ret == SOCKET_ERROR) {
    /* WSANOTINITIALISED or WSAENETDOWN or WSAEINVAL */
    if (h_errno == WSAENETDOWN) {
        /* Watch out for hAcceptSock! */
        /* close what is left of the connection */
        closesocket(sd);
    }
    }
}
break;

 case IDM_EXIT:
ret = WSACleanup();
if (ret == SOCKET_ERROR && h_errno == WSAEINPROGRESS){
    MessageBox(hOurWnd,
   "Data transfer in progress.\nStop transfer first.",
   "WndProc()", MB_OK | MB_APPLMODAL|MB_ICONINFORMATION);
```

```
            break; /* 一个套接口正处于阻塞状态 */
        }

        _free((char NEAR *) lpmyWSAData);
        EndDialog(hOurDlg, TRUE) ; /* 退出 */
        break;

        case IDM_ABOUT:
        lpProcAbout = MakeProcInstance(About, hInst);
        DialogBox(hInst, "AboutBox", hOurDlg, lpProcAbout);
        FreeProcInstance(lpProcAbout);
        break;

        case IDM_SETTINGS:
        lpProcSettings = MakeProcInstance(Settings, hInst);
        DialogBox(hInst, "SettingsDialog", hOurDlg, lpProcSettings);
        FreeProcInstance(lpProcSettings);
        break;

        default:
        break;

        } /* switch (wParam) */
        break;
    } /* switch (message) */
    return FALSE;
}


/* 此函数处理 About 对话框 */
BOOL FAR PASCAL About(HWND hDlg, WORD message, WORD wParam, LONG lParam)
{
    char tempBuf[15];

    switch (message) {
    case WM_INITDIALOG:
   SetDlgItemText(hDlg, IDA_COPYRIGHT,(LPSTR)lpmyWSAData-
>szDescription);
```

```
    wsprintf(tempBuf, "%d.%2d\n",MAJOR_VERSION, MINOR_VERSION);
    SetDlgItemText(hDlg, IDA_APP_VERSION, (LPSTR) tempBuf);

    wsprintf(tempBuf, "%d.%2d\n",
             lpmyWSAData->wVersion%256,lpmyWSAData->wVersion/256);
    SetDlgItemText (hDlg, IDA_DLL_VERSION, (LPSTR) tempBuf);
    return (FALSE);

     case WM_COMMAND:
    if (wParam == IDOK
    || wParam == IDCANCEL) {
    EndDialog(hDlg, TRUE);
    return (TRUE);
    }
    break;
     }
     return (FALSE);
}

/* 此函数处理 Settings 对话框 */
BOOL FAR PASCAL Settings(HWND hDlg, WORD message, WORD wParam, LONG
lParam)
{
     int buffer_len = len;
     int port_no    = iPortNo;

     switch (message) {
     case WM_INITDIALOG:
    /* Select a default send() buffer length */
    SetDlgItemInt(hDlg, IDS_BUFFLEN, len, 0);

    /* Select a default port number */
    SetDlgItemInt(hDlg, IDS_PORTNO, iPortNo, 0);

    if (iClientOrServer == iShout)     /* 程序类型 */
        CheckThisProgBoxOn(hDlg, IDS_CLIENT);
    else
```

```
    CheckThisProgBoxOn(hDlg, IDS_SERVER);

if (iProto == iTCP)         /* 协议类型 */
    CheckThisProtoBoxOn(hDlg, IDS_TCP);
else
    CheckThisProtoBoxOn(hDlg, IDS_UDP);

if (!blocking_option)           /* 阻塞模式 */
    CheckThisBoxOn(hDlg, IDS_BLOCK);
else
    CheckThisBoxOn(hDlg, IDS_NOBLOCK);

SendDlgItemMessage(hDlg,    /* dialog handle  */
    IDS_PORTNO,          /* where to send msg */
    EM_SETSEL,       /* select characters */
    NULL,            /* additional info   */
    MAKELONG(0, 0x7fff));       /* entire contents   */

SendDlgItemMessage(hDlg,    /* dialog handle  */
    IDS_BUFFLEN,         /* where to send msg */
    EM_SETSEL,       /* select characters */
    NULL,            /* additional info   */
    MAKELONG(0, 0x7fff));       /* entire contents   */
SetFocus(GetDlgItem(hDlg, IDS_BUFFLEN));
return (TRUE);

 case WM_COMMAND:
switch (wParam){
case IDS_CLIENT:
    /* USer has set to Shout */
    CheckThisProgBoxOn(hDlg, IDS_CLIENT);
    tClientOrServer = iShout;
    SetDlgItemText(hMainDlg, IDD_COHOST,"Foreign host:");
    SetDlgItemText(hMainDlg, IDD_HNAME,"");
    break;
case IDS_SERVER:
    /* USer has set to Listen */
    CheckThisProgBoxOn(hDlg, IDS_SERVER);
```

```
        tClientOrServer = iListen;
        SetDlgItemText(hMainDlg, IDD_COHOST,"Listening to:");
        SetDlgItemText(hMainDlg, IDD_HNAME,"[Hit 'Start']");
        break;
    case IDS_TCP:
        /* USer has set to TCP */
        CheckThisProtoBoxOn(hDlg, IDS_TCP);
        temporary_protocol = iTCP;
        break;
    case IDS_UDP:
        /* USer has set to UDP */
        CheckThisProtoBoxOn(hDlg, IDS_UDP);
        temporary_protocol = iUDP;
        break;
    case IDS_BLOCK:
        /* User has set to blocking mode */
        CheckThisBoxOn(hDlg, IDS_BLOCK);
        temporary_option = 0L;
        break;
    case IDS_NOBLOCK:
        /* User has set to nonblocking mode */
        CheckThisBoxOn(hDlg, IDS_NOBLOCK);
        temporary_option = 1L;
        break;
    case IDOK:
        /* 用户已完成对设置的修改 */
        buffer_len = GetDlgItemInt(hDlg, IDS_BUFFLEN, NULL, 0);
        if (buffer_len == 0 || buffer_len > 8192) {
       MessageBox(hOurWnd, "Buffer length must be between 1 and 8K",
              "Settings()",
              MB_OK | MB_APPLMODAL | MB_ICONSTOP);
       return (FALSE);
        }

        port_no = GetDlgItemInt(hDlg, IDS_PORTNO, NULL, 0);
        if (port_no == 0) {
       MessageBox(hDlg, "Port number must be between 0 and 65,535",
              "Settings()",
```

```
                MB_OK | MB_APPLMODAL | MB_ICONSTOP);
            return (FALSE);
            }
            len     = buffer_len;
            iPortNo = port_no;
            blocking_option = temporary_option;
            iProto    = temporary_protocol;
            iClientOrServer = tClientOrServer;

        case IDCANCEL:
            /* 用户不想改变设置 */
            EndDialog(hDlg, TRUE);
            return (TRUE);

        default:
            break;
        }

        default:
        break;
        }
        return (FALSE);
}

void
CheckThisBoxOn(HWND hDlg, int ButtonID)
{
        switch (ButtonID) {
        case IDS_BLOCK:
        CheckDlgButton(hDlg, IDS_BLOCK, 1);
        CheckDlgButton(hDlg, IDS_NOBLOCK, 0);
        break;
        case IDS_NOBLOCK:
        CheckDlgButton(hDlg, IDS_BLOCK, 0);
        CheckDlgButton(hDlg, IDS_NOBLOCK, 1);
        break;
        default:
        break;
```

```
    }
    return;
}

void
CheckThisProtoBoxOn(HWND hDlg, int ButtonID)
{
    switch (ButtonID) {
    case IDS_TCP:
   CheckDlgButton(hDlg, IDS_TCP, 1);
   CheckDlgButton(hDlg, IDS_UDP, 0);
   break;
    case IDS_UDP:
   CheckDlgButton(hDlg, IDS_TCP, 0);
   CheckDlgButton(hDlg, IDS_UDP, 1);
   break;
    default:
   break;
    }
    return;
}

void
CheckThisProgBoxOn(HWND hDlg, int ButtonID)
{
    switch (ButtonID) {
    case IDS_CLIENT: /* Shout */
   CheckDlgButton(hDlg, IDS_CLIENT, 1);
   CheckDlgButton(hDlg, IDS_SERVER, 0);
   break;
    case IDS_SERVER: /* Listen */
   CheckDlgButton(hDlg, IDS_CLIENT, 0);
   CheckDlgButton(hDlg, IDS_SERVER, 1);
   break;
    default:
   break;
    }
    return;
```

```
}

/* 以下就是我们如何处理“模拟阻塞”－本函数检查消息队列，如果发现需要
处理的消息，就返回一个正的值。*/
int
ShoutBlockingHook (void)
{
  MSG msg;    /* lets us pull messages via PeekMessage */
  int ret = PeekMessage(&msg, NULL, 0, 0, PM_REMOVE);

  if (ret) {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }
  return ret;
}

char *
_calloc (nelem, elsize)
 unsigned nelem, elsize;
{
    HANDLE hMem;
    PSTR ptr;
    unsigned size = nelem * elsize;

   if ((hMem = LocalAlloc(LPTR, size)) == NULL)
      return (char *) 0;
   if ((ptr = LocalLock(hMem)) == NULL) {
      LocalFree(hMem);
      return (char *) 0;
   }
   return (char *) ptr;
}

void
_free (void *cP)
{
   (void) LocalFree(LocalHandle((WORD) cP));
```

```
}

void
ClearBoxes(HWND hOurDlg)
{
    wsprintf(prbuf,"         \n");
    SetDlgItemText(hOurDlg, IDD_WRITE, (LPSTR) prbuf);
    SetDlgItemText(hOurDlg, IDD_SENT, (LPSTR) prbuf);
    SetDlgItemText(hOurDlg, IDD_TIME, (LPSTR) prbuf);
    SetDlgItemText(hOurDlg, IDD_WRITES,(LPSTR) prbuf);
    SetDlgItemText(hOurDlg, IDD_BYTES, (LPSTR) prbuf);
    SetDlgItemText(hOurDlg, IDD_BITS,  (LPSTR) prbuf);

    return;
}

/*
 * wshout_err()函数
 * 描述:
 *
 * 通过错误代码获取相应的错误描述文本，与用户提供的错误前缀合
 * 并，并显示在对话框中。
 */
void wshout_err (HWND hOurDlg, /* 对话框的窗口句柄 */
        int wsa_err,           /* WinSock 错误代码 */
        char far *err_prefix)  /* 错误前缀字符串 */
{
   char errbuf[PRBUF_LEN];   /* 错误描述字符串缓冲区 */

       /* 获取错误描述字符串 */
   WSAsperror(hInst, wsa_err, (LPSTR)errbuf, PRBUF_LEN);

   /* 合并错误描述字符串与用户错误前缀字符串 */
   wsprintf((LPSTR)prbuf, "%s:%s", (LPSTR) err_prefix, (LPSTR)errbuf);

   /* 在对话框中显示错误文本 */
   SetDlgItemText(hOurDlg, IDD_COMMENT, (LPSTR) prbuf);
```

```
}  /* end wshout_err() */


/* eof */
```

3.4.3.2 wshout.h 清单

```
/*
 * 文件名: WSHOUT.H
 */

#ifndef _WSHOUT_INC_
#define _WSHOUT_INC_

/* Windows 3.0 头文件 */
#include <windows.h>
#define _INC_WINDOWS
#include <winsock.h>

#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */


/* WSHOUT.C 中定义的全局变量 */
extern HANDLE hInst;        /* Instance handle */
extern HWND   hOurWnd;      /* Main Window Handle */
extern int ret;           /* work variable */

#define PRBUF_LEN 50
extern char   prbuf[PRBUF_LEN]; /* work buffer */

/* 菜单 IDs   */
#define IDM_START 101
#define IDM_ABOUT 102
#define IDM_STOP  103
#define IDM_EXIT  104
#define IDM_SETTINGS 105
```

```
/* 对话框控制 IDs */
#define IDD_HNAME 200
#define IDD_CONNECT  IDOK
#define IDD_CANCEL   IDCANCEL
#define IDD_WRITES   208
#define IDD_BYTES 210
#define IDD_BITS  212
#define IDD_HELP  214
#define IDD_SENT  230
#define IDD_WRITE 232
#define IDD_TIME  234
#define IDD_COMMENT  236
#define IDD_COHOST   240

/* “Settings”对话框控制 IDs   */
#define IDS_BUFFLEN  300
#define IDS_PORTNO   301
#define IDS_BLOCK 302
#define IDS_NOBLOCK  304
#define IDS_TCP      306
#define IDS_UDP      308
#define IDS_CLIENT   310
#define IDS_SERVER   312
#define IDS_DEFAULT  314

/* “About”对话框控制 IDs   */
#define IDA_COPYRIGHT    400
#define IDA_APP_VERSION  402
#define IDA_DLL_VERSION  404

/* 程序控制 IDs   */
#define    WM_SELECT     WM_USER+16

/* 全局变量*/
#define    SOCK_DISCARD  9   /* use the UDP ttytst source port for test
*/
#define SOCK_SHOUT   32766  /* TCP port used for SHOUT & LISTEN
```

```
*/
#define BUF_SIZE  8192
#define WRITE_TIMER  1

/* 函数原型 */
int PASCAL WinMain(HANDLE, HANDLE, LPSTR, int);
long FAR PASCAL ShoutWndProc(HWND, WORD, WORD, LONG);
BOOL FAR PASCAL About(HWND, WORD, WORD, LONG);
BOOL FAR PASCAL DialogProc(HWND, WORD, WORD, LONG);
BOOL FAR PASCAL Settings(HWND, WORD, WORD, LONG);
BOOL InitApp(HANDLE);
void CheckThisBoxOn(HWND, int);
void CheckThisProtoBoxOn(HWND, int);
void CheckThisProgBoxOn(HWND, int);
void ClearBoxes(HWND);
SOCKET ResolveAndConnectHost(LPSTR, HWND, int, int);
SOCKET GetSocketAndBind(HWND, int, int);
long UWriteData(SOCKET, HWND, int);
long UReadData(SOCKET, HWND, int);
long TWriteData(SOCKET, HWND, int);
long TReadData(SOCKET, HWND, int);
int ShoutBlockingHook (void);
int PASCAL FAR WSAsperror (HANDLE, int, char far *, int);
void wshout_err (HWND, int, char far *);

#define bcopy(a,b,c) _fmemcpy(b,a,c)
char * _calloc (unsigned, unsigned);
void _free (void *);

#ifdef _cplusplus
}
#endif /* __cplusplus */

#endif /* ifndef _WSHOUT_INC_ */

/* eof */
```

3.4.3.3 wshout.rc 清单

```
/*
 * 文件名: WSHOUT.RC
 */
#include <windows.h>
#include <winsock.h>
#include "wshout.h"

MainMenu MENU
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&Start", IDM_START
        MENUITEM "Sto&p", IDM_STOP
        MENUITEM SEPARATOR
        MENUITEM "E&xit", IDM_EXIT
    END

    POPUP "&Options"
    BEGIN
        MENUITEM "&Settings ...", IDM_SETTINGS
        MENUITEM SEPARATOR
        MENUITEM "&About Shout...", IDM_ABOUT
    END

END

ABOUTBOX DIALOG 22, 17, 144, 102
CAPTION "About Shout for Windows"
STYLE DS_MODALFRAME | WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU
BEGIN
    CTEXT "Windows Shout", -1, 29, 5, 85, 8
    CTEXT "Version", -1, 46, 13, 33, 8, SS_CENTER | WS_GROUP
    CTEXT "WINSOCK.DLL \n FTP Software, Inc. \nCopyright 1993",
IDA_COPYRIGHT, 38, 40, 68, 25, SS_CENTER | WS_GROUP
    CTEXT "Version", -1, 46, 67, 33, 8, SS_CENTER | WS_GROUP
    CTEXT "num", IDA_DLL_VERSION, 79, 67, 18, 8, SS_CENTER | WS_GROUP
```

```
    CONTROL "OK", 1, "BUTTON", BS_DEFPUSHBUTTON | WS_GROUP | WS_TABSTOP,
56, 82, 32, 14
    ICON "SHOUT", -1, 11, 8, 16, 16
    CONTROL "num", IDA_APP_VERSION, "STATIC", SS_LEFT | WS_GROUP, 79, 13,
18, 8
    CONTROL "using", -1, "STATIC", SS_CENTER | WS_GROUP, 55, 26, 30, 8
END

SettingsDialog DIALOG 9, 16, 172, 117
CAPTION "Settings"
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
BEGIN
    CONTROL "   send/recv \nBuffer &length", -1, "STATIC", SS_LEFT |
WS_GROUP, 84, 8, 48, 20
    CONTROL "&Port number", -1, "STATIC", SS_LEFT | WS_GROUP, 84, 31, 48,
10
    CONTROL "&Blocking", IDS_BLOCK, "BUTTON", BS_AUTOCHECKBOX |
WS_TABSTOP, 100, 61, 56, 12
    CONTROL "&TCP", IDS_TCP, "BUTTON", BS_AUTOCHECKBOX | WS_TABSTOP, 20,
60, 41, 12
    CONTROL "&Client", IDS_CLIENT, "BUTTON", BS_AUTOCHECKBOX |
WS_TABSTOP, 19, 15, 35, 12
    CONTROL "&Server", IDS_SERVER, "BUTTON", BS_AUTOCHECKBOX |
WS_TABSTOP, 19, 26, 35, 12
    CONTROL "&UDP", IDS_UDP, "BUTTON", BS_AUTOCHECKBOX | WS_TABSTOP, 20,
72, 41, 12
    CONTROL "&Nonblocking", IDS_NOBLOCK, "BUTTON", BS_AUTOCHECKBOX |
WS_TABSTOP, 100, 73, 56, 12
    CONTROL "O.K.", IDOK, "BUTTON", BS_PUSHBUTTON | WS_TABSTOP, 40, 95,
37, 14
    CONTROL "Cancel", IDCANCEL, "BUTTON", B S_PUSHBUTTON | WS_TABSTOP, 90,
95, 37, 14
    CONTROL "", IDS_BUFFLEN, "EDIT", ES_CENTER | WS_BORDER | WS_TABSTOP,
130, 11, 36, 12
    CONTROL "", IDS_PORTNO, "EDIT", ES_CENTER | WS_BORDER | WS_TABSTOP,
130, 29, 36, 12
    CONTROL "Protocol", 237, "button", BS_GROUPBOX, 6, 49, 70, 38
    CONTROL "I/O Mode", 239, "button", BS_GROUPBOX, 90, 49, 70, 38
```

```
    CONTROL "Program Mode", 241, "button", BS_GROUPBOX, 6, 7, 70, 34
END

MainDialog DIALOG 17, 32, 163, 135
CAPTION "Windows Shout"
MENU MainMenu
STYLE DS_ABSALIGN | WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU |
WS_MINIMIZEBOX
BEGIN
    CONTROL "", IDD_HNAME, "EDIT", ES_CENTER | WS_BORDER | WS_GROUP |
WS_TABSTOP, 62, 9, 93, 12
    CONTROL "", IDD_WRITE, "STATIC", SS_CENTER | SS_NOPREFIX | WS_BORDER,
7, 95, 45, 11
    CONTROL "", IDD_SENT, "STATIC", SS_CENTER | WS_BORDER, 59, 95, 45,
11
    CONTROL "", IDD_TIME, "STATIC", SS_CENTER | WS_BORDER, 111, 95, 45,
11
    CONTROL "", IDD_WRITES, "STATIC", SS_CENTER | WS_BORDER, 7, 120, 45,
11
    CONTROL "", IDD_BYTES, "STATIC", SS_CENTER | WS_BORDER, 59, 120, 45,
11
    CONTROL "", IDD_BITS, "STATIC", SS_CENTER | WS_BORDER, 111, 120, 45,
11
    CONTROL "writes[reads]", 105, "STATIC", SS_CENTER | WS_GROUP, 3, 85,
52, 9
    CONTROL "writes[reads]/s", 105, "STATIC", SS_CENTER | WS_GROUP, 3,
111, 55, 9
    CONTROL "bytes", 105, "STATIC", SS_CENTER | WS_GROUP, 61, 85, 42, 9
    CONTROL "bytes/sec", 105, "STATIC", SS_CENTER | WS_GROUP, 61, 111,
42, 9
    CONTROL "time (sec)", 105, "STATIC", SS_CENTER | WS_GROUP, 111, 85,
45, 9
    CONTROL "bits/sec", 105, "STATIC", SS_CENTER | WS_GROUP, 113, 111,
42, 9
    CONTROL "Host", IDD_COHOST, "STATIC", SS_LEFT, 7, 10, 52, 10
    CONTROL "", IDD_COMMENT, "STATIC", SS_CENTER | WS_BORDER | WS_GROUP,
9, 68, 146, 11
    CONTROL "&Start", IDOK, "BUTTON", BS_PUSHBUTTON | WS_TABSTOP, 6, 32,
```

```
32, 20
    CONTROL "Sto&p", IDCANCEL, "BUTTON", BS_PUSHBUTTON | WS_TABSTOP, 65,
32, 32, 20
    CONTROL "E&xit", IDM_EXIT, "BUTTON", BS_PUSHBUTTON | WS_TABSTOP, 125,
32, 32, 20
    CONTROL "", -1, "static", SS_BLACKFRAME, 0, 60, 163, 1
END

SHOUT ICON wshout.ico


/*
 *  错误描述字符串表
 *  用于 WSAsperror()函数
 */
STRINGTABLE
BEGIN
  WSABASEERR,         "[0] No Error"
  WSAEINTR,           "[10004] Interrupted system call"
  WSAEBADF,           "[10009] Bad file number"
  WSAEACCES,          "[10013] Permission denied"
  WSAEFAULT,          "[10014] Bad address"
  WSAEINVAL,          "[10022] Invalid argument"
  WSAEMFILE,          "[10024] Too many open files"
  WSAEWOULDBLOCK,     "[10035] Operation would block"
  WSAEINPROGRESS,     "[10036] Operation now in progress"
  WSAEALREADY,        "[10037] Operation already in progress"
  WSAENOTSOCK,        "[10038] Socket operation on non-socket"
  WSAEDESTADDRREQ,    "[10039] Destination address required"
  WSAEMSGSIZE,        "[10040] Message too long"
  WSAEPROTOTYPE,      "[10041] Protocol wrong type for socket"
  WSAENOPROTOOPT,     "[10042] Bad protocol option"
  WSAEPROTONOSUPPORT, "[10043] Protocol not supported"
  WSAESOCKTNOSUPPORT, "[10044] Socket type not supported"
  WSAEOPNOTSUPP,      "[10045] Operation not supported on socket"
  WSAEPFNOSUPPORT,    "[10046] Protocol family not supported"
 WSAEAFNOSUPPORT,   "[10047] Address family not supported by protocol
family"
```

```
  WSAEADDRINUSE,      "[10048] Address already in use"
  WSAEADDRNOTAVAIL,   "[10049] Can't assign requested address"
  WSAENETDOWN,        "[10050] Network is down"
  WSAENETUNREACH,     "[10051] Network is unreachable"
  WSAENETRESET,       "[10052] Net dropped connection or reset"
  WSAECONNABORTED,    "[10053] Software caused connection abort"
  WSAECONNRESET,      "[10054] Connection reset by peer"
  WSAENOBUFS,         "[10055] No buffer space available"
  WSAEISCONN,         "[10056] Socket is already connected"
  WSAENOTCONN,        "[10057] Socket is not connected"
  WSAESHUTDOWN,       "[10058] Can't send after socket shutdown"
  WSAETOOMANYREFS,    "[10059] Too many references, can't splice"
  WSAETIMEDOUT,       "[10060] Connection timed out"
  WSAECONNREFUSED,    "[10061] Connection refused"
  WSAELOOP,           "[10062] Too many levels of symbolic links"
  WSAENAMETOOLONG,    "[10063] File name too long"
  WSAEHOSTDOWN,       "[10064] Host is down"
  WSAEHOSTUNREACH,    "[10065] No Route to Host"
  WSAENOTEMPTY,       "[10066] Directory not empty"
  WSAEPROCLIM,        "[10067] Too many processes"
  WSAEUSERS,          "[10068] Too many users"
  WSAEDQUOT,          "[10069] Disc Quota Exceeded"
  WSAESTALE,          "[10070] Stale NFS file handle"
  WSAEREMOTE,         "[10071] Too many levels of remote in path"
  WSASYSNOTREADY,     "[10091] Network SubSystem is unavailable"
  WSAVERNOTSUPPORTED, "[10092] WINSOCK DLL Version out of range"
 WSANOTINITIALISED,  "[10093] Successful WSASTARTUP not yet performed"
  WSAHOST_NOT_FOUND,  "[11001] Host not found"
  WSATRY_AGAIN,       "[11002] Non-Authoritative Host not found"
 WSANO_RECOVERY,    "[11003] Non-Recoverable errors: FORMERR, REFUSED,
NOTIMP"
  WSANO_DATA,        "[11004] Valid name, no data record of requested
type"
END

/* eof */
```

3.4.3.4 ushout.c 清单

```
/*
 * 文件名: USHOUT.C
 */

#include "wshout.h"

/* MSC Include files: */
#include <stdio.h>
#include <io.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

/* Returns the number of bytes written */
long UWriteData(SOCKET hSock, HWND hOurDlg, int send_len)
{
    int counter;
   static int DataBuffer[BUF_SIZE];  /* Buffer to hold generated data*/
    static  char ReplyBuffer[512];    /* Buffer to hold any reply rcvd
*/
    long bytes_sent = 0L;         /* Counter of bytes on connection
   */
    long total_len = 1024L*1024L;     /* Total # of bytes to generate
   */
    time_t start, end;            /* variables to hold read timing */
    long total_time = 0L;         /* variable to hold delta t  */
    long write_count = 0L;        /* number of times    */
    long tmp = 0L;             /* holds count for bytes written */
    long ltemp = 0L;
    int i_temp;
    extern int run_cancelled;

    struct sockaddr_in dest;  /* Destination machine address structure
*/

    /* What makes shout unique is that it generates data*
```

```
 * in memory (as opposed to accessing the disk). *
 * This tests the 'raw' speed of the TCP connection *
 * as the rate-limiting access time is eliminated.  *
 * First, generate the data and place it into an *
 * array, data_buffer:             */

for (counter = 0; counter < BUF_SIZE; counter++)
DataBuffer[counter] = counter;

/* Write data on the descriptor like a banshee,
 * careful to time the writes and count data
 * transmitted:
 */

SetDlgItemText(hOurDlg, IDD_COMMENT, "Sending UDP Data ...");
time( &start );
while (bytes_sent < total_len){/* while still bytes to send */
do {
    ;
} while (ShoutBlockingHook()); /* Dispatch messages if any */

if (run_cancelled) {
    WSASetLastError(WSAEINTR);
    break; /* Non-blocking mode was cancelled */
}

tmp = send(hSock, (char FAR *) &DataBuffer, send_len, 0);
if (tmp == SOCKET_ERROR) {
    if (h_errno == WSAEWOULDBLOCK) /* if no data, read again */
   continue;
    else {
   wshout_err (hOurDlg, WSAGetLastError(), "send()");
    }

    /* Calc. time elapsed & stats about any data sent */
    time(&end);
    if (total_time = (long) difftime(end, start)) {
   /* Print the statistics gathered    */
```

```
    wsprintf((LPSTR)prbuf,"%ld\n",write_count);
    SetDlgItemText(hOurDlg, IDD_WRITE, (LPSTR) prbuf);

    wsprintf((LPSTR)prbuf,"%ld\n",bytes_sent);
    SetDlgItemText(hOurDlg, IDD_SENT, (LPSTR) prbuf);

    wsprintf((LPSTR)prbuf,"%ld\n",total_time);
    SetDlgItemText(hOurDlg, IDD_TIME, (LPSTR) prbuf);

    ltemp = write_count/total_time;
    wsprintf((LPSTR)prbuf,"%ld\n", ltemp);
    SetDlgItemText(hOurDlg, IDD_WRITES,(LPSTR) prbuf);

    ltemp = bytes_sent/total_time;
    wsprintf((LPSTR)prbuf,"%ld\n", ltemp);
    SetDlgItemText(hOurDlg, IDD_BYTES, (LPSTR) prbuf);

    ltemp = 8 * (bytes_sent/total_time);
    wsprintf((LPSTR)prbuf,"%ld\n", ltemp);
    SetDlgItemText(hOurDlg, IDD_BITS,  (LPSTR) prbuf);

    /* exit from the while loop */
    break;
     } /* end if (total_time) */
     write_count++;   /* incr. counter of times written  */
     bytes_sent += tmp;    /* # of bytes placed on connection */
     wsprintf((LPSTR)prbuf,"%ld\n",bytes_sent);
     SetDlgItemText(hOurDlg, IDD_SENT, (LPSTR) prbuf);
} /* end if (tmp == -1) */

write_count++;    /* incr. counter of times written  */
bytes_sent += tmp;    /* # of bytes placed on connection */
wsprintf((LPSTR)prbuf,"%ld\n",write_count);
SetDlgItemText(hOurDlg, IDD_WRITE, (LPSTR) prbuf);

wsprintf((LPSTR)prbuf,"%ld\n",bytes_sent);
SetDlgItemText(hOurDlg, IDD_SENT, (LPSTR) prbuf);
 } /* end while */
```

```
    /* Look for a reply ... NOTE: most hosts won't give
     * a 'reply', done to illustrate communication between
     * sockets. Our ulisten example will give a reply though.
     */

    SetDlgItemText(hOurDlg, IDD_COMMENT, "Waiting for reply from
server..\n");
    while (1) {
    tmp = sizeof(dest);
    i_temp = recvfrom(hSock,(char FAR *)
&ReplyBuffer,sizeof(ReplyBuffer),
                    0, (struct sockaddr *) &dest, (int FAR *) &tmp);
    if (i_temp == SOCKET_ERROR) {
        if (h_errno == WSAEWOULDBLOCK) /* if no data, read again */
       continue;
        else {
       /* any error besides these. just punt */
       wshout_err (hOurDlg, WSAGetLastError(), "recvfrom()");
        }
        break;
    } /* end if (i_temp == SOCKET_ERROR) */
    /* else got a reply ...*/
    wsprintf((LPSTR)prbuf, "Server: %s\n", (LPSTR) ReplyBuffer);
    SetDlgItemText(hOurDlg, IDD_COMMENT, prbuf);
    break;
    } /* end while(1) */

    /* All done */
    return bytes_sent;
}

/* eof */
```

3.4.3.5 ulisten.c 清单

```
/*
 * 文件名: ULISTEN.C
```

```
 */

#include "wshout.h"


/* MSC Include files: */
#include <stdio.h>
#include <io.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

/* Returns the number of bytes written */
long UReadData(SOCKET hSock, HWND hOurDlg, int read_len)
{
    static char   ReadBuf[BUF_SIZE];
    static char   SendBuf[512];
    struct sockaddr_in local;   /* Local machine address structure */
    int i;        /* General purpose return code     */
    long total_time = 0L;       /* variable to hold delta t    */
    int    tmp, len   = 0;
    int    num_reads  = 0;
    long bytes_read = 0L;
    long last_time, now, timeout = 15L;
    long ltemp;
    extern int run_cancelled;

    BOOL bTemp = TRUE;

   SetDlgItemText(hOurDlg, IDD_COMMENT, "Awaiting the UDP Data ...");
    SetDlgItemText(hOurDlg, IDD_HNAME, "                        ");

    time(&now); time(&last_time);
    while (last_time + timeout > now) {
   time(&now);
   tmp = sizeof(local);
   do {
       ;
```

```
    } while (ShoutBlockingHook()); /* Dispatch messages while available
*/

    if (run_cancelled) {
        WSASetLastError(WSAEINTR);
        break; /* Non-blocking mode was cancelled */
    }

    len = recvfrom(hSock, ReadBuf, read_len, 0,
                   (struct sockaddr *) &local, &tmp);
    if (len == SOCKET_ERROR) {
        if (h_errno == WSAEWOULDBLOCK) {/* if no data, read again */
       continue;
        } /* end: if (errno == WSAEWOULDBLOCK) */
        else {
       if (bytes_read) {
            wshout_err (hOurDlg, WSAGetLastError(), "recvfrom()");
       }
        } /* end else */
        break;
     } /* end: if (len == SOCKET_ERROR) */

     if (bTemp) { /* To update our main display once */
         /* Do not use wsprintf() or you will add an extra char */
         _fmemcpy(prbuf, inet_ntoa(local.sin_addr), 4*sizeof(u_long));
         SetDlgItemText(hOurDlg, IDD_HNAME, (LPSTR) prbuf);
         SetDlgItemText(hOurDlg, IDD_COMMENT, "Reading UDP Data ...");
         bTemp = FALSE;
     }

     num_reads++;
     if (len != SOCKET_ERROR)
         bytes_read += len;
     /* Print the statistics gathered    */
     wsprintf((LPSTR)prbuf,"%d\n",num_reads);
     SetDlgItemText(hOurDlg, IDD_WRITE, (LPSTR) prbuf);

     wsprintf((LPSTR)prbuf,"%ld\n",bytes_read);
```

```
SetDlgItemText(hOurDlg, IDD_SENT, (LPSTR) prbuf);

time(&last_time);
 } /* end: while */
 total_time = timeout;

 wsprintf((LPSTR)prbuf,"%ld\n",total_time);
 SetDlgItemText(hOurDlg, IDD_TIME, (LPSTR) prbuf);

 ltemp = num_reads/total_time;
 wsprintf((LPSTR)prbuf,"%ld\n", ltemp);
 SetDlgItemText(hOurDlg, IDD_WRITES,(LPSTR) prbuf);

 ltemp = bytes_read/total_time;
 wsprintf((LPSTR)prbuf,"%ld\n", ltemp);
 SetDlgItemText(hOurDlg, IDD_BYTES, (LPSTR) prbuf);

 ltemp = 8 * (bytes_read/total_time);
 wsprintf((LPSTR)prbuf,"%ld\n", ltemp);
 SetDlgItemText(hOurDlg, IDD_BITS,  (LPSTR) prbuf);

 if (bytes_read) {
SetDlgItemText(hOurDlg, IDD_COMMENT, "...UDP Listen Done\n");
 } /* end: if (bytes_read) */
 else {
wsprintf((LPSTR)prbuf, "Timed out. No data received.\n");
SetDlgItemText(hOurDlg, IDD_COMMENT, prbuf);
goto come_here;
 } /* end: else */

 /* send reply to 'client' */
 wsprintf((LPSTR)prbuf,
"Replied to %s\n", inet_ntoa(local.sin_addr));
 SetDlgItemText(hOurDlg, IDD_COMMENT, prbuf);

 for (i=0; i<10; ++i) {
sprintf(SendBuf, "Rec'd %ld bytes.\n", bytes_read);
if(len = sendto(hSock, SendBuf, sizeof(SendBuf), 0,
```

```
                    (struct sockaddr FAR *)&local,sizeof(local)))
         {
         if (len == SOCKET_ERROR) { /* if could not send bec. */
         if (h_errno == WSAEWOULDBLOCK)
             continue;
         wshout_err (hOurDlg, WSAGetLastError(), "sendto()");
         break;
         } /* end: if (len == -1) */
     } /* end: if (len = sendto()) */
     } /* end for */
     come_here:
     return (bytes_read);
 }

/* eof */
```

3.4.3.6 tshout.c 清单

```
/*
 * 文件名: TSHOUT.C
 */

#include "wshout.h"

/* MSC Include files: */
#include <stdio.h>
#include <io.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>


/* Returns the number of bytes written */
long TWriteData(SOCKET hSock, HWND hOurDlg, int send_len)
{
    int counter;
    static int DataBuffer[BUF_SIZE];     /* Buffer to hold generated
data  */
```

```
    long   total_len = 1024L*1024L;  /* Total # of bytes to generate   */
    long   bytes_sent = 0L;    /* Counter of bytes on connection */
    int    tmp = 0;         /* holds count for bytes written  */
    long   write_count = 0L;   /* number of times         */
    time_t start, end;         /* variables to hold read timing  */
    long   total_time = 0L;    /* variable to hold delta t       */
    long ltemp = 0L;
    extern int run_cancelled;

    /* What makes shout unique is that it generates data*
     * in memory (as opposed to accessing the disk). *
     * This tests the 'raw' speed of the TCP connection *
     * as the rate-limiting access time is eliminated.  *
     * First, generate the data and place it into an *
     * array, data_buffer:             */

    for (counter = 0; counter < BUF_SIZE; counter++)
   DataBuffer[counter] = counter;

    /* Write data on the descriptor like a banshee,
     * careful to time the writes and count data
     * transmitted:
     */

    SetDlgItemText(hOurDlg, IDD_COMMENT, "...Sending TCP Data");
    time(&start);
   while ( bytes_sent < total_len) {   /* while still bytes to send...
*/
   do {
       ;
   } while (ShoutBlockingHook()); /* Dispatch messages if any */

   if (run_cancelled) {
       WSASetLastError(WSAEINTR);
       break; /* Non-blocking mode was cancelled */
   }

   tmp = send(hSock, (char FAR*) &DataBuffer, send_len, 0);
```

```
if (tmp == SOCKET_ERROR) {
    if (h_errno == WSAEWOULDBLOCK)
   continue;
    else {
   wshout_err (hOurDlg, WSAGetLastError(), "send()");
    }

    /* Calc. time elapsed & stats about any data sent */
    time(&end);
    /* exit from the while loop */
    break;

} /* end if (tmp == -1) */
write_count++;        /* incr. counter of times written */
bytes_sent += tmp;  /* total # of bytes placed on connection*/
wsprintf(prbuf,"%ld\n",write_count);
SetDlgItemText(hOurDlg, IDD_WRITE, (LPSTR) prbuf);

wsprintf(prbuf,"%ld\n",bytes_sent);
SetDlgItemText(hOurDlg, IDD_SENT, (LPSTR) prbuf);
 } /* end while */
 time(&end);

 if (total_time = (long) difftime(end, start)) {
/* Print the statistics gathered    */
wsprintf((LPSTR)prbuf,"%ld\n",total_time);
SetDlgItemText(hOurDlg, IDD_TIME, (LPSTR) prbuf);

ltemp = write_count/total_time;
wsprintf((LPSTR)prbuf,"%ld\n", ltemp);
SetDlgItemText(hOurDlg, IDD_WRITES,(LPSTR) prbuf);

ltemp = bytes_sent/total_time;
wsprintf((LPSTR)prbuf,"%ld\n", ltemp);
SetDlgItemText(hOurDlg, IDD_BYTES, (LPSTR) prbuf);

ltemp = 8 * (bytes_sent/total_time);
wsprintf((LPSTR)prbuf,"%ld\n", ltemp);
```

```
    SetDlgItemText(hOurDlg, IDD_BITS,  (LPSTR) prbuf);
    } /* end if (total_time) */

    /* All done */
    SetDlgItemText(hOurDlg, IDD_COMMENT, "...TCP Shout Done");
    return bytes_sent;
}

/* eof */
```

3.4.3.7 tlisten.c 清单

```
/*
 * 文件名：TLISTEN.C
 */

#include "wshout.h"

/* MSC Include files: */
#include <stdio.h>
#include <io.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>


/* Returns the number of bytes written */
long TReadData(SOCKET hSock, HWND hOurDlg, int read_len)
{
    static char  ReadBuf[BUF_SIZE];
    SOCKET hAcceptSock;
    struct sockaddr_in local;  /* Local machine address structure */
    long total_time = 0L;       /* variable to hold delta t    */
    int   tmp, len   = 0;
    int   num_reads  = 0;
    long bytes_read = 0L;
    long last_time = 0L;
    long now = 0L;
```

```
long ltemp;
extern long blocking_option;
extern int run_cancelled;
struct linger AcceptLinger;

BOOL running = FALSE;
BOOL bTemp = TRUE;

SetDlgItemText(hOurDlg, IDD_COMMENT, "Awaiting the TCP Data ...");
SetDlgItemText(hOurDlg, IDD_HNAME, "                        ");

tmp = sizeof(local);
if (!blocking_option) {
hAcceptSock = accept(hSock,(struct sockaddr FAR *)&local,
   (int FAR *)&tmp);
}
else {
for (;;) {
   do {
   ;
   } while (ShoutBlockingHook()); /* Dispatch messages if any */

   if (run_cancelled) {
   WSASetLastError(WSAEINTR);
   break; /* Non-blocking mode was cancelled */
   }

   hAcceptSock = accept(hSock,(struct sockaddr FAR *)&local,
                 (int FAR *)&tmp);
   if (hAcceptSock == INVALID_SOCKET) {
   if (h_errno == WSAEWOULDBLOCK)
      /* Try again */
      ;
   else {
      /* Fatal error -- pop out. */
      break;
   }
   } /* end if ((hAcceptSock = .. */
```

```
        else {
        /* Success -- pop out. */
        break;
        }
    } /* end for */
    } /* end else */

    if (hAcceptSock == INVALID_SOCKET) {
    wshout_err (hOurDlg, WSAGetLastError(), "accept()");
    return 0;
    }

     /* Now, read the data as fast as we can until no more to read */
     time(&last_time);
     do {
    do {
         ;
    } while (ShoutBlockingHook()); /* Dispatch messages while available
*/

    if (run_cancelled) {
         WSASetLastError(WSAEINTR);
         break; /* Non-blocking mode was cancelled */
    }

    len = recv(hAcceptSock, ReadBuf, read_len, 0);
    if (len == SOCKET_ERROR) {
         if (h_errno == WSAEWOULDBLOCK)
        continue;
         else
        break;
    }
    else if (len == 0)
         break;
    num_reads++;
    bytes_read += len;

    wsprintf((LPSTR)prbuf,"%d\n",num_reads);
```

```
SetDlgItemText(hOurDlg, IDD_WRITE, (LPSTR) prbuf);

wsprintf((LPSTR)prbuf,"%ld\n",bytes_read);
SetDlgItemText(hOurDlg, IDD_SENT, (LPSTR) prbuf);

if (bTemp) {  /* To update our main display once */
    /* Do not use wsprintf() or you will add an extra char */
     _fmemcpy(prbuf, inet_ntoa(local.sin_addr), 4*sizeof(u_long));
     SetDlgItemText(hOurDlg, IDD_HNAME, (LPSTR) prbuf);
     SetDlgItemText(hOurDlg, IDD_COMMENT, "Reading TCP Data ...");
     bTemp = FALSE;
}

} while ((len != 0) || (len != SOCKET_ERROR));
time (&now);

if (len == SOCKET_ERROR) {
if ((h_errno == WSAESHUTDOWN) || (h_errno == WSAENOTCONN)) {
    /* nothing available for read. */
    wsprintf((LPSTR)prbuf,
   "Connection from %s closed.\n",inet_ntoa(local.sin_addr));
    SetDlgItemText(hOurDlg, IDD_COMMENT, prbuf);
}
else { /* Other error */
    wshout_err (hOurDlg, WSAGetLastError(), "recv()");
}
}
else if (len == 0) {
/* Other side shut down the connection */
wsprintf((LPSTR)prbuf,
    "Connection from %s closed.\n",inet_ntoa(local.sin_addr));
SetDlgItemText(hOurDlg, IDD_COMMENT, prbuf);
}

AcceptLinger.l_onoff = 1;
AcceptLinger.l_linger = 0;
ret = setsockopt(hAcceptSock, SOL_SOCKET, SO_LINGER,
            (char FAR *) &AcceptLinger, sizeof(AcceptLinger));
```

```
    if (ret == SOCKET_ERROR) {
    wshout_err (hOurDlg, WSAGetLastError(), "setsockopt()");
    }

    ret = closesocket(hAcceptSock);
    if (ret == SOCKET_ERROR) {
    wshout_err (hOurDlg, WSAGetLastError(), "closesocket()");
    }

    total_time = (long) difftime(now, last_time);
    if (total_time == 0)
    total_time = 1L;     /* Avoid dividing by zero */

    wsprintf((LPSTR)prbuf,"%ld\n",total_time);
    SetDlgItemText(hOurDlg, IDD_TIME, (LPSTR) prbuf);

    ltemp = num_reads/total_time;
    wsprintf((LPSTR)prbuf,"%ld\n", ltemp);
    SetDlgItemText(hOurDlg, IDD_WRITES,(LPSTR) prbuf);

    ltemp = bytes_read/total_time;
    wsprintf((LPSTR)prbuf,"%ld\n", ltemp);
    SetDlgItemText(hOurDlg, IDD_BYTES, (LPSTR) prbuf);

    ltemp = 8 * (bytes_read/total_time);
    wsprintf((LPSTR)prbuf,"%ld\n", ltemp);
    SetDlgItemText(hOurDlg, IDD_BITS,  (LPSTR) prbuf);

    if (bytes_read) {
    SetDlgItemText(hOurDlg, IDD_COMMENT, "...TCP Listen Done\n");
    } /* end: if (bytes_read) */

    return (bytes_read);
}

/* eof */
```

3.4.3.8 errno.c 清单

```
#include <windows.h>
#include <winsock.h>

/*
 * 文件名: ERRNO.C
 */

/*
 * Function: WSAsperror()
 *
 * Description:
 *
 * Copies string corresponding to the error code provided
 * into buf, maximum length len. Returns length actually
 * copied to buffer, or zero if error code is unknown.
 * String resources should be present for each error code
 * using the value of the code as the string ID (except for
 * error = 0, which is mapped to WSABASEERR to keep it with
 * the others). The DLL is free to use any string IDs that
 * are less than WSABASEERR for its own use.
 *
 */
int PASCAL FAR WSAsperror (HANDLE hInst,    /* Instance Handle */
             int errorcode,   /* WSA Error Number */
             char far * buf,  /* Buffer for error string */
             int len)         /* Length of buffer */
{
   int err_len;   /* length of error text */

       if (errorcode == 0)            /* If error passed is 0, use the
*/
             errorcode = WSABASEERR; /*  base resource file number */
        if (errorcode < WSABASEERR)     /* If invalid Error code */
               return 0;                /*  return string length of
zero */
```

```
    /* error string from the table in the Resource file into buffer */
        err_len = LoadString(hInst,errorcode,buf,len);

    return (err_len);  /* return length of error string retrieved */

}  /* end WSAsperror() */

/* eof */
```

3.4.3.9 resolve.c 清单

```
/*
 * 文件名: RESOLVE.C
 */

#include "wshout.h"

/* MSC Include files: */
#include <stdio.h>
#include <io.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>


SOCKET
ResolveAndConnectHost(LPSTR lpHostName,HWND hOurDlg,int iproto, int
iSockPort)
{
    struct hostent FAR *host_ptr;      /* Ptr to the host name  */
    struct sockaddr_in dest;    /* Addr of target host   */
    SOCKET hSock;             /* The socket to create */
    int iSockType;
    extern int iTCP;
    extern int iUDP;

    /* Internet family addressing */
    dest.sin_family = PF_INET;
```

```
if (iproto == iTCP) {
iSockType = SOCK_STREAM;
}
else if (iproto == iUDP) {
iSockType = SOCK_DGRAM;
}
else {
return (SOCKET) -1;      /* Unknown protocol */
}

/* default port to connect to. Must be in network byte order  */
dest.sin_port = htons((u_int) iSockPort);

SetDlgItemText(hOurDlg, IDD_COMMENT,"Resolving hostname...");

/* Resolve the host name */
host_ptr = gethostbyname(lpHostName);
if (host_ptr == NULL) {
wshout_err (hOurDlg, WSAGetLastError(), "gethostbyname()");
return (SOCKET) -1;
}
/* Patch host address into struct describing conn: */
bcopy(host_ptr->h_addr,&dest.sin_addr,host_ptr->h_length);

/* Allocate a network (socket) descriptor:       */
hSock = socket(PF_INET, iSockType, 0);
if (hSock == INVALID_SOCKET) {
wshout_err (hOurDlg, WSAGetLastError(), "socket()");
return (SOCKET) -1;
}

/* Start connection process to host described in 'dest' *
 * struct.
 */
SetDlgItemText(hOurDlg, IDD_COMMENT, "Connecting ...");
ret=connect(hSock,(struct sockaddr FAR *)&dest,sizeof(dest));

if (ret == SOCKET_ERROR) {
```

```
    wshout_err (hOurDlg, WSAGetLastError(), "connect()");
    closesocket(hSock);
    return (SOCKET) -1;
    }

    SetDlgItemText(hOurDlg, IDD_COMMENT, "...Connected");
    return hSock;
}


SOCKET GetSocketAndBind(HWND hOurDlg, int iProto, int iSockPort)
{
    SOCKET hSock;        /* Connection socket descriptor  */
    struct sockaddr_in local;      /* Local machine address structure*/
    int iSockType;
    extern int iTCP;
    extern int iUDP;

    /* Internet family addressing */
    if (iProto == iTCP) {
    iSockType = SOCK_STREAM;
    }
    else {
    iSockType = SOCK_DGRAM;
    }

    memset(&local, '\0', sizeof (local));
    local.sin_family = PF_INET;
    local.sin_port = htons((u_short)iSockPort);

    /* allocate a socket descriptor */
    hSock = socket(PF_INET, iSockType, 0);
    if (hSock == INVALID_SOCKET) { /* socket() failed   */
    wshout_err (hOurDlg, WSAGetLastError(), "socket()");
    return (SOCKET) -1;
    }

    /* bind socket to a local addr */
```

```
    ret = bind(hSock, (struct sockaddr FAR *) &local, sizeof(local));
    if (ret == SOCKET_ERROR){       /* bind() failed     */
   wshout_err (hOurDlg, WSAGetLastError(), "bind()");
   return (SOCKET) -1;
    }

    if (iProto == iUDP)
   return (hSock);

    /* If iProto == iTCP, then must listen() and accept() also */

    ret = listen(hSock, 0);     /* listen on the socket */
    if (ret == SOCKET_ERROR){   /* listen() failed      */
   wshout_err (hOurDlg, WSAGetLastError(), "listen()");
   return (SOCKET) -1;
    }

    return(hSock);
}

/* eof */
```

第四章 Windows Socket 1.1 库函数概览

4.1 套接口函数

Windows Sockets 规范包含了以下 Berkeley 风格的套接口例程：

accept()* 响应联结请求，并且新建一个套接口。原来的套接口则返回监听状态。

bind() 把一个本地的名字和一个无名的套接口捆绑起来。

closesocket()* 把套接口从拥有对象参考表中取消。该函数只有在 SO_LINGER 被设置时才会阻塞。

connect()* 初始化到一个指定套接口上的连接。

getpeername() 得到连接在指定套接口上的对等通讯方的名字。

getsockname() 得到指定套接口上当前的名字。

getsockopt() 得到与指定套接口相关的属性选项。

htonl() 把 32 位的数字从主机字节顺序转换到网络字节顺序。

htons() 把 16 位的数字从主机字节顺序转换到网络字节顺序。

inet_addr() 把一个 Internet 标准的"."记号地址转换成 Internet 地址数值。

inet_ntoa() 把 Internet 地址数值转换成带"."的 ASCII 字符串。

ioctlsocket() 为套接口提供控制。

listen() 监听某一指定套接口上连接请求的到来。

ntohl()	把 32 位数字从网络字节顺序转换为主机字节顺序。
ntons()	把 16 位数字从网络字节顺序转换为主机字节顺序。
recv()*	从一个已连接的套接口接收数据。
recvfrom()*	从一个已连接的或未连接的套接口接收数据。
select()*	执行同步 I/O 多路复用。
send()*	从一已连接的套接口发送数据。
sendto()*	从已连接或未连接的套接口发送数据。
setsockopt()	设置与指定套接口相关的属性选项。
shutdown()	关闭一部分全双工的连接。
socket()	创建一个通讯端点并返回一个套接口。

*表示例程在应用于阻塞套接口时会阻塞。

4.1.1 阻塞/非阻塞和数据易失性

阻塞是在把应用程序从 Berkeley 套接口环境中移植到 Windows 环境中的一个主要焦点。阻塞是指唤起一个函数，该函数直到相关操作完成时才返回。由于操作可能需要任意长的时间才能完成，于是问题就出现了。最明显的一个例子就是 recv()，这个函数会一直处于阻塞状态直到收到对方系统发送的数据。在 Berkeley 套接口模型中，一个套接口的操作的缺省行为是阻塞方式的，除非程序员显式地请求该操作为非阻塞方式。我们强烈推荐程序员在尽可能的情况下使用非阻塞方式（异步方式）的操作。因为非阻塞方式的操作能够更好地在非占先的 Windows 环境下工作。程序员应该在绝对必要的时候才采用阻塞方式。而且在你必须使用阻塞方式的操作前仔细阅读并理解这一部分。

即使在阻塞方式下，有些操作（例如 bind(),getsockopt(),getpeername()）也会立刻完成。对于这些操作，阻塞方式和非阻塞方式并没有什么两样。其他一些操作（例如 recv()）可能立刻完成，也可能需要阻塞一段随机的时间才能完成。这都取决于不同的传输情况。当用于阻塞套接口时，这些操作被认为是工作于阻塞方式的，所有会阻塞的例程在以前或以后的列表中都打上了星号作标记。

在 Windows Sockets 实现中，一个无法立刻完成的阻塞操作是按如下方式处理的。DLL 先初始化操作，然后进入一个循环，在循环中发送收到的任何信息－为了使在必要时把处理器交给其他线程，然后检查 Windows Sockets 功能是否完成。如果功能完成了，或者 WSACancelBlockingCall()被唤起，阻塞操作以一个适当的返回值结束。完整的关于这种机制的描述，请参见 5.3.13 节，WSASetBlockingHook()，这一部分还包括了对于各种函数伪代码的讨论。

如果一个正在运行某一阻塞操作的进程收到了一个 Windows 消息，那么应用程序有可能试图发出另一个 Windows Sockets 调用，由于很难安全地处理这种情形，Windows Sockets 规范不支持这种应用程序的工作方式。在这种情况下，有两个函数可以帮助程序员。WSAIsBlocking()可以用来确定在该进程上是否有阻塞的 Windows Sockets 调用。WSACancelBlookingCall()可以用来取消在线的阻塞调用，如果有的话。任何其他的 Windows Sockets 函数如果在这种情况下被调用，则会失败并返回错误代码 WSAEINPROGRESS。要强调的是，这一限制适用于所有阻塞和非阻塞的操作。

虽然这种机制对于简单的应用程序已经足够了，但这不能支持高级应用程序的复杂的消息发送要求。（例如，那些 MDI 模型的用户）对于这样的应用程序，Windows Sockets API 设计了 WSASetBlockingHook()函数，这个函数可以允许程序员定义特殊的阻塞钩子来代替上面讨论的缺省消息发送例程。

只有在以下都为真时，Windows Sockets DLL 才调用阻塞钩子函数：例程是被定义为可以阻塞的，指定的套接口也是阻塞套接口，而且请求不能被立刻完成。(套接口是被缺省地设为阻塞方式的,但 IOCTL FIONBIO 和 WSAAsyncSelect()都可以把套接口设置成为非阻塞模式）。如果应用程序只使用非阻塞方式的套接口，而且使用 WSAAsyncSelect()和/或 WSAAsyncGetXByY()例程，而不是使用 select()和/或 getXbyY()例程，那么阻塞钩子函数就永远也不会被调用，应用程序也不用再操心由于阻塞钩子函数而带来的重入问题。

如果一个应用程序在唤起异步或非阻塞方式调用时使用了一个内存对象的指针（例如一个缓冲区，或者一个全程变量）作为参数，那么应用程序要保证那个对象在 Windows Sockets 实现的整个操作中都可得到并使用。应用程序不能再唤起可能影响到内存唤射或寻址能力的其他的 Windows 函数。在多线程系统中，应用程序也有责任使用某种同步机制来协调对内存对象的存取。Windows Sockets 实现不能，也不会提出这种事情。没有遵守这条规则，所可能产生的后果已不在规范讨论的范围之内。

4.2 数据库函数

Windows Sockets规范定义了如下数据库例程。正如我们先前提出的,Windows Sockets 提供者有可能不采用依赖于本地数据库的方式来实现这些函数。某些数据库例程返回的指针（例如 gethostbyname()）指向的区域是由 Windows Sockets

函数库分配的。这些指针指向的数据是易失的。它们只在该线程的下一个Windows Sockets API 调用前有效。此外，应用程序不应试图修改这个结构，或者释放其中的任何一部分。在一个线程中，这个结构只有一份拷贝。因此，应用程序应该在发出下一个 Windows Sockets API 调用以前把所需的信息拷贝下来。

gethostbyaddr()*　从网络地址得到对应的名字（有可能多个）和地址。

gethostbyname()*　从主机名得到对应的名字（有可能多个）和地址。

gethostname()　得到本地主机名。

getprotbyname()*　从协议名得到对应的协议名和数值。

getservbyname()*　从一个服务的名字得到对应的服务名以及端口号。

getservbyport()*　从一个端口号得到对应的服务名以及端口号。

*表示例程在某些情况下可能会阻塞。

4.3 针对 Microsoft Windows 的扩展函数

Windows Sockets 规范提供了许多在标准的 Berkelet 套接口例程之外的扩展函数。本质上，这些扩展的 API 是为了应用程序能更好地处理基于消息的异步发送的网络事件。虽然基于 Windows Sockets 的编程并不强制要使用这个扩展的 API 集（WSAStartup()和 WSACleanup()除外）但我们推荐应用程序开发者遵循 Microsoft Windows 的编程范例。

WSAAsyncGetHostByAddr()　一个标准的Berkeley的getXbyY()函数集合的异步版本。例如 WSAAsyncGetHostByName()函数提供了一个标准 Berkeley 的 gethostbyname()函数的异步基于消息的实现。

WSAAsyncGetHostByName()

WSAAsyncGetProtoByName()

WSAAsyncGetProtByNumber()

WSAAsyncGetServByName()

WSAAsyncGetServByPort()	
WSAAsyncSelect()	select()函数的异步版本。
WSACancelAsyncRequest()	取消一个未完成的 WSAAsyncGetXByY()函数的实例。
WSACancelBlockingCall()	取消未完成的阻塞的 API 调用。
WSACleanup()	从底层的 Windows Sockets DLL 中撤销注册。
WSAGetLastError()	得到最近的一个 Windows Sockets API 调用错误的详细情况。
WSAIsBlocking()	确定底层的 Windows Sockets DLL 是否在该线程已经被一个调用阻塞。
WSASetBlockingHook()	为底层的 Windows Sockets 实现设置阻塞钩子。
WSASetLastError()	设置下一次 WSAGetLastError()返回的错误信息。
WSAStartup()	初始化底层的 Windows Sockets DLL。
WSAUnhookBlockingHook()	恢复原始的阻塞钩子。

4.3.1 异步选择机制

WSAAsyncSelect()调用允许应用程序程序注册一个或多个感兴趣的网络事件。这一 API 调用用来取代探寻网络 I/O 调用。在 select()或非阻塞 I/O 例程（例如 send()和 recv()）已经被调用或将要被调用的情况下都可以使用 WSAAsyncSelect()调用。在这种情况下，在声明感兴趣的网络事件时，你必须提供一个通知时使用的窗口句柄。那么在你声明的感兴趣的网络事件发生时，对应的窗口将收到一个基于消息的通知。

Windows Sockets 允许对于一特定的套接口声明如下感兴趣的网络事件：

*套接口已准备读数据。
*套接口已准备写数据。
*带外数据准备好。
*套接口准备接收连接请求。
*非阻塞连接操作结束。
*连接关闭。

4.3.2 异步支持例程

异步数据库函数允许应用程序用异步方式请求信息。某些网络实现和/或配置，需要通过执行网络操作来应答这些请求。WSAAsyncGetXByY()函数允许应用程序开发者不必象在使用 Berkeley 标准函数时阻塞整个 Windows 环境。WSACancelAsyncRequest()函数可以允许一个应用程序取消任何未完成的异步的 WSAAsyncGetXByY()请求。

4.3.3 阻塞钩子函数方法

正如 4.1.1 节所讲述的，Windows Sockets 实现以这样一种方式阻塞一个操作，Windows 消息处理可以继续，发出调用的应用程序仍然可以收到 Windows 消息。但在某些情况下，应用程序可能希望影响或改变这种伪阻塞的实现方式。WSASetBlockingHook()函数就提供了这样一种功能。它使得应用程序可以替换 Windows Sockets 实现在“阻塞”操作中放弃处理器时调用的例程。

4.3.4 错误处理

为了与基于线程的环境兼容，API 调用的错误细节可以通过 WSAGetLastError()调用得到。虽然已经为大家接收的 Berkeley 风格的机制是通过"errno"得到关于套接口的网络错误的，这种机制不能够保证在多线程环境中错误代码的完整性和正确性。WSAGetLastError()允许程序员能够得到对应于每一线程的最近的错误代码。

WSAGetLastError()所返回的错误代码尽量避免与标准的 Microsoft C 错误代码冲突。但是某些 Windows Sockets 例程返回的错误代码是在 Microsoft C 定义的标准错误代码之内的。如果你使用的应用程序开发环境中的错误代码定义与 Microsoft C 不同，那么我们建议你使用 Windows Sockets 错误代码前缀"WSA"来保证准确的检测错误。

这份规范定义了一个推荐的错误代码的集合，而且列举了每一个函数有可能返回的错误。但是某些 Windows Sockets 实现也有可能返回一些在我们列举之外

的错误代码。应用程序应该具备能够处理在每个 API 描述下列举的错误代码之外的错误的能力。然而 Windows Sockets 实现不能返回在附录 4.1 中列举的合法 Windows Sockets 错误之外的任何数值。

4.3.5 通过中介 DLL 调用 Windows Sockets DLL

Windows Sockets DLL 既可以直接从应用程序中调用也可以通过中介 DLL 调用。通过中介 DLL 的例子是：使用 Windows Sockets 为应用程序实现一个提供通用网络功能的虚拟网络 API 层，这样的 DLL 可以同时被多个应用程序使用。担这样的 DLL 必须对 WSAStartup()和 WSACleanup()这两个函数非常警惕，它们必须保证在任何一个使用 Windows Sockets 调用的任务前后均调用了 WSAStartup()和 WSACleanup()。这是因为 Windows Sockets DLL 需要一个对 WSAStartup()的调用来为每个任务初始化其数据结构，也需要一个对 WSACleanup()的调用来释放为任务分配的所有资源。

有至少两种方法去完成这一任务。最简单的方法是中介 DLL 具有与 WSAStarup()和 WSACleanup()类似的调用提供给应用程序使用，DLL 将在这些例程中调用 WSAStartup()和 WSACleanup()。另一种机制就是中介 DLL 建立一个任务句柄列表。任务句柄是从 GetCurrentTask()这一个 Windows API 中获得的。在中介 DLL 的每一个入口处检查对于当前任务 WSAStartup()函数是否已被调用，并且在必要的时候调用 WSACleanup()函数。

在 Windows NT 环境中，这一点是没有必要的。因为 Windows NT 中的 DLL 结构与流程是与 Windows 不同的。在 Windows NT 中，中介 DLL 只需简单的在它的 DLL 初始化例程中调用 WSAStartup()即可。这个例程将在任何一个新的进程试图使用 DLL 的开始时刻被执行。

如果中介DLL调用了阻塞操作而又没有安装任何它自己的阻塞钩子,那么DLL作者必须清楚地认识到控制有可能会通过应用程序安装的阻塞钩子或缺省的阻塞钩子回到应用程序手中。这样应用程序有可能通过 WSACancelBlockingCall()来取消中介 DLL 的阻塞操作，如果这种情况发生了，中介 DLL 的阻塞操作会失败并返回错误代码 WSAEINTR。这时候，中介 DLL 必须尽快地把控制交还给调用它的任务。因为用户有可能按了 Cancel 或者 Close 按钮。应用程序正在急切地盼望获得 CPU 的控制权。我们推荐中介 DLL 在进行阻塞调用时安装自己的阻塞钩子来防止不可预见的中介 DLL 和应用程序之间的互相影响。

4.3.6 Windows Sockets 实现内部对消息的使用

为了把 Windows Sockets 实现成一个纯粹的 DLL，有时在 DLL 内部互相发送消息来通讯和定时是必要的。这是非常合法的。但是 Windows DLL 不应该无缘无

故地发送消息给一个由客户打开的窗口句柄，除非应用程序要求这些消息。所以为了自身的目的而需要使用消息的 Windows Sockets DLL 打开了一个隐藏的窗口，并且发送必要的消息给这个窗口的句柄。

4.3.7 私有的 API 接口

附录 B.3 中的 WINSOCK.DEF 文件列出了 Windows Sockets API 功能调用的序数。除了已经列出的序数值外，所有小于 999 的序数都是保留给将来的 Windows Sockets 使用的。对于一个 Windows Sockets 实现来说，提供附加的私有的接口也是很方便的。这是完全可以接受的，只要这些调用的序数大于 1000，要注意的是，任何使用了某个特定 Windows Sockets DLL 私有的 API 的应用程序极有可能在任何其他供应商的 Windows Sockets DLL 上无法工作。应该注意到，只有使用在这份规范中定义的 API 才能可以保证每一个 Windows Sockets 实现都支持。

如果一个应用程序使用了某个供应商的 Windows Sockets DLL 的特定接口，最好不要把应用程序与 DLL 静态连接，而通过 Windows Sockets 例程 LoadLibrary()和 GetProcAddress()动态载入，这就使得应用程序在其他不支持同样的扩展功能集的 Windows Sockets DLL 系统上运行时，可以得到适当的错误信息。

第五章 套接口库函数参考

5.1 Windows Socket 1.1 库函数参考

本章以字母顺序列出了套接口库函数，并介绍了技术细节。

使用任一库函数时应在程序中包含 WINSOCK.H 头文件。在附录 A.2 中还列出了一些与 BERKELEY 兼容的头文件。这些头文件只起到兼容性的作用，它们都包含了 WINSOCK.H 头文件，WINDOWS.H 头文件也是必需的，但 WINSOCK.H 会视需要包含这一头文件。

5.1.1 accept()

简述：

在一个套接口接受一个连接。

```
#include <winsock.h>

SOCKET PASCAL FAR accept( SOCKET s, struct sockaddr FAR* addr,
int FAR* addrlen);
```

s：套接口描述字，该套接口在 listen()后监听连接。

addr：（可选）指针，指向一缓冲区，其中接收为通讯层所知的连接实体的地址。Addr 参数的实际格式由套接口创建时所产生的地址族确定。

addrlen：（可选）指针，指向存有 addr 地址长度的整形数。

注释：

本函数从 s 的等待连接队列中抽取第一个连接，创建一个与 s 同类的新的套接口并返回句柄。如果队列中无等待连接，且套接口为非阻塞方式，则 accept()阻塞调用进程直至新的连接出现。如果套接口为非阻塞方式且队列中等待连接，则 accept()返回一错误代码。已接受连接的套接口不能用于接受新的连接，原套接口仍保持开放。

addr 参数为一个返回参数，其中填写的是为通讯层所知的连接实体地址。addr 参数的实际格式由通讯时产生的地址族确定。addrlen 参数也是一个返回参数，在调用时初始化为 addr 所指的地址空间；在调用结束时它包含了实际返回的地

址的长度（用字节数表示）。该函数与 SOCK_STREAM 类型的面向连接的套接口一起使用。如果 addr 与 addrlen 中有一个为零 NULL，将不返回所接受的套接口远程地址的任何信息。

返回值：

如果没有错误产生，则 accept()返回一个描述所接受包的 SOCKET 类型的值。否则的话，返回 INVALID_SOCKET 错误，应用程序可通过调用 WSAGetLastError()来获得特定的错误代码。

addrlen 所指的整形数初始时包含 addr 所指地址空间的大小，在返回时它包含实际返回地址的字节长度。

错误代码：

WSANOTINITIALISED：在使用此 API 之前应首先成功地调用 WSAStartup()。
WSAENETDOWN：WINDOWS 套接口实现检测到网络子系统失效。
WSAEFAULT：addrlen 参数太小（小于 socket 结构的大小）。
WSAEINTR：通过一个 WSACancelBlockingCall()来取消一个（阻塞的）调用。
WSAEINPROGRESS：一个阻塞的 WINDOWS 套接口调用正在运行中。
WSAEINVAL：在 accept()前未激活 listen()。
WSAEMFILE：调用 accept()时队列为空，无可用的描述字。
WSAENOBUFS：无可用缓冲区空间。
WSAENOTSOCK：描述字不是一个套接口。
WSAEOPNOTSUPP：该套接口类型不支持面向连接服务。
WSAEWOULDBLOCK：该套接口为非阻塞方式且无连接可供接受。

参见：

bind(), connect(), listen(), select(), socket(), WSAAsyncSelect().

5.1.2 bind()

简述：

将一本地地址与一套接口捆绑。

```
#include <winsock.h>

int PASCAL FAR bind( SOCKET s, const struct sockaddr FAR* name,
int namelen);
```

s：标识一未捆绑套接口的描述字。
name：赋予套接口的地址。sockaddr 结构定义如下：

```
struct sockaddr{
  u_short sa_family;
  char sa_data[14];
};
```

namelen：name 名字的长度。

注释：

本函数适用于未连接的数据报或流类套接口，在 connect()或 listen()调用前使用。当用 socket()创建套接口后，它便存在于一个名字空间（地址族）中，但并未赋名。bind()函数通过给一个未命名套接口分配一个本地名字来为套接口建立本地捆绑（主机地址/断口号）。

在 Internet 地址族中，一个名字包括几个组成部分，对于 SOCK_PGRAM 和 SOCK_STREAM 类套接口，名字由三部分组成：主机地址，协议号（显式设置为 UDP 和 TCP）和用以区分应用的端口号。如果一个应用并不关心分配给它的地址，则可将 Internet 地址设置为 INADDR_ANY，或将端口号置为 0。如果 Internet 地址段为 INADDR_ANY，则可使用任意网络接口；在有多种主机环境下可简化编程。如果端口号置为 0，则 WINDOWS 套接口实现将给应用程序分配一个值在 1024 到 5000 之间的唯一的端口。应用程序可在 bind()后用 getsockname()来获知所分配的地址，但必需注意的是，getsockname()只有在套接口连接成功后才会填写 Internet 地址，这是由于在多种主机环境下若干种 Internet 地址都是有效的。

如果一个应用程序需要把端口捆绑到超过 1024－5000 范围的特定端口时，比如 rsh 需要捆绑到任一保留端口，则可如下编程：

```
SOCKADDR_IN sin;
SOCKET s;
u_short alport=IPPORT_RESERVED;
sin.sin_family=AF_INET;
sin.sin_addr.s_addr=0;
for (;;) {
  sin.sin_port=htons(alport);
  if (bind(s,(LPSOCKADDR)&sin, sizeof(sin))=0) {
      /* it worked */
  }
  if (GetLastError()!=WSAEADDRINUSE) {
```

```
        /* fail */
    }
    alport-;
    if (alport=IPPORT_RESERVED/2) {
        /* fail—all unassigned reserved ports are */
        /* in use. */
    }
  }
```

返回值：

如无错误发生，则 bind()返回 0。否则的话，将返回 SOCKET_ERROR，应用程序可通过 WSAGetLastError()获取相应错误代码。

错误代码：

WSANOTINITIALISED：在使用此 API 之前应首先成功地调用 WSAStartup()。

WSAENETDOWN：WINDOWS 套接口实现检测到网络子系统失效。

WSAEADDRINUSE：所定端口已在使用中（参见 setoption()中的 SO_REUSEADDR 选项）。

WSAEFAULT：namelen 参数太小（小于 sockaddr 结构的大小）。

WSAEINPROGRESS：一个阻塞的 WINDOWS 套接口调用正在运行中。

WSAEAFNOSUPPORT：本协议不支持所指定的地址族。

WSAEINVAL：该套接口已与一个地址捆绑。

WSAENOBUFS：无足够可用缓冲区，连接过多。

WSAENOTSOCK：描述字不是一个套接口。

参见：

connect(), listen(), getsockname(), setsockopt(), socket(), WSACancelBlockingCall().

5.1.3 closesocket()

简述：

关闭一个套接口。

```
#include <winsock.h>
```

int PASCAL FAR closesocket(SOCKET s);

s：一个套接口的描述字。

注释：

本函数关闭一个套接口。更确切地说，它释放套接口描述字 s，以后对 s 的访问均以 WSAENOTSOCK 错误返回。若本次为对套接口的最后一次访问，则相应的名字信息及数据队列都将被释放。closesocket()的语义受 SO_LINGER 与 SO_DONTLINGER 选项影响，对比如下：

选项	间隔	关闭方式	等待关闭与否
SO_DONTLINGER	不关心	优雅	否
SO_LINGER	零	强制	否
SO_LINGER	非零	优雅	是

若设置了 SO_LINGER（亦即 linger 结构中的 l_onoff 域设为非零，参见 2.4，4.1.7 和 4.1.21 各节），并设置了零超时间隔，则 closesocket()不被阻塞立即执行，不论是否有排队数据未发送或未被确认。这种关闭方式称为“强制”或“失效”关闭，因为套接口的虚电路立即被复位，且丢失了未发送的数据。在远端的 recv()调用将以 WSAECONNRESET 出错。

若设置了 SO_LINGER 并确定了非零的超时间隔，则 closesocket()调用阻塞进程，直到所剩数据发送完毕或超时。这种关闭称为“优雅的”关闭。请注意如果套接口置为非阻塞且 SO_LINGER 设为非零超时，则 closesocket()调用将以 WSAEWOULDBLOCK 错误返回。

若在一个流类套接口上设置了 SO_DONTLINGER（也就是说将 linger 结构的 l_onoff 域设为零；参见 2.4，4.1.7，4.1.21 节），则 closesocket()调用立即返回。但是，如果可能，排队的数据将在套接口关闭前发送。请注意，在这种情况下 WINDOWS 套接口实现将在一段不确定的时间内保留套接口以及其他资源，这对于想用所以套接口的应用程序来说有一定影响。

返回值：

如无错误发生，则 closesocket()返回 0。否则的话，返回 SOCKET_ERROR 错误，应用程序可通过 WSAGetLastError()获取相应错误代码。

错误代码：

WSANOTINITIALISED：在使用此 API 之前应首先成功地调用 WSAStartup()。

WSAENETDOWN：WINDOWS 套接口实现检测到网络子系统失效。

WSAENOTSOCK：描述字不是一个套接口。

WSAEINPROGRESS：一个阻塞的 WINDOWS 套接口调用正在运行中。
WSAEINTR：通过一个 WSACancelBlockingCall()来取消一个（阻塞的）调用。
WSAEWOULDBLOCK :该套接口设置为非阻塞方式且 SO_LINGER 设置为非零超时间隔。

参见：
accept(), socket(), ioctlsocket(), setsockopt(), WSAAsyncSelect().

5.1.4 connect()

简述：
建立与一个端的连接。

```
#include <winsock.h>

int PASCAL FAR connect( SOCKET s, const struct sockaddr FAR* name,
int namelen);
```

s：标识一个未连接套接口的描述字。
name：欲进行连接的端口名。
namelen：名字长度。

注释：
本函数用于创建与指定外部端口的连接。s 参数指定一个未连接的数据报或流类套接口。如套接口未被捆绑，则系统赋给本地关联一个唯一的值，且设置套接口为已捆绑。请注意若名字结构中的地址域为全零的话，则 connect()将返回 WSAEADDRNOTAVAIL 错误。
对于流类套接口（SOCK_STREAM 类型），利用名字来与一个远程主机建立连接，一旦套接口调用成功返回,它就能收发数据了。对于数据报类套接口(SOCK_DGRAM 类型），则设置成一个缺省的目的地址，并用它来进行后续的 send()与 recv()调用。

返回值：
若无错误发生，则 connect()返回 0。否则的话，返回 SOCKET_ERROR 错误，应用程序可通过 WSAGetLastError()获取相应错误代码。对阻塞套接口而言，若返回值为 SOCKET_ERROR 则应用程序调用 WSAGetLsatError()。如果它指出错误代码为 WSAEWOULDBLOCK，则您的应用程序可以：
1.用 select()，通过检查套接口是否可写，来确定连接请求是否完成。或者，

2.如果您的应用程序使用基于消息的WSAAsynSelect()来表示对连接事件的兴趣，则当连接操作完成后，您会收到一个 FD_CONNECT 消息。

错误代码：

WSAENOTINITIALISED：在使用此 API 之前应首先成功地调用 WSAStartup()。
WSAENETDOWN：WINDOWS 套接口实现检测到网络子系统失效。
WSAEADDRINUSE：所指的地址已在使用中。
WSAEINTR：通过一个 WSACancelBlockingCall()来取消一个（阻塞的）调用。
WSAEINPROGRESS：一个阻塞的 WINDOWS 套接口调用正在运行中。
WSAEADDRNOTAVAIL：在本地机器上找不到所指的地址。
WSAENOTSUPPORT：所指族中地址无法与本套接口一起使用。
WSAECONNREFUSED：连接尝试被强制拒绝。
WSAEDESTADDREQ：需要目的地址。
WSAEFAULT：namelen 参数不正确。
WSAEINVAL：套接口没有准备好与一地址捆绑。
WSAEISCONN：套接口早已连接。
WSAEMFILE：无多余文件描述字。
WSAENETUNREACH：当前无法从本主机访问网络。
WSAENOBUFS：无可用缓冲区。套接口未被连接。
WSAENOTSOCK：描述字不是一个套接口。
WSAETIMEOUT：超时时间到。
WSAEWOULDBLOCK 套接口设置为非阻塞方式且连接不能立即建立。可用select()调用对套接口写，因为 select()时会进行连接。

参见：

accept(), bind(), getsockname(), socket(), select(), WSAAsyncSelect().

5.1.5 getpeername()

简述：

获取与套接口相连的端地址。

```
#include <winsock.h>

int PASCAL FAR getpeername( SOCKET s, struct sockaddr FAR* name,
int FAR* namelen);
```

s：标识一已连接套接口的描述字。
name：接收端地址的名字结构。
namelen：一个指向名字结构的指针。

注释：
getpeername()函数用于从端口 s 中获取与它捆绑的端口名，并把它存放在 sockaddr 类型的 name 结构中。它适用于数据报或流类套接口。

返回值：
若无错误发生，getpeername()返回 0。否则的话，返回 SOCKET_ERROR，应用程序可通过 WSAGetLastError()来获取相应的错误代码。

错误代码：
WSANOTINITIALISED：在使用此 API 之前应首先成功地调用 WSAStartup()。
WSAENETDOWN：WINDOWS 套接口实现检测到网络子系统失效。
WSAEFAULT：namelen 参数不够大。
WSAEINPROGRESS：一个阻塞的 WINDOWS 套接口调用正在运行中。
WSAENOTCONN 套接口未连接。
WSAENOTSOCK：描述字不是一个套接口。

参见：
bind(), socket(), getsockname().

5.1.6 getsockname()

简述：
获取一个套接口的本地名字。

```
#include <winsock.h>

int PASCAL FAR getsockname( SOCKET s, struct sockaddr FAR* name,
int FAR* namelen);
```

s：标识一个已捆绑套接口的描述字。
name：接收套接口的地址（名字）。
namelen：名字缓冲区长度。

注释：

getsockname()函数用于获取一个套接口的名字。它用于一个已捆绑或已连接套接口 s，本地地址将被返回。本调用特别适用于如下情况：未调用 bind()就调用了 connect()，这时唯有 getsockname()调用可以获知系统内定的本地地址。在返回时，namelen 参数包含了名字的实际字节数。

若一个套接口与 INADDR_ANY 捆绑，也就是说该套接口可以用任意主机的地址，此时除非调用 connect()或 accept()来连接，否则 getsockname()将不会返回主机 IP 地址的任何信息。除非套接口被连接，WINDOWS 套接口应用程序不应假设 IP 地址会从 INADDR_ANY 变成其他地址。这是因为对于多个主机环境下，除非套接口被连接，否则该套接口所用的 IP 地址是不可知的。

返回值：

若无错误发生，getsockname()返回 0。否则的话，返回 SOCKET_ERROR 错误，应用程序可通过 WSAGetLastError()获取相应错误代码。

错误代码：

WSANOTINITIALISED：在使用此 API 之前应首先成功地调用 WSAStartup()。
WSAENETDOWN：WINDOWS 套接口实现检测到网络子系统失效。
WSAEFAULT：namelen 参数不够大。
WSAEINPROGRESS：一个阻塞的 WINDOWS 套接口调用正在运行中。
WSAENOTSOCK：描述字不是一个套接口。
WSAEINVAL：套接口未用 bind()捆绑。

参见：

bind(), socket(), getpeername().

5.1.7 getsockopt()

简述：

获取一个套接口选项。

```
#include <winsock.h>

int PASCAL FAR getsockopt( SOCKET s, int level, int optname,
char FAR* optval, int FAR* optlen);
```

s：一个标识套接口的描述字。
level：选项定义的层次。支持的层次仅有 SOL_SOCKET 和 IPPROTO_TCP。
optname：需获取的套接口选项。
optval：指针，指向存放所获得选项值的缓冲区。
optlen：指针，指向 optval 缓冲区的长度值。

注释：
getsockopt()函数用于获取任意类型、任意状态套接口的选项当前值，并把结果存入 optval。在不同协议层上存在选项，但往往是在最高的“套接口”层次上，设置选项影响套接口的操作，诸如操作的阻塞与否、包的选径方式、带外数据的传送等。
被选中选项的值放在 optval 缓冲区中。optlen 所指向的整形数在初始时包含缓冲区的长度，在调用返回时被置为实际值的长度。对 SO_LINGER 选项而言，相当于 linger 结构的大小，对其他选项来说，是一个整形数的大小。
如果未进行 setsockopt()调用，则 getsockopt()返回系统缺省值。
getsockopt()支持下列选项。其中“类型”栏指出了 optval 所指向的值。仅有 TCP_NODELAY 选项使用了 IPPROTO_TCP 层；其余选项均使用 SOL_SOCKET 层。

选项	类型	意义
SO_ACCEPTCONN	BOOL	套接口正在用 listen()监听。
SO_BROADCAST	BOOL	套接口设置为传送广播信息。
SO_DEBUG	BOOL	允许调试。
SO_DONTLINER	BOOL	若为真，则 SO_LINGER 选项被禁止。
SO_DONTROUTE	BOOL	禁止选径。
SO_ERROR	int	获取错误状态并清除。
SO_KEEPALIVE	BOOL	发送“保持活动”信息。
SO_LINGER	struct linger FAR*	返回当前各 linger 选项。
SO_OOBINLINE	BOOL	在普通数据流中接收带外数据。
SO_RCVBUF	int	接收缓冲区大小。
SO_REUSEADDR	BOOL	套接口能和一个已在使用中的地址捆绑。
SO_SNDBUF	int	发送缓冲区大小。
SO_TYPE	int	套接口类型（如 SOCK_STREAM）。
TCP_NODELAY	BOOL	禁止发送合并的 Nagle 算法。

getsockopt()不支持的 BSD 选项有：

选项名	类型	意义

SO_RCVLOWAT int　　接收低级水印。
SO_RCVTIMEO int　　接收超时。
SO_SNDLOWAT int　　发送低级水印。
SO_SNDTIMEO int　　发送超时。
IP_OPTIONS　　获取 IP 头中选项。
TCP_MAXSEG int　　获取 TCP 最大段的长度。

用一个未被支持的选项去调用 getsockopt()将会返回一个 WSAENOPROTOOPT 错误代码（可用 WSAGetLastError()获取）。

返回值：

若无错误发生，getsockopt()返回 0。否则的话，返回 SOCKET_ERROR 错误，应用程序可通过 WSAGetLastError()获取相应错误代码。

错误代码：

WSANOTINITIALISED：在使用此 API 之前应首先成功地调用 WSAStartup()。
WSAENETDOWN：WINDOWS 套接口实现检测到网络子系统失效。
WSAEFAULT：optlen 参数非法。
WSAEINPROGRESS：一个阻塞的 WINDOWS 套接口调用正在运行中。
WSAENOPROTOOPT：未知或不支持选项。其中，SOCK_STREAM 类型的套接口不支持 SO_BROADCAST 选项，SOCK_DGRAM 类型的套接口不支持 SO_ACCEPTCONN、SO_DONTLINGER 、SO_KEEPALIVE、SO_LINGER 和 SO_OOBINLINE 选项。
WSAENOTSOCK：描述字不是一个套接口。

参见：

setsockopt(), WSAAsyncSelect(), socket().

5.1.8 htonl()

简述：

将主机的无符号长整形数转换成网络字节顺序。

```
#include <winsock.h>

u_long PASCAL FAR htonl( u_long hostlong);
```

hostlong：主机字节顺序表达的 32 位数。

注释：
本函数将一个 32 位数从主机字节顺序转换成网络字节顺序。

返回值：
htonl()返回一个网络字节顺序的值。

参见：
htons(), ntohl(), ntohs().

5.1.9 htons()

简述：
将主机的无符号短整形数转换成网络字节顺序。

```
#include <winsock.h>

u_short PASCAL FAR htons( u_short hostshort);
```

hostshort：主机字节顺序表达的 16 位数。

注释：
本函数将一个 16 位数从主机字节顺序转换成网络字节顺序。

返回值：
htons()返回一个网络字节顺序的值。

参见：
htonl(), ntohl(), ntohs().

5.1.10 inet_addr()

简述：将一个点间隔地址转换成一个 in_addr。

```
#include <winsock.h>

unsigned long PASCAL FAR inet_addr( const struct FAR* cp);
```

cp：一个以 Internet 标准“ . ”间隔的字符串。

注释：

本函数解释 cp 参数中的字符串，这个字符串用 Internet 的“ . ”间隔格式表示一个数字的 Internet 地址。返回值可用作 Internet 地址。所有 Internet 地址以网络字节顺序返回(字节从左到右排列)。

Internet 地址用“ . ”间隔的地址可有下列几种表达方式：

a.b.c.d，a.b.c，a.b，a

当四个部分都有定值时，每个都解释成一个字节数据，从左到右组成 Internet 四字节地址。请注意，当一个 Internet 地址在 Intel 机器上表示成一个 32 位整型数时，则上述的字节为“ d.c.b.a ”。这是因为 Intel 处理器的字节是从右向左排列的。

请注意：只有 Berkeley 支持下述表达法，Internet 其余各处均不支持。考虑到与软件的兼容性，应按规定进行使用。

对一个三部分地址，最后一部分解释成 16 位数据并作为网络地址的最右两个字节。这样，三部分地址便很容易表示 B 组网络地址，如“ 128.net.host ”.

对一个两部分地址，最后一部分解释成 24 位数据并作为网络地址的最右三个字节，这样，两部分地址便很容易表示 C 组网络地址，如“ net.host ”。

对仅有一个部分的地址，则将它的值直接存入网络地址不作任何字节的重组。

返回值：

若无错误发生，inet_addr()返回一个无符号长整型数，其中以适当字节顺序存放 Internet 地址。如果传入的字符串不是一个合法的 Internet 地址，如“ a.b.c.d ”地址中任一项超过 255，那么 inet_addr()返回 INADDR_NONE。

参见：

inet_ntoa().

5.1.11 inet_ntoa()

简述：

将网络地址转换成“ . ”点隔的字符串格式。

```
#include <winsock.h>

char FAR* PASCAL FAR inet_ntoa( struct in_addr in);
```

in：一个表示 Internet 主机地址的结构。

注释：

本函数将一个用 in 参数所表示的 Internet 地址结构转换成以“.”间隔的诸如“a.b.c.d”的字符串形式。请注意 inet_ntoa()返回的字符串存放在 WINDOWS 套接口实现所分配的内存中。应用程序不应假设该内存是如何分配的。在同一个线程的下一个 WINDOWS 套接口调用前，数据将保证是有效。

返回值：

若无错误发生，inet_ntoa()返回一个字符指针。否则的话，返回 NVLL。其中的数据应在下一个 WINDOWS 套接口调用前复制出来。

参见：

inet_addr().

5.1.12 ioctlsocket()

简述:

控制套接口的模式。

```
#include <winsock.h>

int PASCAL FAR ioctlsocket( SOCKET s, long cmd, u_long FAR* argp);
```

s：一个标识套接口的描述字。
cmd：对套接口 s 的操作命令。
argp：指向 cmd 命令所带参数的指针。

注释：

本函数可用于任一状态的任一套接口。它用于获取与套接口相关的操作参数，而与具体协议或通讯子系统无关。支持下列命令：

FIONBIO：允许或禁止套接口 s 的非阻塞模式。argp 指向一个无符号长整型。如允许非阻塞模式则非零，如禁止非阻塞模式则为零。当创建一个套接口时，它就处于阻塞模式（也就是说非阻塞模式被禁止）。这与 BSD 套接口是一致的。WSAAsynSelect()函数将套接口自动设置为非阻塞模式。如果已对一个套接口进行了 WSAAsynSelect() 操作，则任何用 ioctlsocket()来把套接口重新设置成阻

塞模式的试图将以 WSAEINVAL 失败。为了把套接口重新设置成阻塞模式，应用程序必须首先用 WSAAsynSelect()调用（IEvent 参数置为 0）来禁至 WSAAsynSelect()。

FIONREAD：确定套接口 s 自动读入的数据量。argp 指向一个无符号长整型，其中存有 ioctlsocket()的返回值。如果 s 是 SOCKET_STREAM 类型，则 FIONREAD 返回在一次 recv()中所接收的所有数据量。这通常与套接口中排队的数据总量相同。如果 S 是 SOCK_DGRAM 型，则 FIONREAD 返回套接口上排队的第一个数据报大小。

SIOCATMARK：确实是否所有的带外数据都已被读入。这个命令仅适用于 SOCK_STREAM 类型的套接口，且该套接口已被设置为可以在线接收带外数据（SO_OOBINLINE）。如无带外数据等待读入，则该操作返回 TRUE 真。否则的话返回 FALSE 假，下一个 recv()或 recvfrom()操作将检索“标记”前一些或所有数据。应用程序可用 SIOCATMARK 操作来确定是否有数据剩下。如果在“紧急”（带外）数据前有常规数据,则按序接收这些数据（请注意,recv()和 recvfrom()操作不会在一次调用中混淆常规数据与带外数据）。argp 指向一个 BOOL 型数，ioctlsocket()在其中存入返回值。

兼容性：

本函数为 Berkeley 套接口函数 ioctl()的一个子集。其中没有与 FIOASYNC 等价的命令，SIOCATMARK 是套接口层次支持的唯一命令。

返回值：

成功后，ioctlsocket()返回 0。否则的话，返回 SOCKET_ERROR 错误，应用程序可通过 WSAGetLastError()获取相应错误代码。

错误代码：

WSANOTINITIALISED：在使用此 API 之前应首先成功地调用 WSAStartup()。

WSAENETDOWN：WINDOWS 套接口实现检测到网络子系统失效。

WSAEINVAL：cmd 为非法命令，或者 argp 所指参数不适用于该 cmd 命令，或者该命令
不适用于此种类型的套接口。

WSAEINPROGRESS：一个阻塞的 WINDOWS 套接口调用正在运行中。

WSAENOTSOCK：描述字不是一个套接口。

参见：

socket(), setsockopt(), getsockopt(), WSAAsyncSelect().

5.1.13 listen()

简述：

创建一个套接口并监听申请的连接.

```
#include <winsock.h>

int PASCAL FAR listen( SOCKET s, int backlog);
```

S：用于标识一个已捆绑未连接套接口的描述字。
backlog：等待连接队列的最大长度。

注释：

为了接受连接，先用 socket()创建一个套接口，然后用 listen()为申请进入的连接建立一个后备日志，然后便可用 accept()接受连接了。listen()仅适用于支持连接的套接口，如 SOCK_STREAM 类型的。套接口 s 处于一种“ 变动 ” 模式，申请进入的连接请求被确认，并排队等待被接受。这个函数特别适用于同时有多个连接请求的服务器；如果当一个连接请求到来时，队列已满，那么客户将收到一个 WSAECONNREFUSED 错误。

当没有可用的描述字时，listen()函数仍试图正常地工作。它仍接受请求直至队列变空。当有可用描述字时，后续的一次 listen()或 accept()调用会将队列按照当前或最近的“ 后备日志 ” 重新填充，如有可能的话，将恢复监听申请进入的连接请求。

兼容性：

后备日志当前被（默认地）限制为 5。如同 4.3 BSD Unix 中的一样，小于 1 或大于 5 的数都会被舍入最近的有效值。

返回值：

如无错误发生，listen()返回 0。否则的话，返回 SOCKET_ERROR 错误，应用程序可通过 WSAGetLastError()获取相应错误代码。

错误代码：

WSANOTINITIALISED：在使用此 API 之前应首先成功地调用 WSAStartup()。
WSAENETDOWN：WINDOWS 套接口实现检测到网络子系统失效。
WSAEADDRINUSE：试图用 listen()去监听一个正在使用中的地址。
WSAEINPROGRESS：一个阻塞的 WINDOWS 套接口调用正在运行中。

WSAEINVAL：该套接口未用 bind()进行捆绑，或已被连接。
WSAEISCONN：套接口已被连接。
WSAEMFILE：无可用文件描述字。
WSAENOBUFS：无可用缓冲区空间。
WSAENOTSOCK：描述字不是一个套接口。
WSAEOPNOTSUPP：该套接口不正常 listen()调用。

参见：
accept(), connet(), socket().

5.1.14 ntohl()

简述：
将一个无符号长整形数从网络字节顺序转换为主机字节顺序。

```
#include <winsock.h>

u_long PASCAL FAR ntohl( u_long netlong);
```

netlong：一个以网络字节顺序表达的 32 位数。

注释：
本函数将一个 32 位数由网络字节顺序转换为主机字节顺序。

返回值：
ntohl()返回一个以主机字节顺序表达的数。

参见：
htonl(), htons(), ntohs().

5.1.15 ntohs()

简述：
将一个无符号短整形数从网络字节顺序转换为主机字节顺序。

```
#include <winsock.h>

u_short PASCAL FAR ntohs( u_short netshort);
```

netshort：一个以网络字节顺序表达的 16 位数。

注释：

本函数将一个 16 位数由网络字节顺序转换为主机字节顺序。

返回值：

ntohs()返回一个以主机字节顺序表达的数。

参见：

htonl(), htons(), ntohl().

5.1.16 recv()

简述：

从一个套接口接收数据。

```
#include <winsock.h>

int PASCAL FAR recv( SOCKET s, char FAR* buf, int len, int flags);
```

s：一个标识已连接套接口的描述字。
buf：用于接收数据的缓冲区。
len：缓冲区长度。
flags：指定调用方式。

注释：

本函数用于已连接的数据报或流式套接口 s 进行数据的接收。

对 SOCK_STREAM 类型的套接口来说，本函数将返回所有可用的信息，最大可达缓冲区的大小。如果套接口被设置为线内接收带外数据（选项为 SO_OOBINLINE），且有带外数据未读入，则返回带外数据。应用程序可通过调用 ioctlsocket()的 SOCATMARK 命令来确定是否有带外数据待读入。

对于数据报类套接口，队列中第一个数据报中的数据被解包，但最多不超过缓冲区的大小。如果数据报大于缓冲区，那么缓冲区中只有数据报的前面部分，其

他的数据都丢失了，并且 recv()函数返回 WSAEMSGSIZE 错误。如果没有数据待读，那么除非是非阻塞模式，不然的话套接口将一直等待数据的到来，此时将返回 SOCKET_ERROR 错误，错误代码是 WSAEWOULDBLOCK。用 select()或 WSAAsynSelect()可以获知何时数据到达。

如果套接口为 SOCK_STREAM 类型，并且远端‘ 优雅 ’地中止了连接，那么 recv()一个数据也不读取，立即返回。如果立即被强制中止，那么 recv()将以 WSAECONNRESET 错误失败返回。在套接口的所设选项之上，还可用标志位 flag 来影响函数的执行方式。也就是说，本函数的语义既取决于套接口选项，也取决于标志位参数。标志位可取下列值：

值　意义
MSG_PEEK 查看当前数据。数据将被复制到缓冲区中，但并不从输入队列中删除。
MSG_OOB 处理带外数据（参见 2.2.3 节具体讨论）。

返回值：

若无错误发生，recv()返回读入的字节数。如果连接已中止，返回 0。否则的话，返回 SOCKET_ERROR 错误，应用程序可通过 WSAGetLastError()获取相应错误代码。

错误代码：

WSANOTINITIALISED：在使用此 API 之前应首先成功地调用 WSAStartup()。
WSAENETDOWN：WINDOWS 套接口实现检测到网络子系统失效。
WSAENOTCONN：套接口未连接。
WSAEINTR：阻塞进程被 WSACancelBlockingCall()取消。
WSAEINPROGRESS：一个阻塞的 WINDOWS 套接口调用正在运行中。
WSAENOTSOCK：描述字不是一个套接口。
WSAEOPNOTSUPP：指定了 MSG_OOB，但套接口不是 SOCK_STREAM 类型的。
WSAESHUTDOWN：套接口已被关闭。当一个套接口以 0 或 2 的 how 参数调用 shutdown()关闭后，无法再用 recv()接收数据。
WSAEWOULDBLOCK：套接口标识为非阻塞模式，但接收操作会产生阻塞。
WSAEMSGSIZE：数据报太大无法全部装入缓冲区，故被剪切。
WSAEINVAL：套接口未用 bind()进行捆绑。
WSAECONNABORTED：由于超时或其他原因，虚电路失效。
WSAECONNRESET：远端强制中止了虚电路。

参见：

recvfrom(), read(), recv(), send(), select(), WSAAsyncSelect(),

socket().

5.1.17 recvfrom()

简述：

接收一个数据报并保存源地址。

```
#include <winsock.h>

int PASCAL FAR recvfrom( SOCKET s, char FAR* buf, int len, int flags,
struct sockaddr FAR* from, int FAR* fromlen);
```

s：标识一个已连接套接口的描述字。
buf：接收数据缓冲区。
len：缓冲区长度。
flags：调用操作方式。
from：（可选）指针，指向装有源地址的缓冲区。
fromlen：（可选）指针，指向 from 缓冲区长度值。

注释：

本函数由于从（已连接）套接口上接收数据，并捕获数据发送源的地址。

对于 SOCK_STREAM 类型的套接口，最多可接收缓冲区大小个数据。如果套接口被设置为线内接收带外数据（选项为 SO_OOBINLINE），且有带外数据未读入，则返回带外数据。应用程序可通过调用 ioctlsocket()的 SOCATMARK 命令来确定是否有带外数据待读入。对于 SOCK_STREAM 类型套接口，忽略 from 和 fromlen 参数。

对于数据报类套接口，队列中第一个数据报中的数据被解包，但最多不超过缓冲区的大小。如果数据报大于缓冲区，那么缓冲区中只有数据报的前面部分，其他的数据都丢失了，并且 recvfrom()函数返回 WSAEMSGSIZE 错误。

若 from 非零，且套接口为 SOCK_DGRAM 类型，则发送数据源的地址被复制到相应的 sockaddr 结构中。fromlen 所指向的值初始化时为这个结构的大小，当调用返回时按实际地址所占的空间进行修改。

如果没有数据待读，那么除非是非阻塞模式，不然的话套接口将一直等待数据的到来，此时将返回 SOCKET_ERROR 错误，错误代码是 WSAEWOULDBLOCK。用 select()或 WSAAsynSelect()可以获知何时数据到达。

如果套接口为 SOCK_STREAM 类型，并且远端“优雅”地中止了连接，那么 recvfrom()一个数据也不读取，立即返回。如果立即被强制中止，那么 recv()

将以 WSAECONNRESET 错误失败返回。

在套接口的所设选项之上，还可用标志位 flag 来影响函数的执行方式。也就是说，本函数的语义既取决于套接口选项，也取决于标志位参数。标志位可取下列值：

值 意义

MSG_PEEK 查看当前数据。数据将被复制到缓冲区中，但并不从输入队列中删除。

MSG_OOB 处理带外数据（参见 2.2.3 节具体讨论）。

返回值：

若无错误发生，recvfrom()返回读入的字节数。如果连接已中止，返回 0。否则的话，返回 SOCKET_ERROR 错误，应用程序可通过 WSAGetLastError()获取相应错误代码。

错误代码：

WSANOTINITIALISED：在使用此 API 之前应首先成功地调用 WSAStartup()。

WSAENETDOWN：WINDOWS 套接口实现检测到网络子系统失效。

WSAEFAULT：fromlen 参数非法；from 缓冲区大小无法装入端地址。

WSAEINTR：阻塞进程被 WSACancelBlockingCall()取消。

WSAEINPROGRESS：一个阻塞的 WINDOWS 套接口调用正在运行中。

WSAEINVAL：套接口未用 bind()进行捆绑。

WSAENOTCONN：套接口未连接（仅适用于 SOCK_STREAM 类型）。

WSAENOTSOCK：描述字不是一个套接口。

WSAEOPNOTSUPP：指定了 MSG_OOB，但套接口不是 SOCK_STREAM 类型的。

WSAESHUTDOWN：套接口已被关闭。当一个套接口以 0 或 2 的 how 参数调用 shutdown()关闭后，无法再用 recv()接收数据。

WSAEWOULDBLOCK：套接口标识为非阻塞模式，但接收操作会产生阻塞。

WSAEMSGSIZE：数据报太大无法全部装入缓冲区，故被剪切。

WSAECONNABORTED：由于超时或其他原因，虚电路失效。

WSAECONNRESET：远端强制中止了虚电路。

参见：

recv(), send(), socket(), WSAAsyncSelect().

5.1.18 select()

简述：

确定一个或多个套接口的状态，如需要则等待。

```
#include <winsock.h>

int PASCAL FAR select( int nfds, fd_set FAR* readfds,
fd_set FAR* writefds, fd_set FAR* exceptfds,
const struct timeval FAR* timeout);
```

nfds：本参数忽略，仅起到兼容作用。
readfds：（可选）指针，指向一组等待可读性检查的套接口。
writefds：（可选）指针，指向一组等待可写性检查的套接口。
exceptfds：（可选）指针，指向一组等待错误检查的套接口。
timeout：select()最多等待时间，对阻塞操作则为 NULL。

注释：

本函数用于确定一个或多个套接口的状态。对每一个套接口，调用者可查询它的可读性、可写性及错误状态信息。用 fd_set 结构来表示一组等待检查的套接口。在调用返回时,这个结构存有满足一定条件的套接口组的子集,并且 select()返回满足条件的套接口的数目。有一组宏可用于对 fd_set 的操作，这些宏与 Berkeley Unix 软件中的兼容，但内部的表达是完全不同的。

readfds 参数标识等待可读性检查的套接口。如果该套接口正处于监听 listen()状态，则若有连接请求到达，该套接口便被标识为可读，这样一个 accept()调用保证可以无阻塞完成。对其他套接口而言，可读性意味着有排队数据供读取。或者对于 SOCK_STREAM 类型套接口来说，相对于该套接口的虚套接口已关闭，于是 recv()或 recvfrom()操作均能无阻塞完成。如果虚电路被“优雅地”中止，则 recv()不读取数据立即返回；如果虚电路被强制复位，则 recv()将以 WSAECONNRESET 错误立即返回。如果 SO_OOBINLINE 选项被设置，则将检查带外数据是否存在（参见 setsockopt()）。

writefds 参数标识等待可写性检查的套接口。如果一个套接口正在 connect()连接（非阻塞），可写性意味着连接顺利建立。如果套接口并未处于 connect()调用中，可写性意味着 send()和 sendto()调用将无阻塞完成。〔但并未指出这个保证在多长时间内有效，特别是在多线程环境中〕。

exceptfds 参数标识等待带外数据存在性或意味错误条件检查的套接口。请注意如果设置了 SO_OOBINLINE 选项为假 FALSE，则只能用这种方法来检查带外数据的存在与否。对于 SO_STREAM 类型套接口，远端造成的连接中止和 KEEPALIVE

错误都将被作为意味出错。如果套接口正在进行连接 connect()（非阻塞方式），则连接试图的失败将会表现在 exceptfds 参数中。

如果对 readfds、writefds 或 exceptfds 中任一个组类不感兴趣，可将它置为空 NULL。

在 winsock.h 头文件中共定义了四个宏来操作描述字集。FD_SETSIZE 变量用于确定一个集合中最多有多少描述字（FD_SETSIZE 缺省值为 64，可在包含 winsock.h 前用#define FD_SETSIZE 来改变该值）。对于内部表示，fd_set 被表示成一个套接口的队列，最后一个有效元素的后续元素为 INVAL_SOCKET。宏为：

FD_CLR(s,*set)：从集合 set 中删除描述字 s。

FD_ISSET(s,*set)：若 s 为集合中一员，非零；否则为零。

FD_SET(s,*set)：向集合添加描述字 s。

FD_ZERO(*set)：将 set 初始化为空集 NULL。

timeout参数控制select()完成的时间。若timeout参数为空指针,则select()将一直阻塞到有一个描述字满足条件。否则的话，timeout 指向一个 timeval 结构，其中指定了 select()调用在返回前等待多长时间。如果 timeval 为{0,0}，则 select()立即返回，这可用于探询所选套接口的状态。如果处于这种状态，则 select()调用可认为是非阻塞的，且一切适用于非阻塞调用的假设都适用于它。举例来说，阻塞钩子函数不应被调用，且 WINDOWS 套接口实现不应 yield。

返回值：

select()调用返回处于就绪状态并且已经包含在 fd_set 结构中的描述字总数；如果超时则返回 0；否则的话，返回 SOCKET_ERROR 错误，应用程序可通过 WSAGetLastError()获取相应错误代码。

错误代码：

WSANOTINITIALISED：在使用此 API 之前应首先成功地调用 WSAStartup()。

WSAENETDOWN：WINDOWS 套接口实现检测到网络子系统失效。

WSAEINVAL：超时时间值非法。

WSAEINTR：通过一个 WSACancelBlockingCall()来取消一个（阻塞的）调用。

WSAEINPROGRESS：一个阻塞的 WINDOWS 套接口调用正在运行中。

WSAENOTSOCK：描述字集合中包含有非套接口的元素。

参见：

WSAAsyncSelect(), accept(), connect(), recv(), recvfrom(), send().

5.1.19 send()

简述：

向一个已连接的套接口发送数据。

```
#include <winsock.h>

int PASCAL FAR send( SOCKET s, const char FAR* buf, int len, int flags);
```

s：一个用于标识已连接套接口的描述字。
buf：包含待发送数据的缓冲区。
len：缓冲区中数据的长度。
flags：调用执行方式。

注释：

send()适用于已连接的数据报或流式套接口发送数据。对于数据报类套接口，必需注意发送数据长度不应超过通讯子网的 IP 包最大长度。IP 包最大长度在 WSAStartup()调用返回的 WSAData 的 iMaxUdpDg 元素中。如果数据太长无法自动通过下层协议，则返回 WSAEMSGSIZE 错误，数据不会被发送。

请注意成功地完成 send()调用并不意味着数据传送到达。

如果传送系统的缓冲区空间不够保存需传送的数据，除非套接口处于非阻塞 I/O 方式，否则 send()将阻塞。对于非阻塞 SOCK_STREAM 类型的套接口，实际写的数据数目可能在 1 到所需大小之间，其值取决于本地和远端主机的缓冲区大小。可用 select()调用来确定何时能够进一步发送数据。

在相关套接口的选项之上，还可通过标志位 flag 来影响函数的执行方式。也就是说，本函数的语义既取决于套接口的选项也取决于标志位。后者由以下一些值组成：

值	意义
MSG_DONTROUTE	指明数据不选径。一个 WINDOWS 套接口供应商可以忽略此标志；参见 2.4 节中关于 SO_DONTROUTE 的讨论。
MSG_OOB	发送带外数据（仅适用于 SO_STREAM；参见 2.2.3 节）。

返回值：

若无错误发生，send()返回所发送数据的总数（请注意这个数字可能小于 len 中所规定的大小）。否则的话，返回 SOCKET_ERROR 错误，应用程序可通过 WSAGetLastError()获取相应错误代码。

错误代码：

WSANOTINITIALISED：在使用此 API 之前应首先成功地调用 WSAStartup()。
WSAENETDOWN：WINDOWS 套接口实现检测到网络子系统失效。
WSAEACESS：要求地址为广播地址，但相关标志未能正确设置。
WSAEINTR：通过一个 WSACancelBlockingCall()来取消一个（阻塞的）调用。
WSAEINPROGRESS：一个阻塞的 WINDOWS 套接口调用正在运行中。
WSAEFAULT：buf 参数不在用户地址空间中的有效位置。
WSAENETRESET：由于 WINDOWS 套接口实现放弃了连接，故该连接必需被复位。
WSAENOBUFS：WINDOWS 套接口实现报告一个缓冲区死锁。
WSAENOTCONN：套接口未被连接。
WSAENOTSOCK：描述字不是一个套接口。
WSAEOPNOTSUPP：已设置了 MSG_OOB，但套接口非 SOCK_STREAM 类型。
WSAESHUTDOWN：套接口已被关闭。一个套接口以 1 或 2 的 how 参数调用 shutdown()关闭后，无法再用 sned()函数。
WSAEWOULDBLOCK：
WSAEMSGSIZE：套接口为 SOCK_DGRAM 类型，且数据报大于 WINDOWS 套接口实现所支持的最大值。
WSAEINVAL：套接口未用 bind()捆绑。
WSAECONNABORTED：由于超时或其他原因引起虚电路的中断。
WSAECONNRESET：虚电路被远端复位。

参见：
recv(), recvfrom(), socket(), sendto(), WSAStartup().

5.1.20 sendto()

简述：
向一指定目的地发送数据。

```
#include <winsock.h>

int PASCAL FAR sendto( SOCKET s, const char FAR* buf, int len, int flags,
const struct sockaddr FAR* to, int tolen);
```

s：一个标识套接口的描述字。
buf：包含待发送数据的缓冲区。
len：buf 缓冲区中数据的长度。
flags：调用方式标志位。

to：（可选）指针，指向目的套接口的地址。
tolen：to 所指地址的长度。

注释：
sendto()适用于已连接的数据报或流式套接口发送数据。对于数据报类套接口，必需注意发送数据长度不应超过通讯子网的 IP 包最大长度。IP 包最大长度在 WSAStartup()调用返回的 WSAData 的 iMaxUdpDg 元素中。如果数据太长无法自动通过下层协议，则返回 WSAEMSGSIZE 错误，数据不会被发送。
请注意成功地完成 sendto()调用并不意味着数据传送到达。
sendto()函数主要用于 SOCK_DGRAM 类型套接口向 to 参数指定端的套接口发送数据报。对于 SOCK_STREAM 类型套接口，to 和 tolen 参数被忽略；这种情况下 sendto()等价于 send()。
为了发送广播数据（仅适用于 SOCK_DGRAM），in 参数所含地址应该把特定的 IP 地址 INADDR_BROADCAST（winsock.h 中有定义）和终端地址结合起来构造。通常建议一个广播数据报的大小不要大到以致产生碎片，也就是说数据报的数据部分（包括头）不超过 512 字节。
如果传送系统的缓冲区空间不够保存需传送的数据，除非套接口处于非阻塞 I/O 方式，否则 sendto()将阻塞。对于非阻塞 SOCK_STREAM 类型的套接口，实际写的数据数目可能在 1 到所需大小之间，其值取决于本地和远端主机的缓冲区大小。可用 select()调用来确定何时能够进一步发送数据。
在相关套接口的选项之上，还可通过标志位 flag 来影响函数的执行方式。也就是说，本函数的语义既取决于套接口的选项也取决于标志位。后者由以下一些值组成：

值　　意义
MSG_DONTROUTE　指明数据不选径。一个 WINDOWS 套接口供应商可以忽略此标志；参见 2.4 节中关于 SO_DONTROUTE 的讨论。
MSG_OOB　　发送带外数据（仅适用于 SO_STREAM；参见 2.2.3 节）。

返回值：
若无错误发生，send()返回所发送数据的总数（请注意这个数字可能小于 len 中所规定的大小）。否则的话，返回 SOCKET_ERROR 错误，应用程序可通过 WSAGetLastError()获取相应错误代码。

错误代码：
WSANOTINITIALISED：在使用此 API 之前应首先成功地调用 WSAStartup()。
WSAENETDOWN：WINDOWS 套接口实现检测到网络子系统失效。

WSAEACESS：要求地址为广播地址，但相关标志未能正确设置。
WSAEINTR：通过一个 WSACancelBlockingCall()来取消一个（阻塞的）调用。
WSAEINPROGRESS：一个阻塞的 WINDOWS 套接口调用正在运行中。
WSAEFAULT：buf 或 to 参数不是用户地址空间的一部分，或 to 参数太小（小于 sockaddr 结构大小）。
WSAENETRESET：由于 WINDOWS 套接口实现放弃了连接，故该连接必需被复位。
WSAENOBUFS：WINDOWS 套接口实现报告一个缓冲区死锁。
WSAENOTCONN：套接口未被连接。
WSAENOTSOCK：描述字不是一个套接口。
WSAEOPNOTSUPP：已设置了 MSG_OOB，但套接口非 SOCK_STREAM 类型。
WSAESHUTDOWN：套接口已被关闭。一个套接口以 1 或 2 的 how 参数调用 shutdown()关闭后，无法再用 sned()函数。
WSAEWOULDBLOCK：套接口被标志为非阻塞， 但该调用会产生阻塞。
WSAEMSGSIZE：套接口为 SOCK_DGRAM 类型，且数据报大于 WINDOWS 套接口实现所支持的最大值。
WSAECONNABORTED：由于超时或其他原因引起虚电路的中断。
WSAECONNRESET：虚电路被远端复位。
WSAEADDRNOTAVAIL：所指地址无法从本地主机获得。
WSAEAFNOSUPPORT：所指定地址族中地址无法与本套接口一切使用。
WSAEDESADDRREQ：需要目的地址。
WSAENETUNREACH：当前无法从本主机联上网络。

参见：
recv(), recvfrom(), socket(), send(), WSAStartup().

5.1.21 setsockopt()

简述：
设置套接口的选项。

```
#include <winsock.h>

int PASCAL FAR setsockopt( SOCKET s, int level, int optname,
const char FAR* optval, int optlen);
```

s：标识一个套接口的描述字。
level：选项定义的层次；目前仅支持 SOL_SOCKET 和 IPPROTO_TCP 层次。

optname：需设置的选项。
optval：指针，指向存放选项值的缓冲区。
optlen：optval 缓冲区的长度。

注释：

setsockopt()函数用于任意类型、任意状态套接口的设置选项值。尽管在不同协议层上存在选项，但本函数仅定义了最高的“套接口”层次上的选项。选项影响套接口的操作，诸如加急数据是否在普通数据流中接收，广播数据是否可以从套接口发送等等。

有两种套接口的选项：一种是布尔型选项，允许或禁止一种特性；另一种是整形或结构选项。允许一个布尔型选项，则将 optval 指向非零整形数；禁止一个选项 optval 指向一个等于零的整形数。对于布尔型选项，optlen 应等于 sizeof(int) ;对其他选项,optval 指向包含所需选项的整形数或结构,而 optlen 则为整形数或结构的长度。SO_LINGER 选项用于控制下述情况的行动：套接口上有排队的待发送数据，且 closesocket()调用已执行。参见 closesocket()函数中关于 SO_LINGER 选项对 closesocket()语义的影响。应用程序通过创建一个 linger 结构来设置相应的操作特性：

```
struct linger {
  int l_onoff;
  int l_linger;
};
```

为了允许 SO_LINGER，应用程序应将 l_onoff 设为非零，将 l_linger 设为零或需要的超时值（以秒为单位），然后调用 setsockopt()。为了允许 SO_DONTLINGER（亦即禁止 SO_LINGER），l_onoff 应设为零，然后调用 setsockopt()。

缺省条件下，一个套接口不能与一个已在使用中的本地地址捆绑（参见 bind()）。但有时会需要“重用”地址。因为每一个连接都由本地地址和远端地址的组合唯一确定，所以只要远端地址不同，两个套接口与一个地址捆绑并无大碍。为了通知 WINDOWS 套接口实现不要因为一个地址已被一个套接口使用就不让它与另一个套接口捆绑，应用程序可在 bind()调用前先设置 SO_REUSEADDR 选项。请注意仅在 bind()调用时该选项才被解释；故此无需（但也无害）将一个不会共用地址的套接口设置该选项，或者在 bind()对这个或其他套接口无影响情况下设置或清除这一选项。

一个应用程序可以通过打开 SO_KEEPALIVE 选项，使得 WINDOWS 套接口实现在 TCP 连接情况下允许使用“保持活动”包。一个 WINDOWS 套接口实现并不是必需支持‘保持活动”,但是如果支持的话,具体的语义将与实现有关,应遵守 RFC1122“Internet 主机要求－通讯层”中第 4.2.3.6 节的规范。如果有关连接由于“保持活动”而失效，则进行中的任何对该套接口的调用都将以 WSAENETRESET 错误

返回，后续的任何调用将以 WSAENOTCONN 错误返回。

TCP_NODELAY 选项禁止 Nagle 算法。Nagle 算法通过将未确认的数据存入缓冲区直到蓄足一个包一起发送的方法，来减少主机发送的零碎小数据包的数目。但对于某些应用来说，这种算法将降低系统性能。所以 TCP_NODELAY 可用来将此算法关闭。应用程序编写者只有在确切了解它的效果并确实需要的情况下，才设置 TCP_NODELAY 选项，因为设置后对网络性能有明显的负面影响。TCP_NODELAY 是唯一使用 IPPROTO_TCP 层的选项，其他所有选项都使用 SOL_SOCKET 层。

如果设置了 SO_DEBUG 选项，WINDOWS 套接口供应商被鼓励（但不是必需）提供输出相应的调试信息。但产生调试信息的机制以及调试信息的形式已超出本规范的讨论范围。

setsockopt()支持下列选项。其中“类型”表明 optval 所指数据的类型。

选项	类型	意义
SO_BROADCAST	BOOL	允许套接口传送广播信息。
SO_DEBUG	BOOL	记录调试信息。
SO_DONTLINER	BOOL	不要因为数据未发送就阻塞关闭操作。设置本选项相当于将 SO_LINGER 的 l_onoff 元素置为零。
SO_DONTROUTE	BOOL	禁止选径；直接传送。
SO_KEEPALIVE	BOOL	发送“保持活动”包。
SO_LINGER	struct linger FAR*	如关闭时有未发送数据，则逗留。
SO_OOBINLINE	BOOL	在常规数据流中接收带外数据。
SO_RCVBUF	int	为接收确定缓冲区大小。
SO_REUSEADDR	BOOL	允许套接口和一个已在使用中的地址捆绑（参见 bind()）。
SO_SNDBUF	int	指定发送缓冲区大小。
TCP_NODELAY	BOOL	禁止发送合并的 Nagle 算法。

setsockopt()不支持的 BSD 选项有：

选项名	类型	意义
SO_ACCEPTCONN	BOOL	套接口在监听。
SO_ERROR	int	获取错误状态并清除。
SO_RCVLOWAT	int	接收低级水印。
SO_RCVTIMEO	int	接收超时。
SO_SNDLOWAT	int	发送低级水印。
SO_SNDTIMEO	int	发送超时。
SO_TYPE	int	套接口类型。
IP_OPTIONS		在 IP 头中设置选项。

返回值：

若无错误发生，setsockopt()返回 0。否则的话，返回 SOCKET_ERROR 错误，应用程序可通过 WSAGetLastError()获取相应错误代码。

错误代码：

WSANOTINITIALISED：在使用此 API 之前应首先成功地调用 WSAStartup()。

WSAENETDOWN：WINDOWS 套接口实现检测到网络子系统失效。

WSAEFAULT：optval 不是进程地址空间中的一个有效部分。

WSAEINPROGRESS：一个阻塞的 WINDOWS 套接口调用正在运行中。

WSAEINVAL：level 值非法，或 optval 中的信息非法。

WSAENETRESET：当 SO_KEEPALIVE 设置后连接超时。

WSAENOPROTOOPT：未知或不支持选项。其中，SOCK_STREAM 类型的套接口不支持 SO_BROADCAST 选项，SOCK_DGRAM 类型的套接口不支持 SO_DONTLINGER 、SO_KEEPALIVE、SO_LINGER 和 SO_OOBINLINE 选项。

WSAENOTCONN：当设置 SO_KEEPALIVE 后连接被复位。

WSAENOTSOCK：描述字不是一个套接口。

参见：

bind(), getsockopt(), ioctlsocket(), socket(), WSAAsyncSelect().

5.1.22 shutdown()

简述：

禁止在一个套接口上进行数据的接收与发送。

```
#include <winsock.h>

int PASCAL FAR shutdown( SOCKET s, int how);
```

s：用于标识一个套接口的描述字。

how：标志，用于描述禁止哪些操作。

注释：

shutdown()函数用于任何类型的套接口禁止接收、禁止发送或禁止收发。

如果 how 参数为 0，则该套接口上的后续接收操作将被禁止。这对于低层协议无影响。对于 TCP 协议，TCP 窗口不改变并接收前来的数据（但不确认）直至窗口满。对于 UDP 协议，接收并排队前来的数据。任何情况下都不会产生 ICMP 错误包。

若 how 为 1，则禁止后续发送操作。对于 TCP，将发送 FIN。
若 how 为 2，则同时禁止收和发。
请注意 shutdown()函数并不关闭套接口，且套接口所占有的资源将被一直保持到 closesocket()调用。

评注：
无论 SO_LINGER 设置与否，shutdown()函数不会阻塞。
一个应用程序不应依赖于重用一个已被 shutdown()禁止的套接口。特别地，一个 WINDOWS 套接口实现不必支持在这样的套接口上使用 connect()调用。

返回值：
如果没有错误发生，shutdown()返回 0。否则的话，返回 SOCKET_ERROR 错误，应用程序可通过 WSAGetLastError()获取相应错误代码。

错误代码：
WSANOTINITIALISED：在使用此 API 之前应首先成功地调用 WSAStartup()。
WSAENETDOWN：WINDOWS 套接口实现检测到网络子系统失效。
WSAEINVAL：how 参数非法。
WSAEINPROGRESS：一个阻塞的 WINDOWS 套接口调用正在运行中。
WSAENOTCONN：套接口未连接（仅适用于 SOCK_STREAM 类型套接口）。
WSAENOTSOCK：描述字不是一个套接口。

参见：
connect(), socket().

5.1.23 socket()

简述：
创建一个套接口。

```
#include <winsock.h>

SOCKET PASCAL FAR socket( int af, int type, int protocol);
```

af：一个地址描述。目前仅支持 PF_INET 格式，也就是说 ARPA Internet 地址格式。
type：新套接口的类型描述。

protocol：套接口所用的协议。如调用者不想指定，可用 0。

注释：

socket()函数用于根据指定的地址族、数据类型和协议来分配一个套接口的描述字及其所用的资源。如果协议 protocol 未指定（等于 0），则使用缺省的连接方式。

对于使用一给定地址族的某一特定套接口，只支持一种协议。但地址族可设为 AF_UNSPEC（未指定），这样的话协议参数就要指定了。协议号特定于进行通讯的“通讯域”。支持下述类型描述：

类型	解释
SOCK_STREAM	提供有序的、可靠的、双向的和基于连接的字节流，使用带外数据传送机制，为 Internet 地址族使用 TCP。
SOCK_DGRAM	支持无连接的、不可靠的和使用固定大小（通常很小）缓冲区的数据报服务，为 Internet 地址族使用 UDP。

SOCK_STREAM 类型的套接口为全双向的字节流。对于流类套接口，在接收或发送数据前必需处于已连接状态。用 connect()调用建立与另一套接口的连接，连接成功后，即可用 send()和 recv()传送数据。当会话结束后，调用 closesocket()。带外数据根据规定用 send()和 recv()来接收。

实现 SOCK_STREAM 类型套接口的通讯协议保证数据不会丢失也不会重复。如果终端协议有缓冲区空间，且数据不能在一定时间成功发送，则认为连接中断，其后续的调用也将以 WSAETIMEOUT 错误返回。

SOCK_DGRAM 类型套接口允许使用 sendto()和 recvfrom()从任意端口发送或接收数据报。如果这样一个套接口用 connect()与一个指定端口连接，则可用 send()和 recv()与该端口进行数据报的发送与接收。

返回值：

若无错误发生，socket()返回引用新套接口的描述字。否则的话，返回 SOCKET_ERROR 错误，应用程序可通过 WSAGetLastError()获取相应错误代码。

错误代码：

WSANOTINITIALISED：在使用此 API 之前应首先成功地调用 WSAStartup()。

WSAENETDOWN：WINDOWS 套接口实现检测到网络子系统失效。

WSAEAFNOSUPPORT：不支持指定的地址族。

WSAEINPROGRESS：一个阻塞的 WINDOWS 套接口调用正在运行中。

WSAEMFILE：无可用文件描述字。

WSAENOBUFS：无可用缓冲区，无法创建套接口。

WSAEPROTONOSUPPORT：不支持指定的协议。
WSAEPROTOTYPE：指定的协议不适用于本套接口。
WSAESOCKTNOSUPPORT：本地址族中不支持该类型套接口。

参见：
accept(), bind(), connect(), getsockname(), getsockopt(), setsockopt(), listen(), recv(), recvfrom(), select(), send(), sendto(), shutdown(), ioctlsocket().4.2 数据库例程

5.2 数据库函数

5.2.1 gethostbyaddr()

简述：
返回对应于给定地址的主机信息。

```
#include <winsock.h>

struct hostent FAR *PASCAL FAR gethostbyaddr(const char
FAR * addr, int len, int type);
```

addr：指向网络字节顺序地址的指针。

len： 地址的长度，在 PF_INET 类型地址中为 4。

type：地址类型，应为 PF_INET。

注释：
gethostbyaddr()返回对应于给定地址的包含主机名字和地址信息的 hostent 结构指针。结构的声明如下：

```
struct hostent {
    char FAR *       h_name;
    char FAR * FAR *  h_aliases;
    short            h_addrtype;
    short            h_length;
    char FAR * FAR *  h_addr_list;
```

};

结构的成员有：

成员	用途
h_name	正规的主机名字（PC）。
h_aliases	一个以空指针结尾的可选主机名队列。
h_addrtype	返回地址的类型，对于 Windows Sockets，这个域总是 PF_INET。
h_legnth	每个地址的长度（字节数），对应于 PF_INET 这个域应该为 4。
h_addr_list	应该以空指针结尾的主机地址的列表,返回的地址是以网络顺序排列的

为了保证其他旧的软件的兼容性，h_addr_list[0]被定义为宏 h_addr。

返回的指针指向一个由 Windows Sockets 实现分配的结构。应用程序不应该试图修改这个结构或者释放它的任何部分。此外，每一线程仅有一份这个结构的拷贝，所以应用程序应该在发出其他 Windows Scokets API 调用前，把自己所需的信息拷贝下来。

返回值：

如果没有错误发生，gethostbyaddr()返回如上所述的一个指向 hostent 结构的指针，否则，返回一个空指针。应用程序可以通过 WSAGetLastError()来得到一个特定的错误代码。

错误代码：

WSANOTINTIALISED	在应用这个 API 前，必须成功地调用 WSAStartup()。
WSAENTDOWN	Windows Sockets 实现检测到了网络子系统的错误。
WSAHOST_NOT_FOUND	没有找到授权应答主机。
WSATRY_AGAIN	没有找到非授权主机，或者 SERVERFAIL。
WSANO_RECOVERY	无法恢复的错误，FORMERR，REFUSED，NOTIMP。
WSANO_DATA	有效的名字，但没有关于请求类型的数据记录。
WSAEINPROGRESS	一个阻塞的 Windows Sockets 操作正在进行。
WSAEINTR	阻塞调用被 WSACancelBlockingCall()取消了.

参见: WSAAsyncGetHostByAddr(), gethostbyname()

5.2.2 gethostbyname()

简述：

返回对应于给定主机名的主机信息。

```
#include <winsock.h>

struct hostent FAR *PASCAL FAR gethostbyname(const char
FAR * addr);
```

name：指向主机名的指针。

注释：

gethostbyname()返回对应于给定主机名的包含主机名字和地址信息的 hostent 结构指针。结构的声明与 gethostaddr()中一致。

返回的指针指向一个由 Windows Sockets 实现分配的结构。应用程序不应该试图修改这个结构或者释放它的任何部分。此外，每一线程仅有一份这个结构的拷贝，所以应用程序应该在发出其他 Windows Scokets API 调用前，把自己所需的信息拷贝下来。

gethostbyname()实现没有必要识别传送给它的 IP 地址串。对于这样的请求，应该把 IP 地址串当作一个未知主机名同样处理。如果应用程序有 IP 地址串需要处理，它应该使用 inet_addr()函数把地址串转换为 IP 地址，然后调用 gethostbyaddr()来得到 hostent 结构。

返回值：

如果没有错误发生，gethostbyname()返回如上所述的一个指向 hostent 结构的指针，否则，返回一个空指针。应用程序可以通过 WSAGetLastError()来得到一个特定的错误代码。

错误代码：

WSANOTINTIALISED	在应用这个 API 前，必须成功地调用 WSAStartup()。
WSAENTDOWN	Windows Sockets 实现检测到了网络子系统的错误。
WSAHOST_NOT_FOUND	没有找到授权应答主机。
WSATRY_AGAIN	没有找到非授权主机，或者 SERVERFAIL。
WSANO_RECOVERY	无法恢复的错误，FORMERR，REFUSED，NOTIMP。
WSANO_DATA	有效的名字，但没有关于请求类型的数据记录。

WSAEINPROGRESS　　一个阻塞的 Windows Sockets 操作正在进行。
WSAEINTR　　阻塞调用被 WSACancelBlockingCall()取消了.

参见: WSAAsyncGetHostByName(), gethostbyaddr()

5.2.3 gethostname()

简述：

返回本地主机的标准主机名。

```
#include <winsock.h>

int PASCAL FAR gethostname(char FAR *name, int namelen);
```

name：　一个指向将要存放主机名的缓冲区指针。

namelen：缓冲区的长度。

注释：

该函数把本地主机名存放入由 name 参数指定的缓冲区中。返回的主机名是一个以 NULL 结束的字符串。主机名的形式取决于 Windows Sockets 实现－它可能是一个简单的主机名，或者是一个域名。然而，返回的名字必定可以在 gethostbyname()和 WSAAsyncGetHostByName()中使用。

返回值：

如果没有错误发生，gethostname()返回 0。否则它返回 SOCKET_ERROR。应用程序可以通过 WSAGetLastError()来得到一个特定的错误代码。

错误代码：
WSAEFAULT　　名字长度参数太小。
WSANOTINTIALISED　在应用这个 API 前，必须成功地调用 WSAStartup()。
WSAENTDOWN　　Windows Sockets 实现检测到了网络子系统的错误。
WSAEINPROGRESS　　一个阻塞的 Windows Sockets 操作正在进行。

参见：gethostbyname(), WSAAsyncGetHostByName()

5.2.4 getprotobyname()

简述：

返回对应于给定协议名的相关协议信息。

```
#include <winsock.h>

struct protoent FAR * PASCAL FAR getprotobyname(const char
FAR * name);
```

name：一个指向协议名的指针。

注释：

getprotobyname()返回对应于给定协议名的包含名字和协议号的 protoent 结构指针。结构的声明如下：

```
struct protoent {
    char FAR *        p_name;
    char Far * far *  p_aliases;
    short             p_proto;
};
```

结构的成员有：

成员	用途
p_name	正规的协议名。
p_aliases	一个以空指针结尾的可选协议名队列。
p_proto	以主机字节顺序排列的协议号

返回的指针指向一个由 Windows Sockets 实现分配的结构。应用程序不应该试图修改这个结构或者释放它的任何部分。此外，每一线程仅有一份这个结构的拷贝，所以应用程序应该在发出其他 Windows Scokets API 调用前，把自己所需的信息拷贝下来。

返回值：

如果没有错误发生，getprotobyname()返回如上所述的一个指向 protoent 结构的指针，否则，返回一个空指针。应用程序可以通过 WSAGetLastError()来得到一个特定的错误代码。

错误代码：

WSANOTINTIALISED	在应用这个 API 前，必须成功地调用 WSAStartup()。
WSAENTDOWN	Windows Sockets 实现检测到了网络子系统的错误。
WSANO_RECOVERY	无法恢复的错误，FORMERR，REFUSED，NOTIMP。
WSANO_DATA	有效的名字，但没有关于请求类型的数据记录。
WSAEINPROGRESS	一个阻塞的 Windows Sockets 操作正在进行。
WSAEINTR	阻塞调用被 WSACancelBlockingCall()取消了.

参见: WSAAsyncGetProtoByName(), getprotobynumber()

5.2.5 getprotobynumber()

简述：

返回对应于给定协议号的相关协议信息。

```
#include <winsock.h>

struct protoent FAR * PASCAL FAR getprotobynumber(int number);
```

number：一个以主机顺序排列的协议号。

注释：

getprotobynumber()返回对应于给定协议名的包含名字和协议号的 protoent 结构指针。结构的声明与 getprotobyname 中的一致。

返回的指针指向一个由 Windows Sockets 实现分配的结构。应用程序不应该试图修改这个结构或者释放它的任何部分。此外，每一线程仅有一份这个结构的拷贝，所以应用程序应该在发出其他 Windows Scokets API 调用前，把自己所需的信息拷贝下来。

返回值：

如果没有错误发生,getprotobynumber()返回如上所述的一个指向 protoent 结构的指针，否则，返回一个空指针。应用程序可以通过 WSAGetLastError()来得到一个特定的错误代码。

错误代码：

WSANOTINTIALISED	在应用这个 API 前，必须成功地调用 WSAStartup()。
WSAENTDOWN	Windows Sockets 实现检测到了网络子系统的错误。

WSANO_RECOVERY	无法恢复的错误，FORMERR，REFUSED，NOTIMP。
WSANO_DATA	有效的名字，但没有关于请求类型的数据记录。
WSAEINPROGRESS	一个阻塞的 Windows Sockets 操作正在进行。
WSAEINTR	阻塞调用被 WSACancelBlockingCall()取消了.

参见: WSAAsyncGetProtoByNumber(), getprotobyname()

5.2.6 getservbyname()

简述：

返回对应于给定服务名和协议名的相关服务信息。

```
#include <windows.h>

struct servent FAR * PASCAL FAR getservbyname(const char
Far * name, const char FAR *proto);
```

name： 一个指向服务名的指针。

proto： 指向协议名的指针（可选）。如果这个指针为空，getservbyname()返回第一个 name 与 s_name 或者某一个 s_aliases 匹配的服务条目。否则 getservbyname()对 name 和 proto 都进行匹配。

注释：

getservbyname()返回与给定服务名对应的包含名字和服务号信息的 servent 结构指针。结构的声明如下：

```
struct servent {
    char FAR *      s_name;
    char Far * FAR * s_aliases;
    short           s_port;
    char FAR *      s_proto;
};
```

结构的成员有：

成员	用途
s_name	正规的服务名。
s_aliases	一个以空指针结尾的可选服务名队列。

s_port　　连接该服务时需要用到的端口号，返回的端口号是以网络字节顺序排列的。
s_proto　　连接该服务时用到的协议名。

返回的指针指向一个由 Windows Sockets 实现分配的结构。应用程序不应该试图修改这个结构或者释放它的任何部分。此外，每一线程仅有一份这个结构的拷贝，所以应用程序应该在发出其他 Windows Scokets API 调用前，把自己所需的信息拷贝下来。

返回值：

如果没有错误发生，getservbyname()返回如上所述的一个指向 servent 结构的指针，否则，返回一个空指针。应用程序可以通过 WSAGetLastError()来得到一个特定的错误代码。

错误代码：

WSANOTINTIALISED	在应用这个 API 前，必须成功地调用 WSAStartup()。
WSAENTDOWN	Windows Sockets 实现检测到了网络子系统的错误。
WSAHOST_NOT_FOUND	没有找到授权应答主机。
WSANO_DATA	有效的名字，但没有关于请求类型的数据记录。
WSAEINPROGRESS	一个阻塞的 Windows Sockets 操作正在进行。
WSAEINTR	阻塞调用被 WSACancelBlockingCall()取消了.

参见: WSAAsyncGetServByName(), getservbyport()

5.2.7 getservbyport()

简述：

返回对应于给定端口号和协议名的相关服务信息。

```
#include <windows.h>

struct servent FAR * PASCAL FAR getservbyport(int port,
const char FAR *proto);
```

port：　给定的端口号，以网络字节顺序排列。

proto：指向协议名的指针（可选）。如果这个指针为空，getservbyport()返回第一个 port 与 s_port 匹配的服务条目。否则 getservbyport()对 port 和

proto 都进行匹配。

注释：

getservbyport()返回与给定服务名对应的包含名字和服务号信息的 servent 结构指针。结构的声明与 getservbyname()中一致。

返回的指针指向一个由 Windows Sockets 实现分配的结构。应用程序不应该试图修改这个结构或者释放它的任何部分。此外，每一线程仅有一份这个结构的拷贝，所以应用程序应该在发出其他 Windows Scokets API 调用前，把自己所需的信息拷贝下来。

返回值：

如果没有错误发生，getservbyport()返回如上所述的一个指向 servent 结构的指针，否则，返回一个空指针。应用程序可以通过 WSAGetLastError()来得到一个特定的错误代码。

错误代码：

WSANOTINTIALISED	在应用这个 API 前，必须成功地调用 WSAStartup()。
WSAENTDOWN	Windows Sockets 实现检测到了网络子系统的错误。
WSAHOST_NOT_FOUND	没有找到授权应答主机。
WSANO_DATA	有效的名字，但没有关于请求类型的数据记录。
WSAEINPROGRESS	一个阻塞的 Windows Sockets 操作正在进行。
WSAEINTR	阻塞调用被 WSACancelBlockingCall()取消了.

参见: WSAAsyncGetServByPort(), getservbyname()

5.3 Windows 扩展函数

5.3.1 WSAAsyncGetHostByAddr()

简述:

获得对应于一个地址的主机信息.－异步版本.

```
#include <winsock.h>

HANDLE PASCAL FAR WSAAsyncGetHostByAddr ( HWND hWnd,
       unsigned int wMsg,  const char FAR * addr, int len, int
```

type, char FAR * buf, int buflen);

hWnd　　当异步请求完成时,应该接收消息的窗口句柄.
wMsg　　当异步请求完成时,将要接收的消息.
addr　　主机网络地址的指针.主机地址以网络字节次序存储.
len 地址长度.对于 PF_INET 来说必须为 4.
type　　地址类型,必须是 PF_INET.
buf 接收 hostent 数据的数据区指针.注意该数据区必须大于 hostent 结构的大小.这是因为不仅 Windows Sockets 实现要用该数据区域容纳 hostent 结构,hostent 结构的成员引用的所有数据也要在该区域内.建议用户提供一个 MAXGETHOSTSTRUCT 字节大小的缓冲区.
buflen　上述数据区的大小.

注释:

本函数是 gethostbyaddr()的异步版本,是用来获取对应于一个网络地址的主机名和地址信息.Windows Sockets 的实现启动该操作后立刻返回调用方,并传回一个异步任务句柄,应用程序可以用它来标识该操作.当操作完成时,结果(若有的话)将会拷贝到调用方提供的缓冲区,同时向应用程序的窗口发一条消息.

当异步操作完成时,应用程序的窗口 hWnd 接收到消息 wMsg. wParam 参数包含了初次函数调用时返回的异步任务句柄.lParam 的高 16 位包含着错误代码.该代码可以是 winsock.h 中定义的任何错误.错误代码为 0 说明异步操作成功.在成功完成的情况下,提供给初始函数调用的缓冲区中包含了一个 hostent 结构.为存取该结构中的元素,初始的缓冲区指针应置为 hostent 结构的指针,并一如平常地存取.

注意若错误代码为 WSAENOBUFS,它说明在初始调用时由 buflen 指出的缓冲区大小对于容纳所有的结果信息来说太小了.在这种情况下,lParam 的低 16 位含有提供所有信息所需的缓冲区大小数值.如果应用程序认为获取的数据不够,它就可以在设置了足够容纳所需信息的缓冲区后,重新调用 WSAAsyncGetHostByAddr().(也就是大于 lParam 低 16 位提供的大小.)

错误代码和缓冲区大小应使用 WSAGETASYNCERROR 和 WSAGETASYNCBUFLEN 宏从 lParam 中取出.两个宏定义如下:

```
#define WSAGETASYNCERROR(lParam)          HIWORD(lParam)
#define WSAGETASYNCBUFLEN(lParam)         LOWORD(lParam)
```

使用这些宏可最大地提高应用程序源代码的可移植性.

返回值:

返回值指出异步操作是否成功地初启.注意它并不隐含操作本身的成功或失

败.

若操作成功地初启,WSAAsyncGetHostByAddr()返回一个 HANDLE 类型的非 0 值，作为请求需要的异步任务句柄.该值可在两种方式下使用.它可通过 WSACancelAsyncRequest()用来取消该操作.也可通过检查 wParam 消息参数,以匹配异步操作和完成消息.

如果异步操作不能初启,WSAAsyncGetHostByAddr()返回一个 0 值,并且可使用 WSAGetLastError()来获取错误号.

评价:

Windows Sockets 的实现使用提供给该函数的缓冲区来构造 hostent 结构以及该结构成员引用的数据区内容.为避免上述的 WSAENOBUFS 错误,应用程序应提供一个至少 MAXGETHOSTSTRUCT 字节大小的缓冲区.

关于 Windows Sockets 提供者的说明：

Windows Sockets 的实现应保证消息能成功地传给应用程序.如果 PostMessage()操作失败,Windows Sockets 的实现必须重发该消息－只要窗口存在.

Windows Sockets 的提供者在消息中组织 lParam 时应使用 WSAMAKEASYNCREPLY 宏.

错误代码:

在应用程序的窗口收到消息时可能会设置下列的错误代码.如上所述,它们可以通过 WSAGETASYNCERROR 宏从应答的消息 lParam 中取出.

WSAENETDOWN　　WINDOWS SOCKETS 实现已检测到网络子系统故障.
WSAENOBUFS　　可用的缓冲区空间不足或没有.
WSAHOST_NOT_FOUND　　未找到授权应答主机.
WSATRY_AGAIN　　未找到非授权应答主机,或 SERVERFAIL.
WSANO_RECOVERY　　不可恢复性错误,FORMERR,REFUSED,NOTIMP.
WSANO_DATA　　合法名,无请求类型的数据记录.

下列的错误可能在函数调用时发生,指出异步操作不能初启.

WSANOTINITIALISED　　在使用本 API 前必须进行一次成功的 WSAStartup()调用.
WSAENETDOWN　　Windows Sockets 的实现已检测到网络子系统故障.
WSAEINPROGRESS　　一个阻塞的 Windows Sockets 操作正在进行.
WSAEWOULDBLOCK　　本异步操作此时由于 Windows Sockets 实现的资源或其它限制的制约而无法调度.

参见:

gethostbyaddr(), WSACancelAsyncRequest()

5.3.2 WSAAsyncGetHostByName()

简述:

获得对应于一个主机名的主机信息.—异步版本.

```
#include <winsock.h>

        HANDLE PASCAL FAR WSAAsyncGetHostByName ( HWND hWnd,
        unsigned int wMsg,  const char FAR * name, char FAR * buf,
        int buflen );
```

hWnd　当异步请求完成时,应该接收消息的窗口句柄.

wMsg　当异步请求完成时,将要接收的消息.

name　指向主机名的指针.

buf 接收 hostent 数据的数据区指针.注意该数据区必须大于 hostent 结构的大小.这是因为不仅 Windows Sockets 实现要用该数据区域容纳 hostent 结构,hostent 结构的成员引用的所有数据也要在该区域内.建议用户提供一个 MAXGETHOSTSTRUCT 字节大小的缓冲区.

buflen　上述数据区的大小.

注释:

本函数是 gethostbyname()的异步版本,是用来获取对应于一个主机名的主机名称和地址信息.Windows Sockets 的实现启动该操作后立刻返回调用方,并传回一个异步任务句柄,应用程序可以用它来标识该操作.当操作完成时,结果(若有的话)将会拷贝到调用方提供的缓冲区,同时向应用程序的窗口发一条消息.

当异步操作完成时,应用程序的窗口 hWnd 接收到消息 wMsg. wParam 参数包含了初次函数调用时返回的异步任务句柄.lParam 的高 16 位包含着错误代码.该代码可以是 winsock.h 中定义的任何错误.错误代码为 0 说明异步操作成功.在成功完成的情况下,提供给初始函数调用的缓冲区中包含了一个 hostent 结构.为存取该结构中的元素,初始的缓冲区指针应置为 hostent 结构的指针,并一如平常地存取.

注意若错误代码为 WSAENOBUFS,它说明在初始调用时由 buflen 指出的缓冲区大小对于容纳所有的结果信息来说太小了.在这种情况下,lParam 的低 16 位含有提供所有信息所需的缓冲区大小数值.如果应用程序认为获取的数据不够,它就可以在设置了足够容纳所需信息的缓冲区后,重新调用 WSAAsyncGetHostByName().(也就是大于 lParam 低 16 位提供的大小.)

错误代码和缓冲区大小应使用 WSAGETASYNCERROR 和 WSAGETASYNCBUFLEN 宏从 lParam 中取出.两个宏定义如下:

```
#define WSAGETASYNCERROR(lParam)        HIWORD(lParam)
#define WSAGETASYNCBUFLEN(lParam)       LOWORD(lParam)
```

使用这些宏可最大地提高应用程序源代码的可移植性.

返回值:

返回值指出异步操作是否成功地初启.注意它并不隐含操作本身的成功或失败.

若操作成功地初启,WSAAsyncGetHostByName()返回一个 HANDLE 类型的非 0 值, 作为请求需要的异步任务句柄.该值可在两种方式下使用.它可通过 WSACancelAsyncRequest()用来取消该操作.也可通过检查 wParam 消息参数,以匹配异步操作和完成消息.

如果异步操作不能初启,WSAAsyncGetHostByName()返回一个 0 值,并且可使用 WSAGetLastError()来获取错误号.

评价:

Windows Sockets 的实现使用提供给该函数的缓冲区来构造 hostent 结构以及该结构成员引用的数据区内容.为避免上述的 WSAENOBUFS 错误,应用程序应提供一个至少 MAXGETHOSTSTRUCT 字节大小的缓冲区.

关于 Windows Sockets 提供者的说明:

Windows Sockets 的实现应保证消息能成功地传给应用程序.如果 PostMessage()操作失败,Windows Sockets 的实现必须重发该消息－只要窗口存在.

Windows Sockets 的提供者在消息中组织 lParam 时应使用 WSAMAKEASYNCREPLY 宏.

错误代码:

在应用程序的窗口收到消息时可能会设置下列的错误代码.如上所述,它们可以通过 WSAGETASYNCERROR 宏从应答的消息 lParam 中取出.

WSAENETDOWN	WINDOWS SOCKETS 实现已检测到网络子系统故障.
WSAENOBUFS	可用的缓冲区空间不足或没有.
WSAHOST_NOT_FOUND	未找到授权应答主机.
WSATRY_AGAIN	未找到非授权应答主机,或 SERVERFAIL.
WSANO_RECOVERY	不可恢复性错误,FORMERR,REFUSED,NOTIMP.
WSANO_DATA	合法名,无请求类型的数据记录.

下列的错误可能在函数调用时发生,指出异步操作不能初启.

WSANOTINITIALISED 在使用本 API 前必须进行一次成功的 WSAStartup()调用.

WSAENETDOWN WINDOWS SOCKETS 实现已检测到网络子系统故障.

WSAEINPROGRESS 一个阻塞的 Windows Sockets 操作正在进行.

WSAEWOULDBLOCK 本异步操作此时由于 Windows Sockets 实现的资源或其它限制的制约而无法调度.

参见:

gethostbyname(), WSACancelAsyncRequest()

5.3.3 WSAAsyncGetProtoByName()

简述:

获得对应于一个协议名的协议信息.—异步版本.

```
#include <winsock.h>

        HANDLE PASCAL FAR WSAAsyncGetProtoByName ( HWND hWnd,
unsigned int wMsg,  const char FAR * name, char FAR * buf,
        int buflen );
```

hWnd 当异步请求完成时,应该接收消息的窗口句柄.

wMsg 当异步请求完成时,将要接收的消息.

name 指向要获得的协议名的指针.

buf 接收 protoent 数据的数据区指针.注意该数据区必须大于 protoent 结构的大小.这是因为不仅 Windows Sockets 实现要用该数据区域容纳 protoent 结构,protoent 结构的成员引用的所有数据也要在该区域内. 建议用户提供一个 MAXGETHOSTSTRUCT 字节大小的缓冲区.

buflen 上述数据区的大小.

注释:

本函数是 getprotobyname()的异步版本,是用来获取对应于一个协议名的协议名称和代号.Windows Sockets 的实现启动该操作后立刻返回调用方,并传回一个异步任务句柄,应用程序可以用它来标识该操作.当操作完成时,结果(若有的话)将会拷贝到调用方提供的缓冲区,同时向应用程序的窗口发一条消息.

当异步操作完成时,应用程序的窗口 hWnd 接收到消息 wMsg. wParam 参数包含了初次函数调用时返回的异步任务句柄.lParam 的高 16 位包含着错误代码.该

代码可以是 winsock.h 中定义的任何错误.错误代码为 0 说明异步操作成功.在成功完成的情况下,提供给初始函数调用的缓冲区中包含了一个 protoent 结构.为存取该结构中的元素,初始的缓冲区指针应置为 protoent 结构的指针,并一如平常地存取.

注意若错误代码为 WSAENOBUFS,它说明在初始调用时由 buflen 指出的缓冲区大小对于容纳所有的结果信息来说太小了.在这种情况下,lParam 的低 16 位含有提供所有信息所需的缓冲区大小数值.如果应用程序认为获取的数据不够,它就可以在设置了足够容纳所需信息的缓冲区后,重新调用 WSAAsyncGetProtoByName().(也就是大于 lParam 低 16 位提供的大小.)

错误代码和缓冲区大小应使用 WSAGETASYNCERROR 和 WSAGETASYNCBUFLEN 宏从 lParam 中取出.两个宏定义如下:

```
#define WSAGETASYNCERROR(lParam)        HIWORD(lParam)
#define WSAGETASYNCBUFLEN(lParam)       LOWORD(lParam)
```

使用这些宏可最大地提高应用程序源代码的可移植性.

返回值:

返回值指出异步操作是否成功地初启.注意它并不隐含操作本身的成功或失败.

若操作成功地初启,WSAAsyncGetProtoByName()返回一个 HANDLE 类型的非 0 值, 作为请求需要的异步任务句柄.该值可在两种方式下使用.它可通过 WSACancelAsyncRequest()用来取消该操作.也可通过检查 wParam 消息参数,以匹配异步操作和完成消息.

如果异步操作不能初启,WSAAsyncGetProtoByName()返回一个 0 值,并且可使用 WSAGetLastError()来获取错误号.

评价:

Windows Sockets 的实现使用提供给该函数的缓冲区来构造 protoent 结构以及该结构成员引用的数据区内容.为避免上述的 WSAENOBUFS 错误,应用程序应提供一个至少 MAXGETHOSTSTRUCT 字节大小的缓冲区.

关于 Windows Sockets 提供者的说明:

Windows Sockets 的实现应保证消息能成功地传给应用程序.如果 PostMessage()操作失败,Windows Sockets 的实现必须重发该消息－只要窗口存在.

Windows Sockets 的提供者在消息中组织 lParam 时应使用 WSAMAKEASYNCREPLY 宏.

错误代码:

在应用程序的窗口收到消息时可能会设置下列的错误代码.如上所述,它们可以通过 WSAGETASYNCERROR 宏从应答的消息 lParam 中取出.

WSAENETDOWN WINDOWS SOCKETS 实现已检测到网络子系统故障.
WSAENOBUFS 可用的缓冲区空间不足或没有.
WSAHOST_NOT_FOUND 未找到授权应答主机.
WSATRY_AGAIN 未找到非授权应答主机,或 SERVERFAIL.
WSANO_RECOVERY 不可恢复性错误,FORMERR,REFUSED,NOTIMP.
WSANO_DATA 合法名,无请求类型的数据记录.

下列的错误可能在函数调用时发生,指出异步操作不能初启.

WSANOTINITIALISED 在使用本 API 前必须进行一次成功的 WSAStartup()调用.
WSAENETDOWN WINDOWS SOCKETS 实现已检测到网络子系统故障.
WSAEINPROGRESS 一个阻塞的 Windows Sockets 操作正在进行.
WSAEWOULDBLOCK 本异步操作此时由于 Windows Sockets 实现的资源或其它限制的制约而无法调度.

参见:

getprotobyname(), WSACancelAsyncRequest()

5.3.4 WSAAsyncGetProtoByNumber()

简述:

获得对应于一个协议号的协议信息.—异步版本.

```
#include <winsock.h>

        HANDLE PASCAL FAR WSAAsyncGetProtoByNumber ( HWND hWnd,
        unsigned int wMsg,  int number, char FAR * buf, int buflen);
```

hWnd 当异步请求完成时,应该接收消息的窗口句柄.
wMsg 当异步请求完成时,将要接收的消息.
number 要获得的协议号,以主机字节序.
buf 接收 protoent 数据的数据区指针.注意该数据区必须大于 protoent 结构的大小.这是因为不仅 Windows Sockets 实现要用该数据区域容纳 protoent 结构,protoent 结构的成员引用的所有数据也要在该区域内. 建议用户提供一个 MAXGETHOSTSTRUCT 字节大小的缓冲区.
buflen 上述数据区的大小.

注释:

本函数是 getprotobynumber()的异步版本,是用来获取对应于一个协议号的协议名称和代号.Windows Sockets 的实现启动该操作后立刻返回调用方,并传回一个异步任务句柄,应用程序可以用它来标识该操作.当操作完成时,结果(若有的话)将会拷贝到调用方提供的缓冲区,同时向应用程序的窗口发一条消息.

当异步操作完成时,应用程序的窗口 hWnd 接收到消息 wMsg. wParam 参数包含了初次函数调用时返回的异步任务句柄.lParam 的高 16 位包含着错误代码.该代码可以是 winsock.h 中定义的任何错误.错误代码为 0 说明异步操作成功.在成功完成的情况下,提供给初始函数调用的缓冲区中包含了一个 protoent 结构.为存取该结构中的元素,初始的缓冲区指针应置为 protoent 结构的指针,并一如平常地存取.

注意若错误代码为 WSAENOBUFS,它说明在初始调用时由 buflen 指出的缓冲区大小对于容纳所有的结果信息来说太小了.在这种情况下,lParam 的低 16 位含有提供所有信息所需的缓冲区大小数值.如果应用程序认为获取的数据不够,它就可以在设置了足够容纳所需信息的缓冲区后,重新调用 WSAAsyncGetProtoByNumber().(也就是大于 lParam 低 16 位提供的大小.)

错误代码和缓冲区大小应使用 WSAGETASYNCERROR 和 WSAGETASYNCBUFLEN 宏从 lParam 中取出.两个宏定义如下:

```
#define WSAGETASYNCERROR(lParam)        HIWORD(lParam)
#define WSAGETASYNCBUFLEN(lParam)       LOWORD(lParam)
```

使用这些宏可最大地提高应用程序源代码的可移植性.

返回值:

返回值指出异步操作是否成功地初启.注意它并不隐含操作本身的成功或失败.

若操作成功地初启,WSAAsyncGetProtoByNumber()返回一个 HANDLE 类型的非 0 值, 作为请求需要的异步任务句柄.该值可在两种方式下使用.它可通过 WSACancelAsyncRequest()用来取消该操作.也可通过检查 wParam 消息参数,以匹配异步操作和完成消息.

如果异步操作不能初启,WSAAsyncGetProtoByNumber()返回一个 0 值,并且可使用 WSAGetLastError()来获取错误号.

评价:

Windows Sockets 的实现使用提供给该函数的缓冲区来构造 protoent 结构以及该结构成员引用的数据区内容.为避免上述的 WSAENOBUFS 错误,应用程序应提供一个至少 MAXGETHOSTSTRUCT 字节大小的缓冲区.

关于 Windows Sockets 提供者的说明：

Windows Sockets 的实现应保证消息能成功地传给应用程序.如果 PostMessage()操作失败,Windows Sockets 的实现必须重发该消息－只要窗口存在.

Windows Sockets 的提供者在消息中组织 lParam 时应使用 WSAMAKEASYNCREPLY 宏.

错误代码:

在应用程序的窗口收到消息时可能会设置下列的错误代码.如上所述,它们可以通过 WSAGETASYNCERROR 宏从应答的消息 lParam 中取出.

WSAENETDOWN　　WINDOWS SOCKETS 实现已检测到网络子系统故障.

WSAENOBUFS　　可用的缓冲区空间不足或没有.

WSAHOST_NOT_FOUND　　未找到授权应答主机.

WSATRY_AGAIN　　未找到非授权应答主机,或 SERVERFAIL.

WSANO_RECOVERY　　不可恢复性错误,FORMERR,REFUSED,NOTIMP.

WSANO_DATA　　合法名,无请求类型的数据记录.

下列的错误可能在函数调用时发生,指出异步操作不能初启.

WSANOTINITIALISED　　在使用本 API 前必须进行一次成功的 WSAStartup()调用.

WSAENETDOWN　　WINDOWS SOCKETS 实现已检测到网络子系统故障.

WSAEINPROGRESS　　一个阻塞的 Windows Sockets 操作正在进行.

WSAEWOULDBLOCK　　本异步操作此时由于 Windows Sockets 实现的资源或其它限制的制约而无法调度.

参见:

getprotobynumber(), WSACancelAsyncRequest()

5.3.5 WSAAsyncGetServByName()

简述:

获得对应于一个服务名和接口的服务信息.－异步版本.

```
#include <winsock.h>

        HANDLE PASCAL FAR WSAAsyncGetServByName ( HWND hWnd,
        unsigned int wMsg,  const char FAR * name, const char FAR *
        proto, char FAR * buf,  int buflen );
```

hWnd　　当异步请求完成时,应该接收消息的窗口句柄.

wMsg　　当异步请求完成时,将要接收的消息.

name　　指向服务名的指针.

proto　　指向协议名称的指针.它可能是 NULL,在这种情况下,WSAAsyncGetServByName()将搜索第一个服务入口(满足 s_name 或 s_aliases 之一和所给的名字匹配.)否则, WSAAsyncGetServByName()将和名和协议同时匹配.

buf 接收 protoent 数据的数据区指针.注意该数据区必须大于 protoent 结构的大小.这是因为不仅 Windows Sockets 实现要用该数据区域容纳 protoent 结构,protoent 结构的成员引用的所有数据也要在该区域内. 建议用户提供一个 MAXGETHOSTSTRUCT 字节大小的缓冲区.

buflen　上述数据区的大小.

注释:

本函数是 getservbyname()的异步版本,是用来获取对应于一个服务名的服务信息.Windows Sockets 的实现启动该操作后立刻返回调用方,并传回一个异步任务句柄,应用程序可以用它来标识该操作.当操作完成时,结果(若有的话)将会拷贝到调用方提供的缓冲区,同时向应用程序的窗口发一条消息.

当异步操作完成时,应用程序的窗口 hWnd 接收到消息 wMsg. wParam 参数包含了初次函数调用时返回的异步任务句柄.lParam 的高 16 位包含着错误代码.该代码可以是 winsock.h 中定义的任何错误.错误代码为 0 说明异步操作成功.在成功完成的情况下,提供给初始函数调用的缓冲区中包含了一个 hostent 结构.为存取该结构中的元素,初始的缓冲区指针应置为 hostent 结构的指针,并一如平常地存取.

注意若错误代码为 WSAENOBUFS,它说明在初始调用时由 buflen 指出的缓冲区大小对于容纳所有的结果信息来说太小了.在这种情况下,lParam 的低 16 位含有提供所有信息所需的缓冲区大小数值.如果应用程序认为获取的数据不够,它就可以在设置了足够容纳所需信息的缓冲区后,重新调用 WSAAsyncGetServByName().(也就是大于 lParam 低 16 位提供的大小.)

错误代码和缓冲区大小应使用 WSAGETASYNCERROR 和 WSAGETASYNCBUFLEN 宏从 lParam 中取出.两个宏定义如下:

```
#define WSAGETASYNCERROR(lParam)          HIWORD(lParam)
#define WSAGETASYNCBUFLEN(lParam)         LOWORD(lParam)
```

使用这些宏可最大地提高应用程序源代码的可移植性.

返回值:

返回值指出异步操作是否成功地初启.注意它并不隐含操作本身的成功或失败.

若操作成功地初启,WSAAsyncGetServByName()返回一个 HANDLE 类型的非 0 值，作为请求需要的异步任务句柄.该值可在两种方式下使用.它可通过 WSACancelAsyncRequest()用来取消该操作.也可通过检查 wParam 消息参数,以匹配异步操作和完成消息.

如果异步操作不能初启,WSAAsyncGetServByName()返回一个 0 值,并且可使用 WSAGetLastError()来获取错误号.

评价:

Windows Sockets 的实现使用提供给该函数的缓冲区来构造 hostent 结构以及该结构成员引用的数据区内容.为避免上述的 WSAENOBUFS 错误,应用程序应提供一个至少 MAXGETHOSTSTRUCT 字节大小的缓冲区.

关于 Windows Sockets 提供者的说明：

Windows Sockets 的实现应保证消息能成功地传给应用程序.如果 PostMessage()操作失败,Windows Sockets 的实现必须重发该消息－只要窗口存在.

Windows Sockets 的提供者在消息中组织 lParam 时应使用 WSAMAKEASYNCREPLY 宏.

错误代码:

在应用程序的窗口收到消息时可能会设置下列的错误代码.如上所述,它们可以通过 WSAGETASYNCERROR 宏从应答的消息 lParam 中取出.

WSAENETDOWN　　WINDOWS SOCKETS 实现已检测到网络子系统故障.
WSAENOBUFS　　可用的缓冲区空间不足或没有.
WSAHOST_NOT_FOUND　　未找到授权应答主机.
WSATRY_AGAIN　　未找到非授权应答主机,或 SERVERFAIL.
WSANO_RECOVERY　　不可恢复性错误,FORMERR,REFUSED,NOTIMP.
WSANO_DATA　　合法名,无请求类型的数据记录.

下列的错误可能在函数调用时发生,指出异步操作不能初启.

WSANOTINITIALISED　　在使用本 API 前必须进行一次成功的 WSAStartup()调用.
WSAENETDOWN　　WINDOWS SOCKETS 实现已检测到网络子系统故障.
WSAEINPROGRESS　　一个阻塞的 Windows Sockets 操作正在进行.
WSAEWOULDBLOCK　　本异步操作此时由于 Windows Sockets 实现的资源或其它限制的制约而无法调度.

参见:

getservbyname(), WSACancelAsyncRequest()

5.3.6 WSAAsyncGetServByPort()

简述:

获得对应于一个服务名和接口的服务信息.－异步版本.

```
#include <winsock.h>

        HANDLE PASCAL FAR WSAAsyncGetServByPort ( HWND hWnd,
        unsigned int wMsg,  int port, const char FAR * proto, char
        FAR * buf, int buflen );
```

hWnd　当异步请求完成时,应该接收消息的窗口句柄.

wMsg　当异步请求完成时,将要接收的消息.

port　服务的接口.以网络字节序.

proto　指向协议名称的指针.它可能是 NULL,在这种情况下,WSAAsyncGetServByName()将搜索第一个服务入口(满足 s_name 或 s_aliases 之一和所给的名字匹配.)否则, WSAAsyncGetServByName()将和名和协议同时匹配.

buf 接收 protoent 数据的数据区指针.注意该数据区必须大于 protoent 结构的大小.这是因为不仅 Windows Sockets 实现要用该数据区域容纳 protoent 结构,protoent 结构的成员引用的所有数据也要在该区域内. 建议用户提供一个 MAXGETHOSTSTRUCT 字节大小的缓冲区.

buflen　上述数据区的大小.

注释:

本函数是 getservbyport()的异步版本,是用来获取对应于一个接口号的服务信息.Windows Sockets 的实现启动该操作后立刻返回调用方,并传回一个异步任务句柄,应用程序可以用它来标识该操作.当操作完成时,结果(若有的话)将会拷贝到调用方提供的缓冲区,同时向应用程序的窗口发一条消息.

当异步操作完成时,应用程序的窗口 hWnd 接收到消息 wMsg. wParam 参数包含了初次函数调用时返回的异步任务句柄.lParam 的高 16 位包含着错误代码.该代码可以是 winsock.h 中定义的任何错误.错误代码为 0 说明异步操作成功.在成功完成的情况下,提供给初始函数调用的缓冲区中包含了一个 hostent 结构.为存取该结构中的元素,初始的缓冲区指针应置为 hostent 结构的指针,并一如平常地存取.

注意若错误代码为WSAENOBUFS,它说明在初始调用时由buflen指出的缓冲区大小对于容纳所有的结果信息来说太小了.在这种情况下,lParam 的低16位含有提供所有信息所需的缓冲区大小数值.如果应用程序认为获取的数据不够,它就可以在设置了足够容纳所需信息的缓冲区后,重新调用WSAAsyncGetServByPort().(也就是大于 lParam 低 16 位提供的大小.)

错误代码和缓冲区大小应使用 WSAGETASYNCERROR 和 WSAGETASYNCBUFLEN 宏从 lParam 中取出.两个宏定义如下:

```
#define WSAGETASYNCERROR(lParam)        HIWORD(lParam)
#define WSAGETASYNCBUFLEN(lParam)       LOWORD(lParam)
```

使用这些宏可最大地提高应用程序源代码的可移植性.

返回值:

返回值指出异步操作是否成功地初启.注意它并不隐含操作本身的成功或失败.

若操作成功地初启,WSAAsyncGetServByPort()返回一个 HANDLE 类型的非 0 值, 作为请求需要的异步任务句柄.该值可在两种方式下使用.它可通过WSACancelAsyncRequest()用来取消该操作.也可通过检查 wParam 消息参数,以匹配异步操作和完成消息.

如果异步操作不能初启,WSAAsyncGetServByPort()返回一个 0 值,并且可使用 WSAGetLastError()来获取错误号.

评价:

Windows Sockets 的实现使用提供给该函数的缓冲区来构造 hostent 结构以及该结构成员引用的数据区内容.为避免上述的 WSAENOBUFS 错误,应用程序应提供一个至少 MAXGETHOSTSTRUCT 字节大小的缓冲区.

关于 Windows Sockets 提供者的说明:

Windows Sockets 的实现应保证消息能成功地传给应用程序.如果PostMessage()操作失败,Windows Sockets 的实现必须重发该消息－只要窗口存在.

Windows Sockets 的提供者在消息中组织 lParam 时应使用WSAMAKEASYNCREPLY 宏.

错误代码:

在应用程序的窗口收到消息时可能会设置下列的错误代码.如上所述,它们可以通过 WSAGETASYNCERROR 宏从应答的消息 lParam 中取出.

WSAENETDOWN　　WINDOWS SOCKETS 实现已检测到网络子系统故障.

WSAENOBUFS　　可用的缓冲区空间不足或没有.
WSAHOST_NOT_FOUND　　未找到授权应答主机.
WSATRY_AGAIN　　未找到非授权应答主机,或 SERVERFAIL.
WSANO_RECOVERY　　不可恢复性错误,FORMERR,REFUSED,NOTIMP.
WSANO_DATA　　合法名,无请求类型的数据记录.

下列的错误可能在函数调用时发生,指出异步操作不能初启.
WSANOTINITIALISED　　在使用本 API 前必须进行一次成功的 WSAStartup()调用.
WSAENETDOWN　　WINDOWS SOCKETS 实现已检测到网络子系统故障.
WSAEINPROGRESS　　一个阻塞的 Windows Sockets 操作正在进行.
WSAEWOULDBLOCK　　本异步操作此时由于 Windows Sockets 实现的资源或其它限制的制约而无法调度.

参见:
getservbyport(), WSACancelAsyncRequest()

5.3.7 WSAAsyncSelect()

简述:
通知套接口有请求事件发生.

```
#include <winsock.h>

int PASCAL FAR WSAAsyncSelect ( SOCKET s, HWND hWnd,
       unsigned int wMsg,  long lEvent );
```

s　标识一个需要事件通知的套接口的描述符.
hWnd　标识一个在网络事件发生时需要接收消息的窗口句柄.
wMsg　在网络事件发生时要接收的消息.
lEvent　位屏蔽码,用于指明应用程序感兴趣的网络事件集合.

注释:
本函数用来请求 Windows Sockets DLL 为窗口句柄发一条消息－无论它何时检测到由 lEvent 参数指明的网络事件.要发送的消息由 wMsg 参数标明.被通知的套接口由 s 标识.
本函数自动将套接口设置为非阻塞模式.
lEvent 参数由下表中列出的值组成.

值　　意义
FD_READ 欲接收读准备好的通知.
FD_WRITE　欲接收写准备好的通知.
FD_OOB　欲接收带边数据到达的通知.
FD_ACCEPT　欲接收将要连接的通知.
FD_CONNECT　欲接收已连接好的通知.
FD_CLOSE　欲接收套接口关闭的通知.

启动一个 WSAAsyncSelect()将使为同一个套接口启动的所有先前的 WSAAsyncSelect()作废. 例如,要接收读写通知,应用程序必须同时用 FD_READ 和 FD_WRITE 调用 WSAAsyncSelect(),如下:

```
rc = WSAAsyncSelect(s, hWnd, wMsg, FD_READ|FD_WRITE);
```

对不同的事件区分不同的消息是不可能的.下面的代码将不会工作;第二个调用将会使第一次调用的作用失效,只有 FD_WRITE 会通过 wMsg2 消息通知到.

```
rc = WSAAsyncSelect(s, hWnd, wMsg1, FD_READ);
rc = WSAAsyncSelect(s, hWnd, wMsg2, FD_WRITE);
```

如果要取消所有的通知,也就是指出 Windows Sockets 的实现不再在套接口上发送任何和网络事件相关的消息,则 lEvent 应置为 0.

```
rc = WSAAsyncSelect(s, hWnd, 0, 0);
```

尽管在本例中,WSAAsyncSelect()立即使传给该套接口的事件消息无效，仍有可能有消息等在应用程序的消息队列中.应用程序因此也必须仍准备好接收网络消息－即使消息作废.用 closesocket()关闭一个套接口也同样使 WSAAsyncSelect()发送的消息作废,但在closesocke()之前队列中的消息仍然起作用.

由于一个已调用 accept()的套接口和用来接收它的侦听套接口有同样的属性，任何为侦听套接口设置的的 WSAAsyncSelect()事件也同样对已接收的套接口起作用.例如，如果一个侦听的套接口有 WSAAsyncSelect()事件 FD_ACCEPT,FD_READ,FD_WRITE，则任何在那个侦听的套接口上接收的套接口将也有 FD_ACCEPT,FD_READ,FD_WRITE 事件,以及同样的 wMsg 的值.若需要不同的 wMsg 及事件,应用程序应调用 WSAAsyncSelect(),将已接收的套接口和想要发送的新消息作为参数传递.

当某一套接口 s 上发生了一个已命名的网络事件,应用程序窗口 hWnd 会接收到消息 wMsg.wParam 参数标识了网络事件发生的套接口.lParam 的低字指明了发生的网络事件.lParam 的高字则含有一个错误代码.该错误代码可以是 winsock.h 中定义的任何错误.

错误代码和事件可以通过 WSAGETSELECTERRORH 和 WSAGETSELECTEVENT 宏从

lParam 中取出.定义如下:

```
#define WSAGETSELECTERROR(lParam)        HIWORD(lParam)
#define WSAGETSELECTEVENT(lParam)        LOWORD(lParam)
```

注意:在 accept()调用和为改变事件或 wMsg 的 WSAAsyncSelect()调用中有一个计时窗口.应用程序如果需要给侦听的和调用过 accept()的套接口以不同的 wMsg,它就应该在侦听的套接口上请求 FD_ACCEPT 事件,然后在 accept()调用后设置相应的事件.由于 FD_ACCEPT 从不发送给已连接的套接口,而 FD_READ,FD_WRITE,FD_OOB 及 FD_CLOSE 也从不发送给侦听套接口,所以不会产生困难.

使用以上的宏将最大限度的提高应用程序的可移植性.

返回的可能网络事件如下:

值	意义
FD_READ	套接口 s 准备读
FD_WRITE	套接口 s 准备写
FD_OOB	带外数据准备好在套接口 s 上读.
FD_ACCEPT	套接口 s 准备接收新的将要到来的连接.
FD_CONNECT	套接口 s 上的连接完成.
FD_CLOSE	由套接口 s 标识的连接已关闭.

返回值:

0	若应用程序感兴趣的网络事件的声明成功.
SOCKET_ERROR	否则.可通过调用 WSAGetLastError()返回特定的错误代码.

评价:

尽管 WSAAsyncSelect()可以以多个事件的组合来调用,应用程序窗口还是会为每个网络事件接收一条消息.

如同 select()函数,WSAAsyncSelect()会被频繁地调用来决定,何时一次数据转移操作(send()或 recv())可以启动,并且可以立刻成功.尽管如此,健壮的应用程序必须做好这样的准备, 即它可能接收到消息及启动了一个会立即返回 WSAEWOULDBLOCK 的 Windows Sockets API 调用.例如,下列的事件序列是可能的:

(i) 数据到达套接口 s;Windows Sockets 传递 WSAAsyncSelect 消息.

(ii) 应用程序处理其它一些消息.

(iii) 在处理过程中,应用程序启动了 ioctlsocket(s,FIONREAD...)并且注意到有数据准备好读.

(iv) 应用程序启动 recv(s,...)来读数据.

(v) 应用程序循环处理下一条消息,最终到达 WSAAsyncSelect 消息,表示数据已准备好读.

(vi)　应用程序启动 recv(s,...),但失败并有错误 WSAEWOULDBLOCK.

其它的事件序列也是可能的.

Windows Sockets DLL 不会不断地为某一特定的网络事件向一个应用程序发送消息. 如果已成功地向应用程序窗口发送了一特定事件的通知,对该应用程序窗口将不再为该网络事件发消息,直到应用程序调用函数隐含地重新通知该网络事件.

事件	重新通知函数
FD_READ	recv()或 recvfrom()
FD_WRITE	send()或 sendto()
FD_OOB	recv()
FD_ACCEPT	accept()
FD_CONNECT	无
FD_CLOSE	无

任何对重新通知函数的调用,即使失败,也会达到为相关事件发重新通知消息的效果.

对 FD_READ,FD_OOB 和 FD_ACCEPT 事件,消息传递是"水平触发"(level-triggered)的.这意味着,若调用了重新通知函数并且相关的事件对该调用仍有效,WSAAsyncSelect()消息就将传给应用程序.这为应用程序提供了事件驱动以及不必考虑在任一时刻到达的数据量的能力.考虑下列序列:

(i) Windows Sockets DLL 在套接口 s 上接收 100 字节的数据并传递一个 FD_READ 消息.

(ii)　应用程序启动 recv(s,buffptr,50,0)接收 50 字节.

(iii)　由于仍有数据未读,Windows Sockets DLL 发送另一个 FD_READ 消息.

根据以上语义,应用程序不必在收到 FD_READ 消息时读进所有可读的数据－对应于每一 FD_READ 消息进行一次 recv()调用是恰当的.如果应用程序为一个 FD_READ 消息而启动了多个 recv()调用,它将接收到多个 FD_READ 消息.这样的应用程序可能希望在开始 recv()调用(通过不为 FD_READ 事件置位的 WSAAsyncSelect()函数调用)之前关闭 FD_READ 消息.

如果在应用程序初次调用 WSAAsyncSelect()或当调用了重新通知函数时,有一个事件为真, 则会发送一个相应的消息.例如,若应用程序调用 listen(),就会试图进行连接,然后应用程序调用 WSAAsyncSelect()声明它需要为套接口接收 FD_ACCEPT 消息,Windows Sockets 的实现就会立即传递一个 FD_ACCEPT 消息.

FD_WRITE 事件处理起来稍有不同.FD_WRITE 消息是在套接口第一次用 connect()连接或由 accept()接受,并且在 send()或 sendto()以 WSAWOULDBLOCK 错误失败后缓冲区空闲时发送的.因此,应用程序可以假设发送可能在第一次 FD_WRITE 消息时开始,并持续到一次返回 WSAEWOULDBLOCK 的发送. 在这样的失败后,应用程序将被通知,FD_WRITE 消息的发送又将可能.

FD_OOB 事件只用在当套接口配置成独立接收带外数据时.如果一个套接口被配置成接收感兴趣的带外数据状态,带外数据将和普通数据等同视之,并且应用程序应该注册它感兴趣的方面,然后将接收 FD_READ 事件,而不是 FD_OOB 事件.应用程序可以设置或监控带外数据处理的方法(通过使用 setsockopt()或 getsockopt()函数,及 SO_OOBINLINE 选项).

在 FD_CLOSE 消息中的错误代码指出套接口的关闭是正常的还是异常的.如果错误代码是 0,则关闭是正常的;若错误代码是 WSAECONNRESET,则套接口的虚套接口将被重置.这些只对 SOCK_STREAM 类型的套接口起作用.

FD_CLOSE 消息在相应套接口的虚电路关闭指令接收到时发送.在 TCP 术语中,这意味着 FD_CLOSE 在连接进入了 FIN WAIT 或 CLOSE WAIT 状态时发送.这是远端对发送方进行了 shutdown()调用或 closesocket()调用的结果.

请注意你的应用程序将只会收到 FD_CLOSE 消息来指出虚电路的关闭.它不会收到 FD_READ 消息来表示该状况.

错误代码:

WSANOTINITIALISED　　在使用本 API 前必须进行一次成功的 WSAStartup()调用.

WSAENETDOWN　　WINDOWS SOCKETS 实现已检测到网络子系统故障.

WSAEINVAL　　指出指定的参数之一是非法的.

WSAEINPROGRESS　　一个阻塞的 Windows Sockets 操作正在进行.

附加的错误代码可能在应用程序窗口接收到消息时被置.这些代码可以用 WSAGETSELECTERROR 宏从 lParam 中取出.对应于每个网络事件的可能错误代码为:

事件:FD_CONNECT

WSAEADDRINUSE　　给定的地址已被使用.

WSAEADDRNOTAVAIL　　指定的地址在本地机器不能使用.

WSAEAFNOSUPPORT　　指定族的地址不能和本套接口同时使用.

WSAECONNREFUSED　　连接的尝试被拒绝.

WSAEDESTADDRREQ　　需要一个目的地址.

WSAEFAULT　　namelen 参数不正确.

WSAEINVAL　　套接口已经约束到一个地址.

WSAEISCONN　　套接口已经连接.

WSAEMFILE　　没有可用的文件描述符.

WSAENETUNREACH　　此时网络不能从该主机访问.

WSAENOBUFS　　无可用的缓冲区空间.套接口不能连接.

WSAENOTCONN　　套接口没有连接.

WSAENOTSOCK　　该描述符是文件,不是套接口.

WSAETIMEDOUT　　试图连接超时,未建立连接.

事件:FD_CLOSE
WSAENETDOWN　　WINDOWS SOCKETS 实现已检测到网络子系统故障.
WSAECONNRESET　　连接由远端重建.
WSAECONNABORTED　　由于超时或其它失败放弃连接.
事件:FD_READ
事件:FD_WRITE
事件:FD_OOB
事件:FD_ACCEPT
WSAENETDOWN　　WINDOWS SOCKETS 实现已检测到网络子系统故障.

关于 Windows Sockets 提供者的说明：

Windows Sockets 的提供者应确保消息可以成功地传给应用程序.如果 PostMessag()操作失败,Windows Sockets 的实现必须重发该消息－只要窗口存在.

Windows Sockets 提供者应使用 WSAMAKESELECTREPLY 宏来构造消息中的 lParam 参数.

当套接口关闭时,Windows Sockets提供者应清除所有保留下来要发送给应用程序窗口的消息.然而应用程序必须准备好接收,放弃任何在closesocket()之前可能已经发送的消息.

参见:

select()

5.3.8 WSACancelAsyncRequest()

简述:

取消一次未完成的异步操作.

```
#include <winsock.h>
int PASCAL FAR WSACancelAsyncRequest(HANDLE hAsyncTackHandle);
```

hAsyncTaskHandle　　指明要取消的异步操作.

注释:

WSACancelAsyncRequest()函数用于取消一次异步操作,该异步操作应是以一个 WSAAsyncGetXByY()函数(诸如 WSAAsyncGetHostByName())启动的.hAsyncTaskHandle 参数标识了要取消的操作,它应由初始函数作为异步任务句柄返回.

返回值:

0　　异步操作成功地被取消.

SOCKET_ERROR　　其它情况.(同时可通过调用 WSAGetLastError()获得错误代码)

评论:

试图取消一个已存在的异步操作 WSAAsyncGetXByY()可能失败(错误代码 WSAEALREADY)，原
因有二：首先，原来的操作已经完成，并且应用程序已经处理了结果消息。其次，原始操作已经完成，但结果消息仍在应用程序窗口队列中等待。

关于 Windows Sockets 提供者的说明：

应用程序是否能有效地区分 WSAEINVAL 和 WSAEALREADY 是不清楚的，因为在这两种情况下，错误代码指出不存在指定句柄的异步操作在运行。(小例外：0 总是非法的异步任务句柄。)Windows Sockets 规格说明不会规定一个 Windows Sockets 实现怎样区分这两种情况。最大可能的情况是，Windows Sockets 应用程序应将两种错误视为相同。

错误代码：

WSANOTINITIALISED　　在使用本 API 前必须进行一次成功的 WSAStartup()调用.

WSAENETDOWN　　WINDOWS SOCKETS 实现已检测到网络子系统故障.

WSAEINVAL　　指出指定的参数之一是非法的.

WSAEINPROGRESS　　一个阻塞的 Windows Sockets 操作正在进行.

WSAEALREADY　　被废除的异步调用已经完成.

参见：

WSAAsyncGetHostByAddr(), WSAAsyncGetHostByName(), WSAAsyncGetProtoByNumber(), WSAAsyncGetProtoByName(), WSAAsyncGetHostByName(), WSAAsyncGetServByPort(), WSAAsyncGetServByName().

5.3.9 WSACancelBlockingCall()

简述：

取消一次正在进行中的阻塞调用。

#include <winsock.h>

int PASCAL FAR WSACancelBlockingCall(void);

注释：

本函数取消了任何本任务中尚未完成的阻塞操作。通常用于以下两种情况：

（1）。在一个阻塞调用进行时，应用程序同时在处理接收到的消息。在这种情况下，WSAIsBlocking()返回 True.

（2）。一个阻塞调用在进行时，Windows Sockets 已经回调了应用程序的“ 阻塞钩子 “ 函数。（如 WSASetBlockingHook()）

在每种情况中，原来的阻塞调用将尽快中止，并产生错误码 WSAEINTR。(在（1）中，中止发生在 Windows 消息调度将控制转移到 Windows Sockets 的阻塞例程中时。在（2）中，阻塞调用将在阻塞钩子函数完成时中止。)

在进行阻塞的 connect()操作的情况下，Windows Sockets 的实现将尽可能中止阻塞调用，但在连接完成（已经复位）或超时之前，它不可能释放套接口资源。同样值得注意的是在应用程序立即尝试打开一个新的套接口(若没有可用的套接口)或试图连接(connect())同一个套接口时。

取消一个 accept()或 select()调用不会迫使套接口经过这些调用。只有特殊的调用会失败H魏卧谌∠　　昂戏 i牟僮彭谌∠　　笠餐　　戏ǎ 捉涌诘淖刺　谌魏吻榭魷露疾换崾苡跋臁*

取消任何除 accept()和 select()之外的操作可能导致套接口进入非终结的状态.如果一个应用程序取消了一个套接口上的阻塞操作，应用程序唯一可以在套接口上操作的函数调用就是 CloseSocket(). 尽管其它一些操作可以在一些 Windows Sockets 实现上运作。如果一个应用程序想获得最大的可移植性，它必须注意不要在取消操作后依赖于 performing operations.应用程序可通过置 SO_LINGER 上的超时为 0 来重置连接。

如果一个取消操作损害了 SOCK_STREAM 的数据流的完整性，Windows Sockets 实现必须重建连接并且用 WSAECONNABORTED 使所有将来的操作(除了 closesocket())失败。

返回值：

0　　操作成功地被取消。

SOCKET_ERROR　　其它。（可通过 WSAGetLastError()获得相应错误代码）

评价：

注意网络操作在 WSACancelBlockingCall()运行之前完成是可能的。例如，在应用程序处于阻塞钩子中时数据可以在中断时接收到用户缓冲区。在这种情况下，阻塞操作将成功返回如同 WSACancelBlockingCall()从未调用过。注意 WSACancelBlockingCall()仍是成功的。确认一个操作是否真正地被取消的唯一

办法是检查从阻塞调用的 WSAEINTR 的返回值。

错误代码：

WSANOTINITIALISED 在使用本 API 前必须进行一次成功的 WSAStartup()调用.

WSAENETDOWN WINDOWS SOCKETS 实现已检测到网络子系统故障.

WSAEINVAL 指出指定的参数之一是非法的.

5.3.10 WSACleanup()

简述:

中止 Windows Sockets DLL 的使用.

```
#include <winsock.h>

int PASCAL FAR WSACleanup ( void );
```

注释:

应用程序或 DLL 在使用 Windows Sockets 服务之前必须要进行一次成功的 WSAStartup()调用.当它完成了 Windows Sockets 的使用后,应用程序或 DLL 必须调用 WSACleanup()将其从 Windows Sockets 的实现中注销,并且该实现释放为应用程序或 DLL 分配的任何资源.任何打开的并已建立连接的 SOCK_STREAM 类型套接口在调用 WSACleanup()时会重置; 而已经由 closesocket()关闭却仍有要发送的悬而未决数据的套接口则不会受影响－ 该数据仍要发送.

对应于一个任务进行的每一次 WSAStartup()调用,必须有一个 WSACleanup()调用.只有最后的 WSACleanup()做实际的清除工作;前面的调用仅仅将 Windows Sockets DLL 中的内置引用计数递减.一个简单的应用程序为确保 WSACleanup()调用了足够的次数,可以在一个循环中不断调用 WSACleanup()直至返回 WSANOTINITIALISED.

返回值:

0 操作成功.

SOCKET_ERROR 否则.同时可以调用 WSAGetLastError()获得错误代码.

评价:

一个常见的 Windows Sockets 编程错误是:试图在一个阻塞钩子函数中调用 WSACleanup()并且检测返回值失败.如果在一次阻塞调用正在进行时应用程序需

要退出,应用程序必须首先通过调用 WSACancelBlockingCall()使该阻塞操作作废，然后一旦控制返回给应用程序时就启动 WSACleanup().

关于 Windows Sockets 提供者的说明：

良好的 Windows Sockets 应用程序会通过调用 WSACleanup()指出它从 Windows Sockets 实现中注销.本函数因此可以用来释放分配给指定应用程序的资源.

Windows Sockets 的实现必须能处理应用程序在调用 WSACleanup()函数之前就中止的情况.－例如,返回一个错误.

在一个多线程的环境下,WSACleanup()中止了 Windows Sockets 在所有线程上的操作.

Windows Sockets 的实现必须确认 WSACleanup()调用后,应用程序能调用 WSAStartup()函数来重新建立 Windows Sockets 的应用.

错误代码:

WSANOTINITIALISED　　使用本 API 前必须要进行一次成功的 WSAStartup()调用.

WSAENETDOWN　　Windows Sockets 的实现已经检测到网络子系统故障.

WSAEINPROGRESS　　一个阻塞的 Windows Sockets 操作正在进行.

参见:

WSAStartup()

5.3.11 WSAGetLastError()

简述:

获得上次失败操作的错误状态.

```
#include <winsock.h>

int PASCAL FAR WSAGetLastError ( void );
```

注释:

本函数返回上次发生的网络错误.当一特定的 Windows Sockets API 函数指出一个错误已经发生,本函数就应调用来获得对应的错误代码.

返回值:

返回值指出了本线程进行的上一次 Windows Sockets API 函数调用时的错误代码.

关于 Windows Sockets 提供者的说明：

这里使用 WSAGetLastError()函数来获得上一次的错误代码,而不是依靠全局错误变量，是为了提供和将来的多线程环境相兼容.

注意在一个非占先的 Windows 环境下,WSAGetLastError()只用来获得 Windows Sockets API 错误.在占先环境下,WSAGetLastError()将调用 GetLastError()，来获得所有在每线程基础上的 Win32 API 函数的错误状态.为提高可移植性,应用程序应在调用失败后立即使用 WSAGetLastError().

参见:

WSASetLastError()

5.3.12 WSAIsBlocking()

简述:

判断是否有阻塞调用正在进行.

```
#include <winsock.h>

BOOL PASCAL FAR WSAIsBlocking ( void );
```

注释:

本函数允许任务判断它是否在等待前一次阻塞调用完成时执行.

返回值:

TRUE　　如果存在一个尚未完成的阻塞函数在等待完成.

FALSE　　否则.

评价:

尽管在阻塞套接口上进行的调用对于应用程序来说似乎"阻塞"着,Windows Sockets DLL 必须放弃处理机以使其它应用程序可以使用.这意味着对于启动该阻塞调用的应用程序来说可能会重入－这依赖于它接收的消息.在这种情况下,WSAIsBlocking()函数可用来确定在等待一个未完成的阻塞调用完成时,本任务是否重入.注意 Windows Sockets 禁止对每一线程多于一个未完成的调用.

关于 Windows Sockets 提供者的说明：

Windows Sockets的实现必须禁止在每个线程上多于一次的未完成阻塞调用.

5.3.13 WSASetBlockingHook()

简述:

建立一个应用程序指定的阻塞钩子函数.

```
#include <winsock.h>

FARPROC PASCAL FAR WSASetBlockingHook ( FARPROC lpBlockFunc );

lpBlockFunc      指向要安装的阻塞函数的函数指针.
```

注释:

本函数安装了一个新的函数,由 Windows Sockets 的实现用来实现阻塞套接口函数调用.

Windows Sockets的实现中包含了一种缺省的机制,通过它可以实现阻塞套接口函数. 函数 WSASetBlockingHook()为应用程序提供了在"阻塞"时执行自己的程序,来代替缺省的函数.

当一个应用程序调用了一个阻塞的 Windows Sockets API 操作时,Windows Sockets 的实现启动该操作,然后进入了和下列伪代码相似的循环:

```
for(;;) {
    /* flush messages for good user response */
    while(BlockingHook())
        ;
    /* check for WSACancelBlockingCall() */
    if(operation_cancelled())
        break;
    /* check to see if operation completed */
    if(operation_complete())
        break;     /* normal completion */
}
```

注意 Windows Sockets 的实现可能以不同的次序运行上述代码,例如,对操作完成的检查可能发生在调用阻塞钩子函数之前.缺省的 BlockingHook()函数如下:

```
BOOL DefaultBlockingHook(void) {
```

```
        MSG msg;
        BOOL ret;
        /* get the next message if any */
        ret = (BOOL)PeekMessage(&msg,NULL,0,0,PM_REMOVE);
        /* if we got one, process it */
        if (ret) {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
        /* TRUE if we got a message */
        return ret;
    }
```

WSASetBlockingHook()函数用来支持需要更复杂消息处理的应用程序－例如,使用了 MDI(多文本界面)的程序.它并不是为运行通常应用程序函数的.特别的,唯一可以由客户阻塞钩子函数调用的唯一 Windows Sockets API 函数是 WSACancelBlockingCall()－它将引起阻塞循环中止.

本函数必须为 Windows 的非多线程版本和多线程版本(如 Windows NT)提供每线程基础上的实现.这样，它为特殊的任务或线程提供了不影响其它任务或线程的基础上替换阻塞机制的能力.

在 Windows 的多线程版本中没有缺省的阻塞钩子函数－阻塞调用阻塞了进行该调用的线程.然而，应用程序可以通过调用 WSASetBlockingHook()安装一个特定的阻塞钩子.这为依赖于阻塞钩子的应用程序提供了简单的可移植性.

返回值:

返回值是一个指向前面安装的阻塞函数例程的指针.调用 WSASetBlockingHook()函数的应用程序或库应该保留返回值,以使它在需要时能恢复.(若"嵌套"并不重要,应用程序可以简单地放弃 WSASetBlockingHook()返回值,并且最终使用 WSAUnhookBlockingHook()来恢复缺省的机制.)如果操作失败,返回一个 NULL 指针,并且可通过调用 WSAGetLastError()获得特定的错误代码.

错误代码:

WSANOTINITIALISED　　使用本 API 前必须要进行一次成功的 WSAStartup()调用.

WSAENETDOWN　　Windows Sockets 的实现已经检测到网络子系统故障.

WSAEINPROGRESS　　一个阻塞的 Windows Sockets 操作正在进行.

参见:
WSAUnhookBlockingHook()

5.3.14 WSASetLastError()

简述:
设置可以被 WSAGetLastError()接收的错误代码.

```
#include <winsock.h>

void PASCAL FAR WSASetLastError ( int iError );
```

iError 指明将被后续的 WSAGetLastError()调用返回的错误代码.

注释:
本函数允许应用程序为当前线程设置错误代码,并可由后来的 WSAGetLastError()调用返回. 注意任何由应用程序调用的后续 Windows Sockets 函数都将覆盖本函数设置的错误代码.

关于 Windows Sockets 提供者的说明:
在 Win32 环境中,本函数将调用 SetLastError().

返回值:
无.

错误代码:
WSANOTINITIALISED 使用本 API 前必须要进行一次成功的 WSAStartup()调用.

参见:
WSAGetLastError()

5.3.15 WSAStartup()

简述:

#include <winsock.h>

int PASCAL FAR WSAStartup (WORD wVersionRequested, LPWSADATA lpWSAData);

wVersionRequested Windows Sockets API 提供的调用方可使用的最高版本号.高位字节指出副版本(修正)号,低位字节指明主版本号.

lpWSAData 指向 WSADATA 数据结构的指针,用来接收 Windows Sockets 实现的细节.

注释:

本函数必须是应用程序或 DLL 调用的第一个 Windows Sockets 函数.它允许应用程序或 DLL 指明 Windows Sockets API 的版本号及获得特定 Windows Sockets 实现的细节.应用程序或 DLL 只能在一次成功的 WSAStartup()调用之后才能调用进一步的 Windows Sockets API 函数.

为支持日后可能和 Windows Sockets 1.1 有功能上差异的 Windows Sockets 实现及应用程序,在 WSAStartup()中规定了一个协议.WSAStartup()的调用方和 Windows Sockets DLL 互相通知对方它们可以支持的最高版本,并且互相确认对方的最高版本是可接受的. 在 WSAStartup()函数的入口,Windows Sockets DLL 检查了应用程序所需的版本.如果版本高于 DLL 支持的最低版本,则调用成功并且 DLL 在 wHighVersion 中返回它所支持的最高版本,在 wVersion 中返回它的高版本和 wVersionRequested 中的较小者.然后 Windows Sockets DLL 就会假设应用程序将使用 wVersion.如果 WSDATA 结构中的 wVersion 域对调用方来说不可接收, 它就应调用 WSACleanup()函数并且要么去另一个 Windows Sockets DLL 中搜索,要么初始化失败.

本协议允许 Windows Sockets DLL 和 Windows Sockets 应用程序共同支持一定范围的 Windows Sockets 版本.如果版本范围有重叠,应用程序就可以成功地使用 Windows Sockets DLL.下列的图表给出了 WSAStartup()在不同的应用程序和 Windows Sockets DLL 版本中是如何工作的:

应用程序版本	DLL 版本	wVersionRequested	wVersion	wHighVersion	最终结果
1.1	1.1	1.1	1.1	1.1	use 1.1
1.0 1.1	1.0	1.1	1.0	1.0	use 1.0
1.0	1.0 1.1	1.0	1.0		

1.1 use 1.0
1.1 1.0 1.1 1.1 1.1
1.1 use 1.1
1.1 1.0 1.1 1.0
1.0 失败
1.0 1.1 1.0 一-
一- WSAVERNOTSUPPORTED
1.0 1.1 1.0 1.1 1.1 1.1
1.1 use 1.1
1.1 2.0 1.1 2.0 1.1
1.1 use 1.1
2.0 1.1 2.0 1.1
1.1 失败

下列代码段给出了只支持 Windows Sockets 1.1 版本的应用程序是如何进行 WSAStartup()调用的:

```
WORD wVersionRequested;
WSADATA wsaData;
int err;

wVersionRequested = MAKEWORD( 1, 1 );

err = WSAStartup( wVersionRequested, &wsaData );
if ( err != 0 ) {
    /* Tell the user that we couldn't find a useable */
    /* winsock.dll.                                  */
    return;
}

/* Confirm that the Windows Sockets DLL supports 1.1.*/
/* Note that if the DLL supports versions greater    */
/* than 1.1 in addition to 1.1, it will still return */
/* 1.1 in wVersion since that is the version we      */
/* requested.                                        */

if ( LOBYTE( wsaData.wVersion ) != 1 ||
        HIBYTE( wsaData.wVersion ) != 1 ) {
```

```
    /* Tell the user that we couldn't find a useable */
    /* winsock.dll.                                  */
    WSACleanup( );
    return;
}

/* The Windows Sockets DLL is acceptable.  Proceed.  */
```

下面的代码段示例了只支持 1.1 版的 Windows Sockets DLL 是如何进行 WSAStartup()协商的:

```
/* Make sure that the version requested is >= 1.1.   */
/* The low byte is the major version and the high    */
/* byte is the minor version.                        */

if ( LOBYTE( wVersionRequested ) < 1 ||
     ( LOBYTE( wVersionRequested ) == 1 &&
       HIBYTE( wVersionRequested ) < 1 ) {
    return WSAVERNOTSUPPORTED;
}

/* Since we only support 1.1, set both wVersion and  */
/* wHighVersion to 1.1.                              */

lpWsaData->wVersion = MAKEWORD( 1, 1 );
lpWsaData->wHighVersion = MAKEWORD( 1, 1 );
```

一旦应用程序或 DLL 进行了一次成功的 WSAStartup()调用,它就可以继续进行其它所需的 Windows Sockets API 调用.当它完成了使用该 Windows Sockets DLL 的服务后,应用程序或 DLL 必须调用 WSACleanup()以允许 Windows Sockets DLL 释放任何该应用程序的资源.

实际的 Windows Sockets 实现细节在 WSAData 结构中描述如下:

```
struct WSAData {
    WORD           wVersion;
    WORD           wHighVersion;
    char
    szDescription[WSADESCRIPTION_LEN+1];
```

```
    char            szSystemStatus[WSASYSSTATUS_LEN+1];
    unsigned short iMaxSockets;
    unsigned short iMaxUdpDg;
    char FAR *          lpVendorInfo;
};
```

该结构的成员为:

成员	用法
wVersion	Windows Sockets DLL 期待调用方使用的 Windows Sockets 规范的版本.
wHighVersion	DLL 可支持的 Windows Sockets 规范的最高版本.通常它和 wVersion 相同.
szDescription	一个 null 结尾的 ASCII 字串,Windows Sockets DLL 将 Windows Sockets 实现的说明及厂商描述拷至该串.这段文本(长度最多 256 个字符)可能包含任何字符, 但厂家注意到不把控制和格式字符包含进去:该字串的最可能用法就是在状态消息中显示.
szSystemStatus	一个 null 结尾的 ASCII 字串,Windows Sockets DLL 将相关的状态和配置信息拷至该串.Windows Sockets DLL 只有在该信息对用户或支撑人员有用时才会使用该域:它不应该被认为是 szDescription 域的扩展.
iMaxSockets	一个进程可以打开的最大套接口数目.Windows Sockets 的实现可以提供一个全局的套接口池给任何进程分配;也可以为每个进程分配套接口资源.该数字可反映出 Windows Sockets DLL 或网络软件是如何配置的.应用程序员可以使用该数字作为该 Windows Sockets 实现是否可以被应用程序使用的原始依据.例如,一个 X Windows 服务器可能在它启动时检查 iMaxSockets:若它小于 8,应用程序应显示一条错误信息, 让用户重新配置网络软件.(这是 szSystemStatus 可能使用到的一种情况.)显然,并不保证一个特定的应用程序可以实际分配到 iMaxSockets 个套接口,因为可能有其它的 Windows Sockets 应用程序在使用.
iMaxUdpDg	以字节表示的可由 Windows Sockets 应用程序发送或接收的最大 UDP 数据报的大小.如果应用程序没有给出限制,iMaxUdpDg 为 0.在 Berkeley 套接口的许多实现中,对于 UDP 数据报的导向有一个隐含的限制 8192 字节.Windows Sockets 的实现可以在分配碎片重组缓冲区的基础上给出界限.对于一般的 Windows Sockets 实现 iMaxUdpDg 的最小值为 512.注意不考虑 iMaxUdpDg的值,而试图在网络上发送一个大于最大传输单元(MTU)的广播数据报是不明智的.(Windows Sockets API 没有提供发现 MTU 的机制,但它必须不小于 512 字节.)
lpVendorInfo	指向厂商规定数据结构的远指针.该结构的定义

(如果提供)超出了本规范的范围.

应用程序或 DLL 若需要多次得到 WSAData 结构信息,就必须多次调用 WSAStartup().然而,wVersionRequired 参数假设在所有调用 WSAStartup()中都相同;也就是,应用程序或 DLL 不能在第一次调用 WSAStartup()后改变 Windows Sockets 的版本号.

对应于每一次 WSAStartup()调用必须有一个 WSACleanup()调用,以使第三级(third-party)DLL 可以利用和应用程序相关的 Windows Sockets DLL.这意味着,例如,如果应用程序调用了 WSAStartup()三次,它就必须调用 WSACleanup()三次.对 WSACleanup()的前两次调用除了减少内置计数器以外不做任何事, 对任务的最后一次 WSACleanup()调用为任务释放了所有所需的资源.

返回值:

0 成功.

否则返回下列的错误代码之一.注意通常依靠应用程序调用 WSAGetLastError()机制获得的错误代码是不能使用的,因为 Windows Sockets DLL 可能没有建立"上一错误"信息储存的客户数据区域.

关于 Windows Sockets 提供者的说明:

每一个 Windows Sockets 应用程序必须在进行其它 Windows Sockets API 调用前进行 WSAStartup()调用.这样,本函数就可以用于初始化的目的.

进一步的说明在 WSACleanup()的说明中有讨论.

错误代码:

WSASYSNOTREADY 指出网络通信依赖的网络子系统还没有准备好.

WSAVERNOTSUPPORTED 所需的 Windows Sockets API 的版本未由特定的 Windows Sockets 实现提供.

WSAEINVAL 应用程序指出的 Windows Sockets 版本不被该 DLL 支持.

参见:

send(), sendto(), WSACleanup()

5.3.16 WSAUnhookBlockingHook()

简述:

恢复缺省的阻塞钩子函数.

```
#include <winsock.h>

int PASCAL FAR WSAUnhookBlockingHook ( void );
```

注释:

本函数除去了任何先前安装的阻塞钩子函数,并且重新安装缺省的阻塞钩子函数.

WSAUnhookBlockingHook()将肯定安装缺省的钩子函数,而非上一个.如果应用程序希望嵌套钩子函数－也就是,建立一个临时的钩子函数,然后返回前一个钩子函数(不论是缺省的还是由前面的 WSASetBlockingHook()建立的)－它必须储存和恢复 WSASetBlockingHook()的返回值;不能使用 WSAUnhookBlockingHook().

在 Windows 的多线程版本(如 Windows NT)中没有缺省的阻塞钩子函数.调用 WSAUnhookBlockingHook()去除了应用程序和任何阻塞调用(阻塞了进行该调用的线程本身)安装的所有阻塞钩子函数.

返回值:

0　　操作成功.

SOCKET_ERROR　　否则.同时可以调用 WSAGetLastError()获得错误代码.

错误代码:

WSANOTINITIALISED　　使用本 API 前必须要进行一次成功的 WSAStartup()调用.

参见:

WSASetBlockingHook()

第六章 Windows Socket 2 的扩展特性

这一章将讨论从 Windows Sockets 1.1 到 Windows Socket 2 的主要变动。

6.1 同时使用多个传输协议

为了用户能够同时使用多个传输协议，在 Windows Socket 2 中，结构有所改变。在 Windows Sockets 1.1 中，软件开发商所提供的 DLL 实现了 Windows Sockets 的 API 和 TCP/IP 协议栈。Windows Sockets DLL 和底层协议栈的接口是唯一而且独占的。Windows Socket 2 改变了这种模型：它定义了一个 Windows Sockets DLL 和底层协议栈间的标准服务提供接口(SPI)。这使得一个 Windows Sockets DLL 能够同时访问不同软件开发商的多个底层协议栈。此外，Windows Sockets 2 并不象 Windows Sockets 1.1 仅支持 TCP/IP 协议栈。与 Windows 开放系统结构(WOSA)兼容的 Windows Sockets 2 的结构如下图：

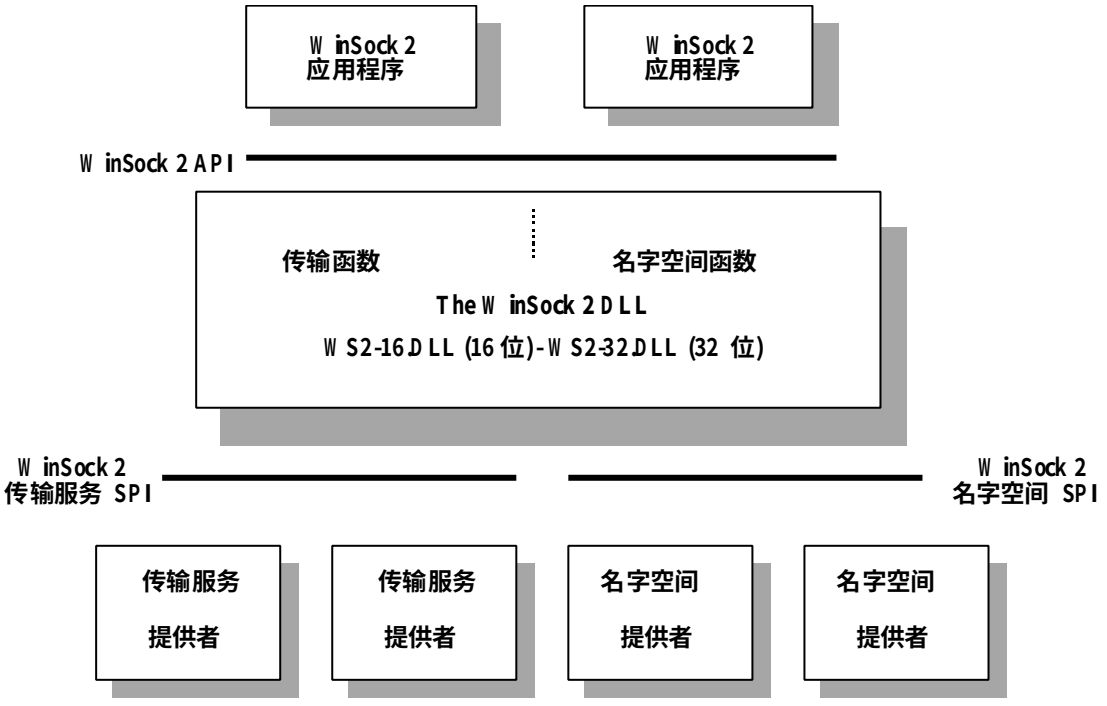

图 6-1：Windows Socket 2 开放系统结构图

注意：16 位的 Windows Sockets 2 应用程序应使用 WS2-16.DLL，而 32 位的 Windows Sockets 2 应用程序应使用 WS2-32.DLL。但今后，为了简单起见，它们将都使用 WINSOCK.DLL。这并不会造成任何问题，因为在它们之间并没有任何语

法上的区别。

由于以上的结构，现在已没有必要每个协议栈开发商都提供它们自己的WINSOCK.DLL（甚至这样做也不是期望的）。因为任何一个WINSOCK.DLL能够在所有协议栈上工作。因此，WINSOCK.DLL可以被看作是一个操作系统组件。Microsoft将在Windows 95和Windows NT上提供一个32位的WINSOCK.DLL。Intel公司目前正在打算提供Windows 3.1和Windows 3.11上的Windows Sockets 2兼容的16位WINSOCK.DLL。

6.2 与Windows Socket 1.1应用程序的向后兼容性

Windows Socket 2与Windows Sockets 1.1在两个基础上向后兼容：源码和二进制代码。这就实现了Windows Sockets应用程序和任何版本的Windows Sockets实现之间的最大的互操作性，而且也减少了Windows Sockets应用程序使用者，网络协议栈提供者和服务提供者的许多痛苦。现有的Windows Sockets 1.1兼容的应用程序可以在Windows Sockets 2实现上不加修改的运行，只要有一个TCP/IP协议栈被安装。

6.2.1 源码的兼容性

Windows Sockets 2中的源码兼容性意味着所有的Windows Sockets 1.1版的API在Windows Sockets 2中都被保留了下来。这意味着现有的Windows Sockets 1.1应用程序的原程序可以被简单的移植到Windows Sockets 2系统上运行。程序员需要做的只是包含新的头文件－WINSOCK2.H和简单的与合适的Windows Sockets 2函数库的连接。应用程序开发者应该把这种工作看作是完全转向Windows Sockets 2的第一步，因为有许多方式可以使用Windows Sockets 2中的新函数来提高原来的Windows Sockets 1.1应用程序的运行性能。

6.2.2 二进制兼容性

在设计Windows Sockets 2时的一个主要目标就是使得现有的Windows Sockets 1.1应用程序在二进制级别上能够不加修改的应用于Windows Sockets 2之上。由于Windows Sockets 1.1是基于TCP/IP上的，二进制兼容性就要求Windows Sockets 2系统提供基于TCP/IP上的Windows Sockets 2的传输和名字解析服务。为了Windows Sockets 1.1应用程序能在这种意义上运行，Windows Sockets 2系统提供了一个附加的组件－Windows Sockets 1.1的DLL。Windows Sockets 2安装时的提示保证了在终端机用户引进Windows Sockets 2系统时不会对已有的Windows Sockets软件环境有任何影响。

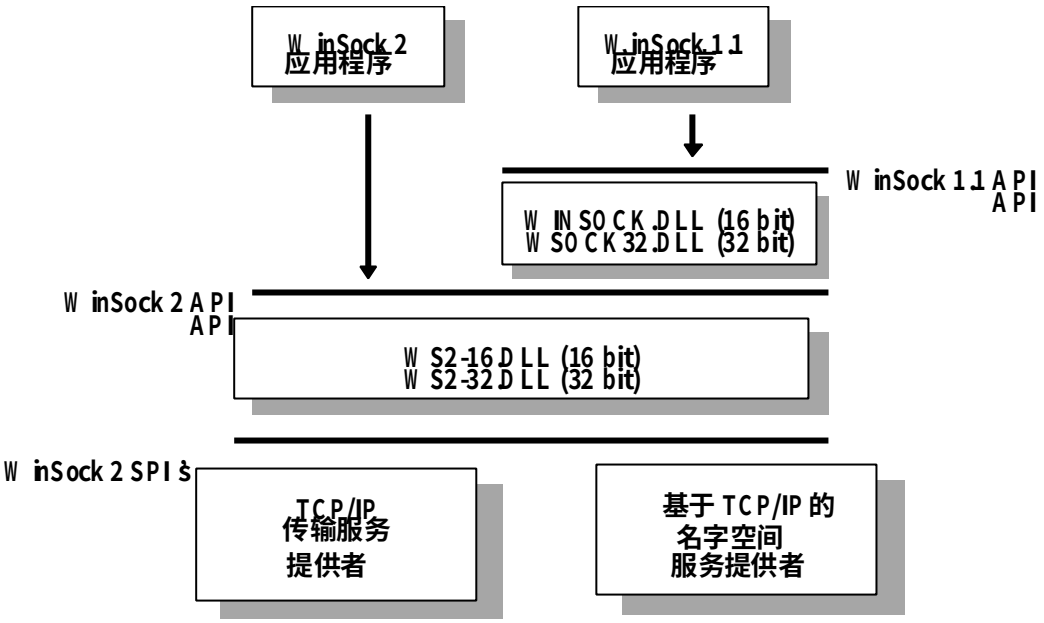

图 6-2：与 Windows Sockets 1.1 二进制兼容性结构图

一个完全的 Windows Sockets 1.1 二进制兼容的必要的前提是在系统上已经安装了至少一个 TCP/IP 协议栈并且在 Windows Sockets 2 中做了注册。Windows Sockets 1.1 目前通过 WSAData 结构中的某些元素来得到关于底层 TCP/IP 协议栈的信息（例如通过 WSAStartup()函数调用），这些信息包括 iMaxSockets，iMaxUdpDg 和 IPVendorInfo。但是在 Windows Sockets 2 中，应用程序应该知道忽略这些信息，因为这些值不能统一地适用于所有协议栈。不过 DLL 必须仍然提供这些值以免破坏 Windows Sockets 1.1 的应用程序。这些信息只能从（因此也只能应用于）缺省的 TCP/IP 服务提供者得到。缺省的 TCP/IP 服务提供者是由 WSAEnumProtocols()调用返回的 PROTOCOL_INFO 结构缓冲区的第一条 TCP/IP 协议栈。

6.3 在 Windows Sockets 中注册传输协议

要使 Windows Sockets 能够利用一个传输协议，该传输协议必须在系统上安装并且在 Windows Sockets 中注册。Windows Sockets 2 的 DLL 包含了一组 API 来完成这个注册过程。这个注册过程包括建立一个新的注册和取消一个已有的注册。在建立新的注册时，调用者（假设是协议栈开发商的安装程序）必须提供一组或多组完整的关于协议的信息，这些信息将被用来填充 PROTOCOL_INFO 结构。

6.3.1 使用多个协议

一个应用程序可以通过 WSAEnumProtocols()功能调用来得到目前有多少个传输协议可以使用，并且得到与每个传输协议相关的信息，这些信息包含在 PROTOCOL_INFO 结构中。然而，某些传输协议可能表现出多种行为。例如 SPX 是基于消息的（发送者发送的消息的边界在网络上被保留了），但是接收的一方可以选择忽略这些边界并把套接口作为一个字节流来对待。这样就很合理地导致了 SPX 有两个不同的 PROTOCOL_INFO 结构条目，每一个条目对应了一种行为。

在 Windows Sockets 1 中仅有一个地址族（AF_INET），它包含了数量不多的一些众所周知的套接口类型和协议标识符。这在 Windows Sockets 2 中已经有所改变。除了现有的地址族，套接口类型和协议标识符为了兼容性原因被保留以外，Windows Sockets 2 加入了许多唯一的但是可能并不为大家所知的地址族，套接口类型和协议标识符。不为大家所知并不意味着会对应用程序开发造成问题，因为一个企图做成协议无关的应用程序应该在对自身合适的基础上选择协议而不应该依赖于某个分配给它的特定的套接口类型或协议类型值。
PROTOCOL_INFO 结构中包含的通讯性质指明了协议的合适性（例如：基于消息的对应于基于字节流的，可靠的对应于不可靠的，等等）。基于合适性原则选取协议而不使用某个特定的协议名和套接口类型。

对于客户机/服务器模型，服务器一端的应用程序最好能够在所有合适的传输协议上建立监听套接口。这样，客户机一端的应用程序就可以通过任何合适的传输协议来与服务器一端的应用程序建立连接。这样做可以使得一个客户机应用程序易于移植。例如一台运行于 LAN 上的台式机的客户机应用程序在转到运行于无线网上的笔记本计算机时就不用作任何改变。

6.3.2 select()函数应用中关于多个服务提供者的限制

在 Windows Sockets 2 中，函数 select()使用 FD_SET 仅能应用于和单个服务提供者相连的套接口。但是这并不限制一个应用程序使用多个服务提供者打开多个套接口。如果应用程序开发者喜欢使用非阻塞方式编程，那么可以使用 WSAAsyncSelect()函数。由于该函数需要一个套接口描述字作为输入参数，那么与该套接口相连的服务提供者是很重要的。如果一个应用程序需要在一组跨越多个服务提供者的套接口上使用带有阻塞语法的函数，那么应该使用 WSAWaitForMultipleEvents()函数。应用程序也可以使用 WSAEventSelect()函数。该函数允许应用程序把 FD_XXX 网络事件和一个事件对象相连接，并且在该事件对象中处理网络事件（这一模式将在下文讨论）。

6.4 协议无关的名字解析

Windows Sockets 2 包含了应用程序可以使用的多种标准化的网络名字服务。Windows Sockets 2 应用程序并不需要理解与名字服务相关的许多迥异的接口（例如 DNS，NIS，X.5000，SAP 等等）。本书的 4.2 介绍了这一主题，并对 API 做了详细介绍。

6.5 重叠 I/O 和事件对象

Windows Sockets 2 引入了重叠 I/O 的概念并且要求所有的传输协议提供者都支持这一功能。重叠 I/O 仅能在由 WSASocket()函数打开的套接口上使用（使用 WSA_FLAG_OVERLAPPED 标记）。这种方式的使用将采用 Win32 建立的模型。

对于接收，应用程序使用 WSARecv()函数或 WSARecvFrom()函数来提供存放接收数据的缓冲区。如果数据在网络接收以前，应用程序已经提供了一个或多个数据缓冲区，那么接收的数据就可以立即被存放进用户缓冲区。这样可以省去使用 recv()函数和 recvfrom()函数时需要进行的拷贝工作。如果在应用程序提供数据缓冲区时已经有数据到来，那么接收的数据将被立即拷贝进用户缓冲区。如果数据到来时，应用程序没有提供接收缓冲区，那么网络将回到我们熟悉的同步操作方式－传送来的数据将被存放进内部缓冲区，直到应用程序发出了接收调用并且提供了接收缓冲区，这时接收的数据就被拷贝进接收缓冲区。这种做法会有一个例外：就是当应用程序使用 setsockopt()函数把接收缓冲区长度置为了 0。在这种情况下，对于可靠传输协议，只有在应用程序提供了接收数据缓冲区后，数据才会被接收；而对于不可靠传输协议，数据将会丢失。

对于发送的一方，应用程序使用 WSASend()函数或 WSASendTo()函数提供一个指向已填充的数据缓冲区的指针。应用程序不应在网络使用完该缓冲区的数据以前以任何方式破坏该缓冲区的数据。

重叠发送和接收调用会立即返回。如果返回值是 0，那么表明了 I/O 操作已经完成，对应的完成指示也已经可以得到。如果返回值是 SOCKET_ERROR，并且错误代码是 WSA_IO_PENDING，那么表明重叠操作已经被成功地初始化，今后发送缓冲区被用完或者接收缓冲区被填满时，将会有完成指示。任何其他的错误代码表明了初始化没有成功，今后也不会有什么完成指示。

发送操作和接收操作都可以被重叠使用。接收函数可以被多次调用，发出接收缓冲区，准备接收到来的数据。发送函数也可以被多次调用，组成一个发送缓冲区队列。要注意的是，应用程序可以通过按顺序提供发送缓冲区来确保一系列重叠发送操作的顺序，但是对应的完成指示有可能是按照另外的顺序排列的。同样的，在接收数据的一方，缓冲区是按照被提供的顺序填充的，但是完成指示也可能按照另外的顺序排列。

WSAIoctl()函数（ioctolsocket()函数的增强版本）还可以使用重叠 I/O 操

作的延迟完成特性。

6.5.1 事件对象

重叠 I/O 概念的引入需要建立一个机制使得应用程序能够正确的把发送和接收事件与今后它们完成时的指示相连接。在 Windows Sockets 2 中，这一点是通过事件对象实现的，它采用了 Win32 事件的模型。Windows Sockets 事件对象是一个相当简单的结构，它可以被创建，关闭，设置，清除，等待和检查。它们的主要用处是使得应用程序能够阻塞并等待直到一个或多个事件对象被设置。

应用程序可以使用 WSACreateEvent()函数来得到一个事件对象句柄，这个句柄可以作为以后的重叠发送和接收函数的输入参数（WSASend()，WSASendTo()，WSARecv()，WSARecvFrom()）。事件对象在创建时被清除，在相关的重叠 I/O 操作完成时由传输协议提供者设置(或者成功,或者出错)。每个被WSACreateEvent()函数创建的事件对象都必须有对应的 WSACloseEvent()函数释放它。

WSAEventSelect()函数把一个或多个 FD_XXX 网络事件与一个事件对象连接。这将在 2.6 中讨论。

在 32 位环境中，与事件对象相关的函数，包括 WSACreateEvent()，WSACloseEvent()，WSASetEvent()，WSAResetEvent()，WSAWaitForMultipleEvent()和 WSAGetOverlappedResult()，都被直接映射到对应的 Win32 函数，例如没有 WSA 前缀的同名函数。

6.5.2 接收操作完成指示

为了提供给应用程序适当的灵活性，Windows Sockets 2 为接收操作完成指示提供了多个选项。它们包括：等待（阻塞）事件对象，检查事件对象和套接口 I/O 完成例程。

6.5.2.1 阻塞并且等待完成指示。

应用程序可以使用 WSAWaitForMultipleEvents()函数来选择阻塞程序直到一个或多个事件对象被设置。在 Win16 实现中，这种方式将使用一个阻塞钩子，就象在标准的阻塞套接口操作时一样。在 Win32 实现中，进程或线程会被真正地阻塞。因为 Windows Sockets 2 事件对象被实现成 Win32 事件，所以 Win32 函数 WaitForMultipleObjects()也可以使用。这在线程需要阻塞套接口和非套接口事件时将会非常有用处。

6.5.2.2 检查完成指示

应用程序如果不希望使用阻塞方式，它可以使用WSAGetOverlappedResults()函数来检查与某个特定的事件对象相连的完成状态。该函数检查重叠操作是否完成，如果完成的话，处理重叠操作的出错信息，使得该信息在WSAGetLastError()函数调用时可以得到。

6.5.2.3 使用套接口 I/O 操作完成例程

所有的用来初始化重叠 I/O 操作的函数（WSASend()，WSASentTo()，WSARecv()，WSARecvFrom()）都把 lpCompletionRoutine 作为输入参数。这是一个应用程序定义的函数指针，在重叠 I/O 操作完成时可以被调用。

在 Win16 环境中，回调函数有可能在 VMM 环境（有时也被称作中断环境）下被激活。传送对时间要求较高的数据（例如视频或音频数据）在这种低延时，占先方式下接受这种指示会很方便。但是应用程序必须知道，在这种特殊情况下，只要很少一部分运行时和 Windows 库函数可以被调用。作为一条规则，应用程序必须把自己限制在一套 Windows 文挡说明的在一个多媒体定时回调函数中可以被安全调用的运行时函数库。

在 Windows 95 和 Windows NT 中，完成例程与 Win32 文件 I/O 完成例程遵循着同样的规则。在所有的环境中，传输协议允许应用程序以完成例程的方式唤起发送和接收操作，而且保证对于给定的一个套接口，I/O 完成例程不会嵌套。这就允许了在一个占先的环境中进行对时间敏感的数据的传送。

6.5.3 WSAOVERLAPPED 的细节

WSAOVERLAPPED 结构提供了一个重叠 I/O 操作的初始化和它将来的如何被完成之间的通讯媒体。WSAOVERLAPPED 结构被设计成与 Win32 中的 OVERLAPPED 结构兼容：

```
typedef struct WSAOVERLAPPED {
    DWORD       Internal;         // reserved
    DWORD       InternalHigh;     // reserved
    DWORD       Offset;           // ignored
    DWORD       OffsetHigh;       // ignored
    WSAEVENT    hEvent;
} WSAOVERLAPPED, LPWSAOVERLAPPED;
```

Internal　　这一个保留的域是由重叠 I/O 实现的实体内部使用的。对

于使用类文件方式创建套接口的传输服务提供者，这一域是被底层的操作系统使用的；对于其他的传输服务提供者（那些创建伪句柄的），可以视需要使用这个域。

InternalHigh　这一个保留的域是由重叠 I/O 实现的实体内部使用的。对于使用类文件方式创建套接口的传输服务提供者，这一域是被底层的操作系统使用的；对于其他的传输服务提供者（那些创建伪句柄的），可以视需要使用这个域。

Offset　由于套接口没有文件偏移量的概念，应用程序可以视需要使用这个域。

OffsetHigh　由于套接口没有文件偏移量的概念，应用程序可以视需要使用这个域。

hEvent　如果一个重叠的 I/O 操作在被调用时没有使用 I/O 操作完成例程（lpCompletionRoutine 为空指针），那么这个域必须包含一个有效的 WSAEVENT 对象的句柄，否则（lpCompletionRoutine 不为空指针），应用程序可以视需要使用这个域。

6.6 使用事件对象异步通知

为了适应一些应用程序例如精灵程序或者某些没有用户界面的服务程序（因此不使用窗口句柄），Windows Socket 2 提供了 WSAEventSelect()函数和 WSAEnumNetworkEvents()函数。WSAEventSelect()函数和 WSAAyncSelect()函数很类似，区别仅在于当一个 FD_XXX 网络事件发生时，WSAEventSelect()函数将导致一个应用程序指定的事件对象被设置，而 WSAAyncSelect()将导致一条 Windows 消息被发送（例如 FD_READ，FD_WRITE 等等）。

此外，传输服务提供者会记住每个特定的 FD_XXX 网络事件的发生。应用程序可以调用 WSAEnumNetworkEvents()函数把目前的网络事件记忆拷贝到应用程序提供的缓冲区中，并且自动清除网络事件记忆。如果需要，应用程序还可以把某个特定的事件对象和网络事件记忆一起清除。

6.7 服务的质量（QOS）

Windows Sockets 2 中的 QOS 机制是从 Craig Partridge 在 RFC 1363 中描述的流规格引入的。这一概念可以大致描述如下：

流规格描述了一个网络上单向数据流的性质的集合。应用程序可以在调用WSAConnect()函数发出连接请求或者使用WSAIoctl()函数等其他QOS命令时，把一对流规格和一个套接口连接（一个规范对应了一个方向）。流规格以参数方式声明了应用程序所要求的服务的级别，并且为应用程序适应不同的网络条件提供了一套反馈机制－如果应用程序要求的服务级别不能达到，应用程序是否愿意松动它的要求。

Windows Sockets 2中QOS的使用模型如下：

对于基于连接的传输服务，应用程序可以很方便的在使用WSAConnect()函数提出连接请求时规定它所要求的服务质量（QOS）。要注意的是：如果应用程序在调用WSAConnect()时QOS参数不为空，那么对于基于连接的套接口，任何预先设置的QOS都会被覆盖。如果WSAConnect()函数成功返回，应用程序就会知道它所要求的QOS已经被网络接受，那么应用程序就可以随意的使用这个套接口进行数据交换。如果连接操作由于资源有限而失败，应用程序应该适当地降低它所要求的服务质量或者干脆就放弃操作。

在每次连接企图之后（不论成功与否），传输服务提供者都会更新flow_spec结构，以便尽可能地指明目前的网络条件。如果应用程序所要求的服务质量仅仅包含了一些传输服务提供者必须满足的缺省值，那么这种更新会是很有用处的。应用程序可以利用这些关于当前网络条件的信息来指导自己使用网络，例如今后的QOS要求。然而应用程序应该注意的是，传输服务提供者在不断更新的flow_spec结构中提供的信息仅仅是一个参考，它们只不过是粗略的估计。应用程序应该很小心的解释这些数据。

无连接的套接口也可以使用WSAConnect()函数为一个指定的通讯规定特定的QOS级别。WSAIoctl()函数也可以用来规定初始的QOS要求，或者用来今后的QOS协商。

即使是一个流规格已经建立，网络的情况也有可能改变，或者通讯的一方可能提出了QOS重协商的要求，这将导致可以得到的服务级别的降低或者提高。Windows Sockets 2引入了一个通知机制。它使用了一般的WS通知方式（FD_QOS和FD_GROUP_QOS事件）来告诉应用程序QOS级别已经改变了。一般服务提供者只在当前的服务级别和上一次报告有很大区别（通常是逆向的），并且有可能会影响到应用程序时才发出FD_QOS/FD_GROUP_QOS通知。应用程序应该使用WSAIoctl()函数来得到当前的状态并且检查服务等级的那些方面有了变化。如果当前的QOS级别是不可接受的，应用程序应该调整自己以去适应当前的状态，试图重新协商或者关闭套接口。

Windows Sockets 2推荐的流规格把QOS特性划分为如下几个方面：

1．源通讯描述：应用程序的通讯事件以什么方式被送入网络。

2．延时性：最大延时和可接受的延时变化。

3．需要保证的服务级别：应用程序是否要求对服务质量的绝对保证。

4．费用：这一项是为将来可以决定有意义的费用时保留的。

5．服务提供者特定的参数：流规格可以根据具体的提供者扩展。

6.8 套接口组

Windows Sockets 2 引入了一个所谓套接口组的概念。它允许应用程序（或者一组共同工作的应用程序）通知底层的服务提供者一组特定的套接口是相关的，它们享有一些特定的性质。组的特性包括了组内单个套接口之间的相关特性和整个组的服务规范的特性。

需要在网络上传输多媒体数据的应用程序会因为在所使用的一组套接口上建立联系而得到好处。至少这可以告诉服务提供者正在传输的数据流的一些相关性质。例如，一个会议应用程序希望传送音频数据的套接口比传送视频数据的套接口有更高的优先级。此外，一些传输服务提供者（例如数字电话和 ATM）可以利用服务规范的组特性来决定底层调用或者线路连接的性质。通过应用程序指明套接口组及其特性，服务提供者可以以最大效率应用这些套接口。

WSASocket()函数和 WSAAccept()函数可以用来在创建一个新的套接口的同时显式的创建或者加入套接口组。getsockopt()函数可以用来得到套接口所属套接口组的标志。

6.9 共享套接口

为了在进程间共享套接口，Windows Sockets 2 引入了 WSADuplicateSocket()函数。共享套接口是通过对底层的套接口创建附加的套接口描述字实现的。该函数的输入是本地的套接口描述字和目标进程的句柄。它返回一个仅在目标进程中有效的新的套接口描述字（目标进程有可能就是原始进程）。这一机制既可以在单线程 Windows 版本(例如 Windows 3.1)中使用，也可以在占先的多线程 Windows 版本（例如 Windows 95 和 Windows NT）中使用。要注意的是，套接口可以在一个进程的不同线程中共享而不需要使用 WSADuplicateSocket()函数，因为一个套接口描述字在进程的所有线程中都有效。

基于一个共享套接口的两个或者单个套接口描述字应该独立地使用套接口 I/O。然而 Windows Sockets 没有实现任何共享控制。因此，在一个共享套接口上协调它们的操作是应用程序的责任。一个典型的使用共享套接口的例子是，有一个进程专门负责创建套接口和建立连接，并把套接口交给其他负责信息交换的进程。由于重新创建的是套接口描述字而不是底层的套接口，所以一切与套接口相关的状态对于所有套接口描述字都是相同的。例如对一个套接口描述字应用 setsockopt()操作后，对所有的套接口描述字应用 getsockopt()操作都可以看到这一变化。一个进程有可能调用 closesocket()函数关闭一个复制的套接口描述字，于是该描述字就被清除了，然而，底层的套接口并不会被关闭，底层的套

接口将一直保持打开，直到最后的一个套接口描述字被关闭。

选择对共享套接口的通知可以使用 WSAAsyncSelect()函数和 WSAEventSelect()函数。对任何共享的套接口描述字发出这些调用将会取消在这一套接口上的所有注册事件，无论先前的注册使用了那个套接口描述字。因此，如果应用程序想使进程 A 接收 FD_READ 事件，进程 B 接收 FD_WRITE 事件，这是做不到的。如果应用程序确实需要使用这种紧密的协调方式，我们建议应用程序开发者使用线程而不要使用进程。

6.10 连接建立和拆除的高级函数

WSAAccept()函数允许应用程序在接受连接请求以前得到请求者的信息，例如请求者的 ID，QOS 等等。这一点是通过对一个应用程序提供的条件函数的回调来实现的。如果服务提供者支持的话，在 WSAConnect()函数的参数或者 WSAAccept()函数的条件函数说明的用户对用户的数据可以在连接建立的时候传送到对方。

在连接拆除时，如果协议支持的话，也可以在通讯的端点间交换用户数据。需要提出拆除连接的通讯端点可以调用 WSASendDisconnect()函数声明没有要传送的数据并启动连接拆除过程。对于某些协议，这一拆除过程包括了从发起拆除连接的一方发送拆除数据。在接收到远端已经启动了连接拆除过程的通知后（通常是 FD_CLOSE），应用程序可以调用 WSARecvDisconnect()函数接收某些拆除数据。

为了解释如何使用拆除数据，我们考虑如下的场景：在客户机/服务器模型中，通常是由客户机决定何时终止套接口的连接。在终止连接的同时，它通过拆除数据提供和服务器连接的次数。服务器也提供它和所有客户机的总的连接次数。这一过程如下所示：

客户机端	服务器端
(1) 调用 WSASendDisconnect()函数终止对话并提供总的交互次数。	
	(2) 得到 FD_CLOSE，recv()函数返回 0，或者 WSARecv()函数返回 WSAEDISCON 错误表示优雅的关闭。
	(3) 调用 WSARecvDisconnect()函数来得到客户机的总的交互次数。
	(4) 计算累积授权次数。
	(5) 调用 WSASendDisconnect()函数来传送累积授权次数。

(6) 接收 FD_CLOSE 指示　　(5') 调用 closesocket()函数。
(7) 调用 WSARecvDisconnect()
函数来接收并存放累积授权次数。
(8) 调用 closesocket()函数。

注意：步骤(5')必须在步骤(5)之后执行，但是与步骤(6)，(7)或(8)没有时间联系.

6.11 扩展的字节顺序转换例程

Windows Sockets 2 并不假设对于所有协议，网络字节顺序都是正确的。所以 Windows Sockets 2 提供了一套把 16 位或者 32 位数字转换到网络字节顺序或者从网络字节顺序转换的例程。这些例程通常有一个整型的输入参数，它通常是一个常量，指明了需要的网络字节顺序是什么（目前或者是 big_endian，或者是 little_endian）。同时，每个协议的 PROTOCOL_INFO 结构包含一个域指明了协议对应的网络字节顺序，它可以用来作为字节顺序转换例程的输入参数。

6.12 分散/聚集方式 I/O

WSASend()，WSASendTo()，WSARecv()和 WSARecvFrom()函数都以应用程序缓冲区数组作为输入参数，因此它们可以进行分散/聚集方式（向量方式）的 I/O 操作。如果应用程序需要传送的信息除了信息体外还包含了一个或多个固定长度的头时，这种操作是很有用的。这些头在发送之前不需要由应用程序连接到一个连续的缓冲区中。同样的，在接收时，这些头会自动的分离到各自的缓冲区中。

如果接收时应用程序提供了多个缓冲区，当有数据到来时，操作就结束了，不论提供的缓冲区是否都被使用了。

6.13 协议无关的多点通讯

Windows Sockets 2 支持基本的数据传输以一般的方式使用不同的传输协议。Windows Sockets 2 也支持应用程序以一般的方式使用传输协议的多点通讯能力。

目前的多点通讯实现在节点加入一个多点对话的方式上有很大的不同。例如，是否有一个特殊的节点被指定为中心节点或者就是根节点；数据是在所有节点之间交换还是只在根节点和它的叶节点之间交换。Windows Sockets 2 中的 PROTOCOL_INFO 结构允许一个协议声明它的多点通讯的各种特性。通过检查这些特性，应用程序可以知道在使用 Windows Sockets 2 函数设置，使用和拆除多点

对话时应该遵循那一种协定。

Windows Sockets 2 中为支持多点通讯而作的增加如下：

* PROTOCOL_INFO 结构中的两个特性位。
* 为 WSASocket()的参数 iflags 定义的四个标志。
* 一个新函数－WSAJoinLeaf()，它用来在多点对话中加入一个叶节点。
* 两个 WSAIoctl()的命令代码。

6.14 新增套接口选项一览

Windows Sockets 2 新增的套接口选项归纳如下：

选项值	类型	含义	缺省值
SO_GROUP_ID	GROUP	套接口所属的套接口组	NULL
SO_GROUP_PRIORITY	int	套接口在套接口组中的相对优先级	0
SO_MAX_MSG_SIZE	int	对于基于消息的套接口，这一选项指明了一个消息的最大长度。对于基于流的套接口，这一选项没有任何意义。	决定于实现
SO_PROTOCOL_INFO	struct PROTOCOL_INFO	描述捆绑到套接口的协议的信息。	决定于协议
PVD_CONFIG	char FAR *	一个包含了服务提供者配置信息的不透明的数据结构对象。	决定于实现

6.15 新增套接口 ioctl 操作代码

Windows Sockets 2 新增的 ioctl 操作代码归纳如下。WSAIoctol()函数支持所有为 ioctlsocket()函数定义的操作代码。

操作代码	输入类型	输出类型	含义
SIO_ASSOCIATE_HANDLE	决定于伴随的 API。	没有使用。	把套接口与一个指定的伴随接口连接。
SIO_ENABLE_CIRCULAR_QUEUEING	没有使用	没有使用	允许循环队列。
SIO_FIND_ROUTE	struct sockaddr	没有使用	请求找到对应于指定地址的例程。
SIO_FLUSH	没有使用	没有使用	废除当前发送队列的内容。
SIO_GET_BROADCAST_址，	没有使用	没有使用	得到特定协议的广播地 该地址可以使用在 send()函数和 WSASend()函数中。
SIO_GET_QOS	没有使用	QOS	得到套接口当前的流协议。
SIO_GET_GROUP_QOS的	没有使用	QOS	得到套接口所属套接口组 流协议。
SIO_MULTIPOINT_LOOK由	BOOL	没有使用	决定多点对话的数据是否 本地主机的同一套接口接收。
SIO_MULTICAST_SCOPE间。	int	没有使用	定义允许多方传送的空
SIO_SET_QOS	QOS	没有使用	为套接口建立新的流协议。
SIO_SET_GROUP_QOS立	QOS	没有使用	为套接口所属套接口组建

新的流协议。

SIO_TRANSLATE_HANDLE int 决定于伴 得到一个上下文有效的套接口
随 API 对应的句柄。

6.16 新增函数一览

Windows Sockets 2 新增的函数列在下表中：

WSAAccept()	accept()函数的扩展版本，它支持条件接收和套接口分组。
WSACloseEvent()	释放一个事件对象。
WSAConnect()	connect()函数的扩展版本，它支持连接数据交换和 QOS 规范。
WSACreateEvent()	创建一个事件对象。
WSADuplicateSocket()	为一个共享套接口创建一个新的套接口描述字。
WSAEnumNetworkEvents()	检查是否有网络事件发生。
WSAEnumProtocols()	得到每个可以使用的协议的信息。
WSAEventSelect()	把网络事件和一个事件对象连接。
WSAGetOverlappedResu()	得到重叠操作的完成状态。
WSAGetQOSByName()	对于一个传输协议服务名字提供相应的 QOS 参数。
WSAHtonl()	htonl()函数的扩展版本。
WSAHtons()	htons()函数的扩展版本。
WSAIoctl()	ioctlsocket()函数的允许重叠操作的版本。
WSAJoinLeaf()	在多点对话中加入一个叶节点。
WSANtohl()	ntohl()函数的扩展版本。
WSANtohs()	ntohs()函数的扩展版本。
WSARecv()	recv()函数的扩展版本。它支持分散/聚集 I/O 和重叠套接口操作。
WSARecvDisconnect()	终止套接口的接收操作。如果套接口是基于连接的，得到拆除数据。
WSARecvFrom()	recvfrom()函数的扩展版本。它支持分散/聚集 I/O 和重叠套接口操作。
WSAResetEvent()	重新初始化一个数据对象。
WSASend()	send()函数的或者版本。它支持分散/聚集 I/O

和重叠套接口操作。

WSASendDisconnect()	启动一系列拆除套接口连接的操作，并且可以选择发送拆除数据。
WSASendTo()	sendto()函数的扩展版本。它支持分散/聚集I/O和重叠套接口操作。
WSASetEvent()	设置一个数据对象。
WSASocket()	socket()函数的扩展版本。它以一个PROTOCOL_INFO结构作为输入参数，并且允许创建重叠套接口。它还允许创建套接口组。
WSAWaitForMultipleEvents()	阻塞多个事件对象。

第七章 Windows Sockets 2 扩展库函数简要参考

7.1 WSAAccept()

简述：根据条件函数的返回值有条件地接受连接，同时（可选地）创建和/或加入一个套接口组。

```
SOCKET WSAAPI WSAAccept ( SOCKET s, struct
sockaddr FAR * addr, int FAR * addrlen,
LPCONDITIONPROC lpfnCondition, DWORD
dwCallbackData );
```

s：标识一个套接口的描述字，该套接口在 listen()后监听连接。
addr：（可选）指针，指向存放通讯层所知的连接实体地址的缓冲区。addr 参数的具体格式由套接口创建时产生的地址族决定。
addrlen：（可选）指针，指向存放 addr 地址长度的整形数。
lpfnCondition：（可选的）用户提供的条件函数的进程实例地址。该函数根据参数传入的调用者信息作出接受或拒绝的决定，并通过给结果参数赋予特定的值来（可选地）创建和/或加入一个套接口组。
dwCallbackData：作为条件函数参数返回给应用程序的回调数据。WinSock 不分析该参数。

返回值：
　若无错误发生，WSAAccept()函数返回所接受套接口的描述字。否则的话，将返回 INVALID_SOCKET 错误，应用程序可通过 WSAGetLastError()来获取相应的错误代码。
　addrlen 参数引用的整形数初始时包含了 addr 参数所指向的空间数，在调用返回时包含了返回地址的实际长度。

错误代码：
WSANOTINITIALISED　　在调用本 API 之前应成功调用 WSAStartup()。
WSAECONNREFUSED　　根据条件函数的返回值（CF_REJECT）强制拒绝连接请求。
WSAENETDOWN　　网络子系统失效。
WSAEFAULT　　addrlen 参数太小（小于 sockaddr 结构的大小），或者

lpfnCondition 并不是用户空间的一部分。
WSAEINTR　　通过 WSACancelBlockingCall()函数取消（阻塞）调用。
WSAEINPROGRESS　　一个阻塞 WinSock 调用正在进行。
WSAEINVAL　　WSAAccept()调用前未执行 listen()调用；条件函数中的 g 参数非法；条件函数的返回值非法；套接口处于非法状态。
WSAEMFILE　　WSAAccept()调用时排队队列非空，且无可用套接口描述字。
WSAENOBUFS　　无可用缓冲区空间。
WSAENOTSOCK　　描述字不是一个套接口。
WSAEOPNOTSUPP　　所引用的套接口不是支持面向连接服务类型的。
WSATRY_AGAIN　　根据条件函数的返回值(CF_DEFER) ，连接请求被推迟。
WSAEWOULDBLOCK　　套接口标志为非阻塞，无连接请求供接受。
WSAEACCES　　被推迟的连接请求超时或撤销。

另请参阅：accept(), bind(), connect(), getsockopt(),listen(), select(), socket(), SAAsyncSelect(), WSAConnect().

7.2 WSACloseEvent()

简述：关闭一个开放的事件对象句柄。

```
#include <winsock2.h>

BOOL WSAAPI WSACloseEvent( WSAEVENT hEvent );
```

hEvent：标识一个开放的事件对象句柄。

返回值：
　如果函数顺利完成，返回值为真 TRUE。如果失败，返回值为假 FALSE。可用 WSAGetLastError()调用获取更多的错误信息。

错误代码：
WSANOTINITIALISED　　在调用本 API 之前应成功调用 WSAStartup()。
WSAENETDOWN　　网络子系统失效。
WSA_INVALID_HANDLE　　hEvent 不是一个合法的事件对象句柄。

另请参阅： WSACreateEvent(), WSAEnumNetworkEvents(),WSAEventSelect(),

WSAGetOverlappedResult(),WSARecv(), WSARecvFrom(),
WSAResetEvent(),WSASend(), WSASendTo(),
WSASetEvent(),WSAWaitForMultipleEvents().

7.3 WSAConnect()

简述：创建一个与远端的连接，交换连接数据，并根据所提供的流描述确定所需的服务质量。

```
#include <winsock2.h>

int WSAAPI WSAConnect ( SOCKET s, const struct
sockaddr FAR * name,
int namelen, LPWSABUF lpCallerData, LPWSABUF
lpCalleeData,
 LPQOS lpSQOS, LPQOS lpGQOS );
```

s：用于描述一个未连接套接口的描述字。
name：欲与套接口连接的远端名字。
namelen：名字长度。
lpCallerData：指向用户数据的指针，该数据在建立连接时将传送到远端。
lpCalleeData：指向用户数据的指针，该数据在建立连接时将从远端传送回本机。
lpSQOS：指向套接口 s 流描述的指针，每个方向一个。
lpGQOS：指向套接口组流描述的指针。（如果有套接口组的话）

返回值：
　如果无错误发生，WSAConnect()返回 0。否则的话，将返回 INVALID_SOCKET 错误，应用程序可通过 WSAGetLastError()来获取相应的错误代码。
　对于阻塞套接口来说，返回值表示连接试图是否成功。
　对于非阻塞套接口来说，连接试图不一定马上完成。在这种情况下，WSAConnect()返回 SOCKET_ERROR，且 WSAGetLastError()返回 WSAEWOULDBLOCK.此时应用程序可以：
　1。利用 select()函数，通过检查套接口是否可写来判断连接请求是否完成。或者，
　2。如果应用程序已使用 WSAAsyncSelect()函数来确定对连接事件的兴趣，则

当连接操作完成时应用程序将收到 FD _CONNECT 通知。或者，

3。如果应用程序已使用 WSAEventSelect()函数来确定对连接事件的兴趣，则当连接操作完成时相应的事件对象将设置信号。

对于一个非阻塞套接口来说，在连接试图完成之前，任何对该套接口的 WSAConnect()调用都将以 WSAEALREADY 错误失败。

如果返回值指出连接试图失败（例如 WSAECONNREFUSED，WSAENETUNREACH，WSAETIMEDOUT）则应用程序可对该套接口再次调用 WSAConnect()函数。

错误代码：

WSANOTINITIALISED　在调用本 API 之前应成功调用 WSAStartup()。

WSAENETDOWN　网络子系统失效。

WSAEADDRINUSE　所指地址已被使用。

WSAEINTR　通过 WSACancelBlockingCall()函数中止了阻塞调用。

WSAEINPROGRESS　一个阻塞的 WinSock 调用正在进行中，或者服务提供者仍在处理一个回调函数。（参见 B.3.6.6 节）

WSAEALREADY　在所指定的套接口上正在进行一个非阻塞的 connect()或 WSAConnect()调用。

WSAEADDRNOTAVAIL　本地机器上无法获得所指定的地址。

WSAEAFNOSUPPORT　所指定地址族中的地址无法与本套接口一起使用。

WSAECONNREFUSED　连接试图被拒绝。

WSAEFAULT　name 或 namelen 参数不是用户地址空间的一个有效部分；namelen 参数太小； lpCalleeData、 lpSQOS 和 lpGQOS 的缓冲区太小；或者 lpCallerData 的缓冲区太大。

WSAEINVAL　套接口已与一个地址捆绑。

WSAEINVAL　套接口未与一个地址捆绑。

WSAEINVAL　s 参数为监听套接口。

WSAEISCONN　套接口已经连接（仅适用于面向连接的套接口）。

WSAENETUNREACH　当前无法从本主机联系网络。

WSAENOBUFS　无可用缓冲区，套接口未连接。

WSAENOTSOCK　描述字不是一个套接口。

WSAEOPNOTSUPP　lpSQOS 和 lpGQOS 中的流描述无法满足。

WSAEPROTONOSUPPORT　服务提供者不支持 lpCallerData 参数。

WSAETIMEDOUT　连接试图超时，连接未建立。

WSAEWOULDBLOCK　套接口标志为非阻塞，连接无法立即完成。当套接口用 select()函数设置为读时，可调用 select()。

WSAEACCES　由于 setsockopt()时未允许 SO_BROADCAST，无法将一个数据报套接口与一个广播地址连接。

另请参阅： accept(), bind(), connect(), getsockname(),getsockopt(), socket(), select(),
WSAAsyncSelect(), WSAEventSelect().

7.4 WSACreateEvent()

简述：创建一个新的事件对象。

```
#include <winsock2.h>

WSAEVENT WSAAPI WSACreateEvent( VOID );
```

返回值：
如果函数成功，则返回值即是事件对象的句柄。
如果函数失败，返回 WSA_INVALID_EVENT。应用程序可通过调用 WSAGetLastError()函数获取进一步的错误信息。

错误代码：

WSANOTINITIALISED	在调用本 API 之前应成功调用 WSAStartup()。
WSAENETDOWN	网络子系统失效。
WSA_NOT_ENOUGH_MEMORY	无足够内存创建事件对象。

另请参阅： WSACloseEvent(), WSAEnumNetworkEvents(),WSAEventSelect(), WSAGetOverlappedResult(),WSARecv(), WSARecvFrom(), WSAResetEvent(),WSASend(), WSASendTo(), WSASetEvent(),WSAWaitForMultipleEvents().

7.5 WSADuplicateSocket()

简述：为一个共享套接口创建一个新的描述字。

```
#include <winsock2.h>

SOCKET WSAAPI WSADuplicateSocket ( SOCKET s,
WSATASK hTargetTask );
```

s：指定本地套接口描述字。
hTargetTask：指定使用共享套接口的目标任务的句柄。

返回值：
若无错误发生，WSADuplicateSocket()返回新的套接口描述字。否则的话，将返回 INVALID_SOCKET 错误，应用程序可通过 WSAGetLastError()来获取相应的错误代码。

错误代码：

WSANOTINITIALISED	在调用本 API 之前应成功调用 WSAStartup()。
WSAENETDOWN	网络子系统失效。
WSAEINVAL	参数中有非法值。
WSAEINPROGRESS	一个阻塞的 WinSock 调用正在进行中，或者服务提供者仍在处理一个回调函数。（参见 B.3.6.6 节）
WSAEMFILE	无可用套接口描述字。
WSAENOBUFS	无可用缓冲区空间，套接口未创建。
WSAENOTSOCK	描述字不是一个套接口。

另请参阅：

7.6 WSAEnumNetworkEvents()

简述：检测所指定套接口上网络事件的发生。

```
#include <winsock2.h>

int WSAAPI WSAEnumNetworkEvents ( SOCKET s,
WSAEVENT hEventObject, LPWSANETWORKEVENTS
lpNetworkEvents,  LPINT lpiCount);
```

s：标识套接口的描述字。
hEventObject：（可选）句柄，用于标识需要复位的相应事件对象。
lpNetworkEvents：一个 WSANETWORKEVENTS 结构的数组，每一个元素记录了一个网络事件和相应的错误代码。
lpiCount：数组中的元素数目。在返回时，本参数表示数组中的实际元素数目；如果返回值是 WSAENOBUFS，则表示为获取所有网络事件所需的元素数目。

返回值：
　如果操作成功则返回 0。否则的话，将返回 INVALID_SOCKET 错误，应用程序可通过 WSAGetLastError()来获取相应的错误代码。

错误代码：
WSANOTINITIALISED　　在调用本 API 之前应成功调用 WSAStartup()。
WSAENETDOWN　　网络子系统失效。
WSAEINVAL　　参数中有非法值。
WSAEINPROGRESS　　一个阻塞的 WinSock 调用正在进行中，或者服务提供者仍在处理一个回调函数。（参见 B.3.6.6 节）
WSAENOBUFS　　所提供的缓冲区太小。

另请参阅：　WSAEventSelect()

7.7 WSAEnumProtocols()

简述：获取现有传送协议的相关信息。

```
#include <winsock2.h>

int WSAAPI WSAEnumProtocols ( LPDWORD
lpdwProtocols, LPVOID lpProtocolBuffer, LPDWORD
lpdwBufferLength);
```

lpdwProtocols：一个以 NULL 结尾的协议标识号数组。本参数可选；如果 lpdwProtocols 为 NULL，则返回所有可用协议的信息，否则的话只返回数组中所开列的协议信息。
lpProtocolBuffer：一个用 PROTOCOL_INFO 结构填充的缓冲区。参见下文中对 PROTOCOL_INFO 结构的具体描述。
lpdwBufferLength：输入时，存有传递给 WSAEnumProtocols()函数的 lpProtocolBuffer 缓冲区长度。输出时，表示为获取所有信息需传递给 WSAEnumProtocols()函数的缓冲区长度。本函数不能重复调用；传入的缓冲区必须足够大以能存放所有的元素。这个规定降低了该函数的复杂度。由于一个机器上装载的协议数目往往是很小的，所以并不会产生问题。

返回值：

若无错误发生，WSAEnumProtocols()返回协议的数目。否则的话，将返回INVALID_SOCKET 错误，应用程序可通过 WSAGetLastError()来获取相应的错误代码。

错误代码：

WSANOTINITIALISED　　在调用本 API 之前应成功调用 WSAStartup()。

WSAENETDOWN　　网络子系统失效。

WSAEINPROGRESS　　一个阻塞 WinSock 调用正在进行。

WSAEINVAL　　参数中有非法值。

WSAENOBUFS　　缓冲区太小，无法保存所有 PROTOCOL_INFO 结构及其相关信息。传入的缓冲区大小至少应等于 lpdwBufferLength 中返回的值。

7.8 WSAEventSelect()

简述：确定与所提供的 FD_XXX 网络事件集合相关的一个事件对象。

```
#include <winsock2.h>

int WSAAPI WSAEventSelect ( SOCKET s, WSAEVENT
hEventObject, long lNetworkEvents );
```

s：一个标识套接口的描述字。

hEventObject：一个句柄，用于标识与所提供的 FD_XXX 网络事件集合相关的一个事件对象。

lNetworkEvents：一个屏蔽位，用于指定感兴趣的 FD_XXX 网络事件组合。

返回值：

如果应用程序指定的网络事件及其相应的事件对象成功设置，则返回 0。否则的话，将返回 INVALID_SOCKET 错误，应用程序可通过 WSAGetLastError()来获取相应的错误代码。

在使用 select()和 WSAAsyncSelect()函数时，WSAEventSelect()常用来决定何时进行数据传送操作（如 send()或 recv()），并期望能立即成功。但是一个稳定的应用程序应该做好这样的准备，即事件对象被设置，并且一个 WinSock 调用以 WSAEWOULDBLOCK 立即返回 。举例来说，有可能发生下述操作序列：

(i) 套接口 s 上到达数据；WinSock 设置了 WSAEventSelect 事件对象。

(ii) 应用程序进行其他操作。
(iii) 在进行操作时，应用程序调用了 ioctlsocket(s, FIONREAD...)并发现有数据可读。
(iv) 应用程序调用一个 recv(s,...)来读取数据。
(v) 最后应用程序等待 WSAEventSelect()所指定的数据对象，该数据对象指出数据可读。
(vi) 应用程序调用 recv(s,...)，但以 WSAEWOULDBLOCK 错误失败。

其他的操作序列也是可能的。

成功地记录了网络事件的发生（通过设置内部网络事件记录的相应位），并且将相应的事件对象设置了信号后，不会对该网络事件作进一步的操作，直到应用程序调用了相应的函数显式地重新允许该网络事件及相应事件对象的信号。

网络事件	重新允许函数
FD_READ	recv() 或 recvfrom()
FD_WRITE	send() 或 sendto()
FD_OOB	recv()
FD_ACCEPT	accept() 或 WSAAccept()，直到返回的错误代码为 WSATRY_AGAIN，指明条件函数返回 CF_DEFER。
FD_CONNECT	NONE
FD_CLOSE	NONE
FD_QOS	用 SIO_GET_QOS 命令调用 WSAIoctl()。
FD_GROUP_QOS	用 SIO_GET_GROUP_QOS 命令调用 WSAIoctl()。

错误代码：

WSANOTINITIALISED	在调用本 API 之前应成功调用 WSAStartup()。
WSAENETDOWN	网络子系统失效。
WSAEINVAL	参数中有非法值，或者指定的套接口处于非法状态。
WSAEINPROGRESS	一个阻塞的 WinSock 调用正在进行中，或者服务提供者仍在处理一个回调函数。（参见 B.3.6.6 节）
WSAENOTSOCK	描述字不是一个套接口。

另请参阅：

WSACloseEvent() ,WSACreateEvent(),WSAEnumNetworkEvents(),WSAGetOverlappedResult(),WSAWaitForMultipleEvents().

7.9 WSAGetOverlappedResult()

简述：返回指定套接口上一个重叠操作的结果。

```
#include <winsock2.h>

BOOL WSAAPI WSAGetOverlappedResult( SOCKET s,
LPWSAOVERLAPPED lpOverlapped, LPDWORD
lpcbTransfer, BOOL fWait, LPDWORD lpdwFlags );
```

s：标识套接口。这就是调用重叠操作（WSARecv()、 WSARecvFrom()、WSASend()、WSASendTo() 或 WSAIoctl()）时指定的那个套接口。
lpOverlapped：指向调用重叠操作时指定的 WSAOVERLAPPED 结构。
lpcbTransfer：指向一个 32 位变量，该变量用于存放一个发送或接收操作实际传送的字节数，或 WSAIoctl()传送的字节数。
fWait：指定函数是否等待挂起的重叠操作结束。若为真 TRUE 则函数在操作完成后才返回。若为假 FALSE 且函数挂起，则函数返回 FALSE，WSAGetLastError()函数返回 WSA_IO_INCOMPLETE。
lpdwFlags：指向一个 32 位变量，该变量存放完成状态的附加标志位。如果重叠操作为 WSARecv()或 WSARecvFrom()，则本参数包含 lpFlags 参数所需的结果。

返回值：

如果函数成功,则返回值为真 TRUE。它意味着重叠操作已经完成,lpcbTransfer 所指向的值已经被刷新。应用程序可调用 WSAGetLastError()来获取重叠操作的错误信息。

如果函数失败，则返回值为假 FALSE。它意味着要么重叠操作未完成，要么由于一个或多个参数的错误导致无法决定完成状态。失败时，lpcbTransfer 指向的值不会被刷新。应用程序可用 WSAGetLastError()来获取失败的原因。

错误代码：

WSANOTINITIALISED	在调用本 API 之前应成功调用 WSAStartup()。
WSAENETDOWN	网络子系统失效。
WSAENOTSOCK	描述字不是一个套接口。
WSA_INVALID_HANDLE	WSAOVERLAPPED 结构的 hEvent 域未包含一个有效的事件对象句柄。
WSA_INVALID_PARAMETER	有不可接受的参数。
WSA_IO_INCOMPLETE	fWait 假 FALSE 且输入/输出操作尚未完成。

另请参阅： WSACreateEvent(), WSAWaitForMultipleEvents(),WSARecv(), WSARecvFrom(), WSASend(), WSASendTo(),WSAConnect(), WSAAccept(), WSAIoctl().

7.10 WSAGetQoSByName()

简述：根据一个模板初始化 QOS。

```
#include <winsock2.h>

BOOL WSAAPI WSAGetQOSByName( SOCKET s, LPWSABUF
lpQOSName, LPQOS lpQOS);
```

s：一个标识套接口的描述字。
lpQOSName：指定 QOS 模板的名字。
lpQOS：指向待填充 QOS 结构的指针。

返回值：
如果函数成功，返回真 TRUE。如果函数失败，返回假 FALSE。可调用 WSAGetLastError()来获取进一步的错误信息。

错误代码：

WSANOTINITIALISED	在调用本 API 之前应成功调用 WSAStartup()。
WSAENETDOWN	网络子系统失效。
WSAENOTSOCK	描述字不是一个套接口。
WSAEFAULT	lpQOS 参数不是用户地址空间的一个有效部分，或 lpQOS 的缓冲区太小。
WSA_INVAL	所指定的 QOS 模板名字非法。

另请参阅： WSAConnect(), WSAAccept(), getsockopt().

7.11 WSAHtonl()

简述：将一个以主机字节顺序表示的无符号长整形数转换为网络字节顺序。

```
#include <winsock2.h>

u_long WSAAPI WSAHtonl ( SOCKET  s, u_long
hostlong );
```

s：一个标识套接口的描述字。
hostlong：一个用主机字节顺序表示的 32 位数。

返回值：WSAHtonl()返回以网络字节顺序表示的值。

错误代码：
WSANOTINITIALISED　　在调用本 API 之前应成功调用 WSAStartup()。
WSAENETDOWN　　网络子系统失效。
WSAENOTSOCK　　描述字不是一个套接口。

另请参阅： htonl(), htons(), ntohs(), ntohl(), WSAHtons(),WSANtohl(), WSANtohs().

7.12 WSAHtons()

简述：将一个以主机字节顺序表示的无符号短整形数转换为网络字节顺序。

```
#include <winsock2.h>

u_short WSAAPI WSAHtons (SOCKET sr, u_short
hostshort );
```

s：一个标识套接口的描述字。
hostshort：一个以主机字节顺序表示的 16 位数。

返回值：WSAHtons()返回以网络字节顺序表示的值。

错误代码：
WSANOTINITIALISED　　在调用本 API 之前应成功调用 WSAStartup()。
WSAENETDOWN　　网络子系统失效。
WSAENOTSOCK　　描述字不是一个套接口。

另请参阅： htonl(), htons(), ntohs(), ntohl(), WSAHtonl(),WSANtohl(), WSANtohs().

7.13 WSAIoctl()

简述：控制一个套接口的模式。

```
#include <winsock2.h>

int WSAAPI WSAIoctl(SOCKET s, DWORD
dwIoControlCode, LPVOID lpvInBuffer, DWORD
cbInBuffer, LPVOID lpvOutBuffer, DWORD
cbOutBuffer, LPDWORD lpcbBytesReturned,
LPWSAOVERLAPPED lpOverlapped,
LPWSAOVERLAPPED_COMPLETION_ROUTINE
lpCompletionRoutine);
```

s：一个套接口的句柄。
dwIoControlCode：将进行的操作的控制代码。
lpvInBuffer：输入缓冲区的地址。
cbInBuffer：输入缓冲区的大小。
lpvOutBuffer：输出缓冲区的地址。
cbOutBuffer：输出缓冲区的大小。
lpcbBytesReturned：输出实际字节数的地址。
lpOverlapped：WSAOVERLAPPED 结构的地址。
lpCompletionRoutine：一个指向操作结束后调用的例程指针。

返回值：
调用成功后，WSAIoctl ()函数返回 0。否则的话，将返回 INVALID_SOCKET 错误，应用程序可通过 WSAGetLastError()来获取相应的错误代码。

错误代码：

WSANOTINITIALISED	在调用本 API 之前应成功调用 WSAStartup()。
WSAENETDOWN	网络子系统失效。
WSAEINVAL	cmd 不是一个合法的命令；或者一个输入参数非法；或者命令对于该种类型的套接口不适用。

WSAEINPROGRESS　　在一个回调函数运行时调用了该函数。
WSAENOTSOCK　　描述字不是一个套接口。
WSAEOPNOTSUPP　　指定的 ioctl 命令无法实现，例如在 SIO_SET_QOS 或 SIO_SET_GROUP_QOS 中指定的流描述无法实现。
WSA_IO_PENDING　　一个重叠操作被成功启动，过后将报告完成情况。
WSAEWOULDBLOCK　　套接口标志为非阻塞，且所需操作将产生阻塞。

另请参阅：　socket()，ioctlsocket()，WSASocket(),setsockopt()，getsockopt().

7.14 WSAJoinLeaf()

简述：将一个叶节点加入一个多点会晤，交换连接数据，根据提供的流描述确定所需的服务质量。

```
#include <winsock2.h>

SOCKET WSAAPI WSAJoinLeaf ( SOCKET s, const struct
sockaddr FAR * name, int namelen, LPWSABUF
lpCallerData, LPWSABUF lpCalleeData,
 LPQOS lpSQOS, LPQOS lpGQOS, int iFlags );
```

s：标识一个多点套接口的描述字。
name：将与套接口连接的远端名字。
namelen：名字长度。
lpCallerData：一个指针，指向多点会晤创建时传送给远端的用户数据。
lpCalleeData：一个指针，指向多点会晤创建时从远端传送回来的用户数据。
lpSQOS：一个指向套接口 s 的流描述的指针，每个方向一个。
lpGQOS：一个指向套接口组（如果存在）流描述的指针。
iFlags：标志位，用于指定套接口作为发送者。接收者或身兼二者。

返回值：若无错误发生，WSAJoinLeaf()返回新创建的多点套接口的描述字。否则的话，将返回 INVALID_SOCKET 错误，应用程序可通过 WSAGetLastError()来获取相应的错误代码。

错误代码：
WSANOTINITIALISED　　在调用本 API 之前应成功调用 WSAStartup()。

WSAENETDOWN　　网络子系统失效。
WSAEADDRINUSE　　指定的地址已经在使用中。
WSAEINTR　　通过 WSACancelBlockingCall()函数取消（阻塞）调用。
WSAEINPROGRESS　　一个阻塞的 WinSock 调用正在进行中，或者服务提供者仍在处理一个回调函数。（参见 B.3.6.6 节）
WSAEALREADY　　在指定的套接口上正在运行一个非阻塞的WSAJoinLeaf()调用。
WSAEADDRNOTAVAIL　本地主机无法获得指定的地址。
WSAEAFNOSUPPORT　　所指定地址族中的地址无法与本套接口一起使用。
WSAECONNREFUSED　　加入试图被强制拒绝。
WSAEFAULT　　name 或 namelen 参数不是用户地址空间的一个有效部分；namelen 参数太小；lpCalleeData、 lpSQOS 和 lpGQOS 的缓冲区太小；lpCallerData 缓冲区太大。
WSAEINVAL　　套接口已与一个地址捆绑。
WSAEINVAL　　套接口未与一个地址捆绑。
WSAEISCONN　　套接口已是多点会晤的一个成员。
WSAENETUNREACH　　当前无法从本主机联系网络。
WSAENOBUFS　　无可用缓冲区空间。套接口无法加入。
WSAENOTSOCK　　描述字不是一个套接口。
WSAEOPNOTSUPP　　lpSQOS 和 lpGQOS 中所指定的流描述无法满足。
WSAEPROTONOSUPPORT　服务提供者不支持 lpCallerData 参数。
WSAETIMEDOUT　　加入试图超时，未建立多点会晤。
WSAEWOULDBLOCK　　套接口被标志为非阻塞，但多点会晤加入操作无法立即完成。当用 select()选为读连接后，可使用 select()对套接口进行操作。

另请参阅： accept(), bind(), select(), WSAAccept(),WSAAsyncSelect(), WSAEventSelect(), WSASocket().

7.15 WSANtohl()

简述：将一个以网络字节顺序表示的无符号长整形数转换为主机字节顺序。

```
#include <winsock2.h>

u_long WSAAPI WSANtohl ( SOCKET s, u_long netlong
);
```

s：一个标识套接口的描述字。
netlong：一个以网络字节顺序表示的 32 位数。

返回值：WSANtohl()返回一个以主机字节顺序表示的值。

错误代码：

WSANOTINITIALISED	在调用本 API 之前应成功调用 WSAStartup()。
WSAENETDOWN	网络子系统失效。
WSAENOTSOCK	描述字不是一个套接口。

另请参阅： ntohl(), htonl(), htons(), ntohs(), WSAHtonl(),WSAHtons(), WSANtohs().

7.16 WSANtohs()

简述：将一个以网络字节顺序表示的无符号短整形数转换为主机字节顺序。

```
#include <winsock2.h>

u_short WSAAPI WSANtohs (SOCKET s, u_short
netshort );
```

s：一个标识套接口的描述字。
netshort：一个以网络字节顺序标识的 16 位数。

返回值：WSANtohs()返回以主机字节顺序表示的值。

错误代码：

WSANOTINITIALISED	在调用本 API 之前应成功调用 WSAStartup()。
WSAENETDOWN	网络子系统失效。
WSAENOTSOCK	描述字不是一个套接口。

另请参阅： htonl(), htons(), ntohs(), ntohl(), WSAHtonl(),WSAHtons(), WSANtohl().

7.17 WSARecv()

简述：从一个套接口接收数据。

```
#include <winsock2.h>

int WSAAPI WSARecv ( SOCKET s, LPWSABUF lpBuffers,
DWORD dwBufferCount, LPDWORD lpNumberOfBytesRecvd,
LPINT lpFlags, LPWSAOVERLAPPED lpOverlapped,
LPWSAOVERLAPPED_COMPLETION_ROUTINE
lpCompletionRoutine );
```

s：一个标识已连接套接口的描述字。
lpBuffers：一个指向 WSABUF 结构数组的指针。每一个 WSABUF 结构包含一个缓冲区的指针和缓冲区的长度。
dwBufferCount：lpBuffers 数组中 WSABUF 结构的数目。
lpNumberOfBytesRecvd：如果接收操作立即结束，一个指向本调用所接收的字节数的指针。
lpFlags：一个指向标志位的指针。
lpOverlapped：一个指向 WSAOVERLAPPED 结构的指针（对于非重叠套接口则忽略）。
lpCompletionRoutine：一个指向接收操作结束后调用的例程的指针（对于非重叠套接口则忽略）。

返回值：

若无错误发生且接收操作立即完成，则 WSARecv()函数返回所接收的字节数。如果连接结束，则返回 0。请注意在这种情况下完成指示（启动指定的完成例程或设置一个事件对象）将早已发生。否则的话，将返回 SOCKET_ERROR 错误，应用程序可通过 WSAGetLastError()来获取相应的错误代码。错误代码 WSA_IO_PENDING 表示重叠操作成功启动，过后将有完成指示。任何其他的错误表示重叠操作未能成功地启动，以后也不会有完成指示。

如果设置了 MSG_INTERRUPT 标志，则返回值的含义变化。零表示成功，具体含义同上。否则的话，返回值直接包含如下所示的错误代码。由于中断环境中无法调用 WSAGetLastError()，故是必需的。请注意仅适用于 Win16 环境，仅适用于 PROTOCOL_INFO 结构中设置了 XP1_INTERRUPT 位的协议。

错误代码：

WSANOTINITIALISED　　在调用本 API 之前应成功调用 WSAStartup()。
WSAENETDOWN　　网络子系统失效。
WSAENOTCONN　　套接口未连接。
WSAEINTR　　通过 WSACancelBlockingCall()函数取消（阻塞）调用。
WSAEINPROGRESS　　一个阻塞的 WinSock 调用正在进行中，或者服务提供者仍在处理一个回调函数。（参见 B.3.6.6 节）
WSAENETRESET　　由于远端的复位造成连接的中止。
WSAENOTSOCK　　描述字不是一个套接口。
WSAEOPNOTSUPP　　设置了 MSG_OOB，但是该套接口不是诸如 SOCK_STREAM 流类型的，与套接口相关的通讯域不支持带外数据，或者套接口是单向的，只支持发送操作。
WSAESHUTDOWN　　套接口已经关闭；一个套接口以 SD_RECEIVE 或 SD_BOTH 的 how 参数 shutdown()后，无法进行 WSARecv()调用。
WSAEWOULDBLOCK　　重叠套接口：太多重叠的输入/输出请求。非重叠套接口：套接口被标志为非阻塞，但是操作不能立即完成。
WSAEINVAL　　套接口未用 bind()捆绑，或者套接口未用重叠标志创建。
WSAECONNABORTED　由于超时或其他错误导致虚电路中止。
WSAECONNRESET　　虚电路被远端复位。
WSAEDISCON　　远端优雅的结束了连接。
WSA_IO_PENDING　　成功启动一个重叠操作，过后将有完成指示。

另请参阅： WSACloseEvent(),WSACreateEvent(),WSAGetOverlappedResult(), WSASocket(),WSAWaitForMultipleEvents()

7.18 WSARecvDisconnect()

简述：中止一个套接口上的接收操作；若套接口为面向连接的，则检索中止连接数据。

```
#include <winsock2.h>

int WSAAPI WSARecvDisconnect ( SOCKET s, LPWSABUF
lpInboundDisconnectData );
```

s：一个标识套接口的描述字。
lpInboundDisconnectData：一个指向前来的中止连接数据的中止。

返回值：若无错误发生，WSARecvDisconnect()返回 0。否则的话，返回 SOCKET_ERROR，应用程序可通过调用 WSAGetLastError()函数来获取相应的错误代码。

错误代码：

WSANOTINITIALISED	在调用本 API 之前应成功调用 WSAStartup()。
WSAENETDOWN	网络子系统失效。
WSAEFAULT	lpInboundDisconnectData 参数所提供的缓冲区太小。
WSAENOPROTOOPT	指定地址族不支持中止连接数据。
WSAEINPROGRESS	一个阻塞的 WinSock 调用正在进行中，或者服务提供者仍在处理一个回调函数。（参见 B.3.6.6 节）
WSAENOTCONN	套接口未连接（仅适用于面向连接的套接口）。
WSAENOTSOCK	描述字不是一个套接口。

另请参阅： connect(), socket().

7.19 WSARecvFrom()

简述：接收一个数据报并保存源地址。

```
#include <winsock2.h>1

int WSAAPI WSARecvFrom ( SOCKET s, LPWSABUF
lpBuffers, DWORD dwBufferCount, LPDWORD
lpNumberOfBytesRecvd, LPINT lpFlags, LPVOID
lpFrom, LPINT lpFromlen, LPWSAOVERLAPPED
lpOverlapped, LPWSAOVERLAPPED_COMPLETION_ROUTINE
lpCompletionRoutine );
```

s：一个标识套接口的描述字。
lpBuffers：一个指向 WSABUF 结构数组的指针。每个 WSABUF 结构包含缓冲区的指针和缓冲区的大小。
dwBufferCount：lpBuffers 数组中 WSABUF 结构的数目。
lpNumberOfBytesRecvd：如果接收操作立即完成，则为一个指向所接收数据字节数的指针。
lpFlags：一个指向标志位的指针。
lpFrom：（可选）指针，指向重叠操作完成后存放源地址的缓冲区。

lpFromlen：指向 from 缓冲区大小的指针，仅当指定了 lpFrom 才需要。
lpOverlapped：指向 WSAOVERLAPPED 结构的指针（对于非重叠套接口则忽略）。
lpCompletionRoutine：一个指向接收操作完成后调用的完成例程的指针。（对于非重叠套接口则忽略）。

返回值：

若无错误发生且接收操作立即完成，则 WSARecvFrom()函数返回所接收的字节数。如果连接结束，则返回 0。请注意在这种情况下完成指示（启动指定的完成例程或设置一个事件对象）将早已发生。否则的话，将返回 SOCKET_ERROR 错误，应用程序可通过 WSAGetLastError()来获取相应的错误代码。错误代码 WSA_IO_PENDING 表示重叠操作成功启动，过后将有完成指示。任何其他的错误表示重叠操作未能成功地启动，以后也不会有完成指示。

如果设置了 MSG_INTERRUPT 标志，则返回值的含义变化。零表示成功，具体含义同上。否则的话，返回值直接包含如下所示的错误代码。由于中断环境中无法调用 WSAGetLastError()，故是必需的。请注意仅适用于 Win16 环境，仅适用于 PROTOCOL_INFO 结构中设置了 XP1_INTERRUPT 位的协议。

错误代码：

WSANOTINITIALISED	在调用本 API 之前应成功调用 WSAStartup()。
WSAENETDOWN	网络子系统失效。
WSAEFAULT	lpFromlen 参数非法；lpFrom 缓冲区太小，无法容纳远端地址。
WSAEINTR	通过 WSACancelBlockingCall()函数取消（阻塞）调用。
WSAEINPROGRESS	一个阻塞的 WinSock 调用正在进行中，或者服务提供者仍在处理一个回调函数。（参见 B.3.6.6 节）
WSAEINVAL	套接口未用 bind()捆绑，或者套接口未用重叠标志创建。
WSAENETRESET	由于远端的复位造成连接的中止。
WSAENOTCONN	套接口未连接。（仅适用于面向连接的套接口）。
WSAENOTSOCK	描述字不是一个套接口。
WSAEOPNOTSUPP	设置了 MSG_OOB，但是该套接口不是诸如 SOCK_STREAM 流类型的，与套接口相关的通讯域不支持带外数据，或者套接口是单向的，只支持发送操作。
WSAESHUTDOWN	套接口已经关闭；一个套接口以 SD_RECEIVE 或 SD_BOTH 的 how 参数 shutdown()后，无法进行 WSARecvFrom()调用。
WSAEWOULDBLOCK	重叠套接口：太多重叠的输入/输出请求。非重叠套接口：套接口被标志为非阻塞，但是操作不能立即完成。
WSAEMSGSIZE	消息太大无法全部装入指定的缓冲区，故被修剪。

WSAECONNABORTED　由于超时或其他错误导致虚电路中止。
WSAECONNRESET　虚电路被远端复位。
WSAEDISCON　远端优雅地中止了连接。
WSA_IO_PENDING　成功启动一个重叠操作，过后将有完成指示。

另请参阅： WSACloseEvent(),WSACreateEvent(),WSAGetOverlappedResult(), WSASocket(),WSAWaitForMultipleEvents()

7.20 WSAResetEvent()

简述：将指定的事件对象状态清除为未置信号。

```
#include <winsock2.h>

BOOL WSAAPI WSAResetEvent( WSAEVENT hEvent );
```

hEvent：标识一个开放的事件对象句柄。

返回值：
如果函数成功，返回真 TRUE。如果函数失败，返回假 FALSE。可调用 WSAGetLastError()来获取进一步的错误信息。

错误代码：
WSA_INVALID_HANDLE　hEvent 不是一个合法的事件对象句柄。

另请参阅： WSACreateEvent(), WSASetEvent(), WSACloseEvent().

7.21 WSASend()

简述：在一个已连接的套接口上发送数据。

```
#include <winsock2.h>

int WSAAPI WSASend ( SOCKET s, LPWSABUF lpBuffers,
DWORD dwBufferCount, LPDWORD lpNumberOfBytesSent,
int iFlags, LPWSAOVERLAPPED lpOverlapped,
```

```
    LPWSAOVERLAPPED_COMPLETION_ROUTINE
    lpCompletionRoutine );
```

s：标识一个已连接套接口的描述字。
lpBuffers：一个指向 WSABUF 结构数组的指针。每个 WSABUF 结构包含缓冲区的指针和缓冲区的大小。
dwBufferCount：lpBuffers 数组中 WSABUF 结构的数目。
lpNumberOfBytesSent：如果发送操作立即完成，则为一个指向所发送数据字节数的指针。
iFlags：标志位。
lpOverlapped：指向 WSAOVERLAPPED 结构的指针（对于非重叠套接口则忽略）。
lpCompletionRoutine：一个指向发送操作完成后调用的完成例程的指针。（对于非重叠套接口则忽略）。

返回值：

若无错误发生且发送操作立即完成，则 WSASend()函数返回所发送的字节数。（注意该数目可能小于 len 参数所指定的值）。如果连接结束，则返回 0。请注意在这种情况下完成指示（启动指定的完成例程或设置一个事件对象）将早已发生。否则的话，将返回 SOCKET_ERROR 错误，应用程序可通过 WSAGetLastError()来获取相应的错误代码。错误代码 WSA_IO_PENDING 表示重叠操作成功启动，过后将有完成指示。任何其他的错误表示重叠操作未能成功地启动，以后也不会有完成指示。

如果设置了 MSG_INTERRUPT 标志，则返回值的含义变化。零表示成功，具体含义同上。否则的话，返回值直接包含如下所示的错误代码。由于中断环境中无法调用 WSAGetLastError()，故是必需的。请注意仅适用于 Win16 环境，仅适用于 PROTOCOL_INFO 结构中设置了 XP1_INTERRUPT 位的协议。

错误代码：

WSANOTINITIALISED	在调用本 API 之前应成功调用 WSAStartup()。
WSAENETDOWN	网络子系统失效。
WSAEACCES	请求地址为广播地址，但相应标志位没有设置。
WSAEINTR	通过 WSACancelBlockingCall()函数取消（阻塞）调用。
WSAEINPROGRESS	一个阻塞的 WinSock 调用正在进行中，或者服务提供者仍在处理一个回调函数。（参见 B.3.6.6 节）
WSAEFAULT	lpBuffer 参数并不是用户地址空间的一个有效部分。
WSAENETRESET	由于远端复位造成连接的中止。
WSAENOBUFS	WinSock 提供者报告一个缓冲区死锁。
WSAENOTCONN	套接口未连接。

WSAENOTSOCK　　描述字不是一个套接口。
WSAEOPNOTSUPP　　设置了 MSG_OOB，但是该套接口不是诸如 SOCK_STREAM 流类型的，与套接口相关的通讯域不支持带外数据，或者套接口是单向的，只支持接收操作。
WSAESHUTDOWN　　套接口已经关闭；一个套接口以 SD_SEND 或 SD _BOTH 的 how 参数 shutdown()后，无法进行 WSASend()调用。
WSAEWOULDBLOCK　　太多的重叠输入/输出操作。
WSAEMSGSIZE　　套接口是面向消息的，但是消息大于底层传送的最大值。
WSAEINVAL　　套接口未用 bind()捆绑，或者套接口未用重叠标志位创建。
WSAECONNABORTED　由于超时或其他错误导致虚电路中止。
WSAECONNRESET　　虚电路被远端复位。
WSA_IO_PENDING　　成功启动重叠操作，过后将有完成指示。

另请参阅： WSACloseEvent(),WSACreateEvent(),WSAGetOverlappedResult(), WSASocket(),WSAWaitForMultipleEvents()

7.22 WSASendDisconnect()

简述：启动套接口连接的中止操作。

```
#include <winsock2.h>

int WSAAPI WSASendDisconnect ( SOCKET s, LPWSABUF
lpOutboundDisconnectData );
```

s：一个标识套接口的描述字。
lpOutboundDisconnectData：指向发出的中止连接数据的指针。

返回值：若无错误发生，WSASendDisconnect()返回 0。否则的话，将返回 SOCKET_ERROR 错误，应用程序可通过 WSAGetLastError()来获取相应的错误代码。

错误代码：
WSANOTINITIALISED　　在调用本 API 之前应成功调用 WSAStartup()。
WSAENETDOWN　　网络子系统失效。

WSAENOPROTOOPT　　lpOutboundDisconnectData 参数非 NULL，复位提供者不支持中止连接数据。
WSAEINPROGRESS　　一个阻塞的 WinSock 调用正在进行中，或者服务提供者仍在处理一个回调函数。（参见 B.3.6.6 节）
WSAENOTCONN　　套接口未连接（仅适用于面向连接的套接口）。
WSAENOTSOCK　　描述字不是一个套接口。

另请参阅：　connect(), socket().

7.23 WSASendTo()

简述：向指定地址发送数据，可能的话使用重叠输入/输出操作。

```
#include <winsock2.h>

int WSAAPI WSASendTo ( SOCKET s, LPWSABUF
lpBuffers, DWORD dwBufferCount, LPDWORD
lpNumberOfBytesSent, int iFlags, LPVOID lpTo, int
iToLen, LPWSAOVERLAPPED lpOverlapped,
LPWSAOVERLAPPED_COMPLETION_ROUTINE
lpCompletionRoutine );
```

s：用于标识一个已连接的套接口，该套接口以 WSA_FLAG_OVERLAPPED 标志调用 WSASocket()创建。
lpBuffers：一个指向 WSABUF 结构数组的指针。每个 WSABUF 结构包含缓冲区的指针和缓冲区的大小。
dwBufferCount：lpBuffers 数组中 WSABUF 结构的数目。
lpNumberOfBytesSent：如果发送操作立即完成，则为一个指向所发送数据字节数的指针。
iFlags：标志位。
lpTo：（可选）指针，指向目标套接口的地址。
lpTolen：lpTo 中地址的大小。
lpOverlapped：指向 WSAOVERLAPPED 结构的指针（对于非重叠套接口则忽略）。
lpCompletionRoutine：一个指向发送操作完成后调用的完成例程的指针。（对于非重叠套接口则忽略）。

返回值：

若无错误发生且发送操作立即完成，则 WSASendTo()函数返回所发送的字节数（请注意它可能小于 len 所指定的值）。请注意在这种情况下完成指示（启动指定的完成例程或设置一个事件对象）将早已发生。否则的话，将返回 SOCKET_ERROR 错误，应用程序可通过 WSAGetLastError()来获取相应的错误代码。错误代码 WSA_IO_PENDING 表示重叠操作成功启动，过后将有完成指示。任何其他的错误表示重叠操作未能成功地启动，以后也不会有完成指示。

如果设置了 MSG_INTERRUPT 标志，则返回值的含义变化。零表示成功，具体含义同上。否则的话，返回值直接包含如下所示的错误代码。由于中断环境中无法调用 WSAGetLastError()，故是必需的。请注意仅适用于 Win16 环境，仅适用于 PROTOCOL_INFO 结构中设置了 XP1_INTERRUPT 位的协议。

错误代码：

WSANOTINITIALISED　在调用本 API 之前应成功调用 WSAStartup()。

WSAENETDOWN　网络子系统失效。

WSAEACCES　请求的地址为广播地址，但未设置相应的标志位。

WSAEINTR　通过 WSACancelBlockingCall()函数取消（阻塞）调用。

WSAEINPROGRESS　一个阻塞的 WinSock 调用正在进行中，或者服务提供者仍在处理一个回调函数。（参见 B.3.6.6 节）

WSAEFAULT　lpBuffer 或 lpTo 参数不是用户地址空间的一部分；或者 lpTo 参数太小（小于 sockaddr 结构的大小）。

WSAENETRESET　远端主机复位造成连接的中止。

WSAENOBUFS　WinSock 提供者报告了一个缓冲区死锁。

WSAENOTCONN　套接口未连接（仅适用于面向连接的套接口）。

WSAENOTSOCK　描述字不是一个套接口。

WSAEOPNOTSUPP　设置了 MSG_OOB，但是该套接口不是诸如 SOCK_STREAM 流类型的，与套接口相关的通讯域不支持带外数据，或者套接口是单向的，只支持接收操作。

WSAESHUTDOWN　套接口已经关闭；一个套接口以 SD_SEND 或 SD _BOTH 的 how 参数 shutdown()后，无法进行 WSASendTo()调用。

WSAEWOULDBLOCK　太多重叠的输入/输出请求。

WSAEMSGSIZE　套接口是面向消息的，且消息大于底层传送所支持的最大长度。

WSAEINVAL　套接口未用 bind()捆绑，或者套接口未用重叠标志位创建。

WSAECONNABORTED　由于超时或其他错误导致虚电路中止。

WSAECONNRESET　虚电路被远端复位。

WSAEADDRNOTAVAIL　本地主机无法获取所指定的地址。

WSAEAFNOSUPPORT　　指定地址族中的地址无法与本套接口一起使用。
WSAEDESTADDRREQ　需要目的地地址。
WSAENETUNREACH　　当前无法从本主机联系网络。
WSA_IO_PENDING　　成功启动一个重叠操作，过后将有完成指示。

另请参阅： WSACloseEvent(),WSACreateEvent(),WSAGetOverlappedResult(), WSASocket(),WSAWaitForMultipleEvents()

7.24 WSASetEvent()

简述：将指定的事件对象状态设置为有信号。

```
#include <winsock2.h>

BOOL WSAAPI WSASetEvent( WSAEVENT hEvent );
```

hEvent：标识一个开放的事件对象句柄。

返回值：
　如果函数成功，返回真 TRUE。
　如果函数失败，返回假 FALSE。可通过调用 WSAGetLastError()来获取进一步的错误信息。

错误代码：
WSANOTINITIALISED　在调用本 API 之前应成功调用 WSAStartup()。
WSAENETDOWN　　网络子系统失效。
WSA_INVALID_HANDLE　hEvent 不是一个合法的事件对象句柄。

另请参阅： WSACreateEvent(), WSAResetEvent(),WSACloseEvent().

7.25 WSASocket()

简述：创建一个与指定传送服务提供者捆绑的套接口，可选地创建和/或加入一个套接口组。

```
#include <winsock2.h>

SOCKET WSAAPI WSASocket ( int af, int type, int
protocol, LPPROTOCOL_INFO lpProtocolInfo, Group g,
int iFlags);
```

af：地址族描述。目前仅支持 PF_INET 格式，亦即 ARPA Internet 地址格式。
type：新套接口的类型描述。
protocol：套接口使用的特定协议，如果调用者不愿指定协议则定为 0。
lpProtocolInfo：一个指向 PROTOCOL_INFO 结构的指针，该结构定义所创建套接口的特性。如果本参数非零，则前三个参数（af，type，protocol）被忽略。
g：套接口组的描述字。
iFlags：套接口属性描述。

返回值：
　若无错误发生，WSASocket()返回新套接口的描述字。否则的话，返回 INVALID_SOCKET，应用程序可定调用 WSAGetLastError()来获取相应的错误代码。

错误代码：
WSANOTINITIALISED　　在调用本 API 之前应成功调用 WSAStartup()。
WSAENETDOWN　　网络子系统失效。
WSAEAFNOSUPPORT　　不支持指定的地址族。
WSAEINPROGRESS　　一个阻塞的 WinSock 调用正在进行中，或者服务提供者仍在处理一个回调函数。（参见 B.3.6.6 节）
WSAEMFILE　　无可用的套接口描述字。
WSAENOBUFS　　无可用的缓冲区空间。套接口无法创建。
WSAEPROTONOSUPPORT　　不支持指定的协议。
WSAEPROTOTYPE　　指定的协议对于本套接口类型错误。
WSAESOCKTNOSUPPORT　　本地址族不支持指定的套接口类型。
WSAEINVAL　　g 参数非法。

另请参阅：　accept(), bind(), connect(), getsockname(),getsockopt(), setsockopt(), listen(), recv(),recvfrom(), select(), send(), sendto(),shutdown(), ioctlsocket().

7.26 WSAWaitForMultipleEvents()

简述：只要指定事件对象中的一个或全部处于有信号状态，或者超时间隔到，则返回。

```
#include <winsock2.h>

DWORD WSAAPI WSAWaitForMultipleEvents( DWORD
cEvents,
const WSAEVENT FAR * lphEvents, BOOL fWaitAll,
DWORD dwTimeout,
BOOL fAlertable );
```

cEvents：指出 lphEvents 所指数组中事件对象句柄的数目。事件对象句柄的最大值为 WSA_MAXIMUM_WAIT_EVENTS。
lphEvents：指向一个事件对象句柄数组的指针。
fWaitAll：指定等待类型。若为真 TRUE，则当 lphEvents 数组中的所有事件对象同时有信号时，函数返回。若为假 FALSE，则当任意一个事件对象有信号时函数即返回。在后一种情况下，返回值指出是哪一个事件对象造成函数返回。
dwTimeout：指定超时时间间隔（以毫秒计）。当超时间隔到，函数即返回，不论 fWaitAll 参数所指定的条件是否满足。如果 dwTimeout 为零，则函数测试指定的时间对象的状态，并立即返回。如果 dwTimeout 是 WSA_INFINITE，则函数的超时间隔永远不会到。
fAlertable：指定当系统将一个输入/输出完成例程放入队列以供执行时，函数是否返回。若为真 TRUE，则函数返回且执行完成例程。若为假 FALSE，函数不返回，不执行完成例程。请注意在 Win16 中忽略该参数。

返回值：
　如果函数成功，返回值指出造成函数返回的事件对象。
　如果函数失败，返回值为 WSA_WAIT_FAILED。可调用 WSAGetLastError()来获取进一步的错误信息。

错误代码：
WSANOTINITIALISED　　在调用本 API 之前应成功调用 WSAStartup()。
WSAENETDOWN　　网络子系统失效。
WSA_NOT_ENOUGH_MEMORY　　无足够内存完成该操作。
WSA_INVALID_HANDLE　　lphEvents 数组中的一个或多个值不是合法的事件对象句柄。

WSA_INVALID_PARAMETER　　cEvents 参数未包含合法的句柄数目。

另请参阅：　WSACreateEvent(), WSACloseEvent().

附录 A 错误代码

以下列出了 WSAGetLastError()函数有可能返回的错误代码和对应的解释。错误代码的数值对于所有的 Windows Sockets 兼容实现都是一致的。

Windows Sockets 代码	Berkeley 对应代码	错误数值	解释
WSAEINTR	EINTR	10004	与标准 C 一致
WSAEBADF	EBADF	10009	与标准 C 一致
WSAEACCES	EACCES	10013	与标准 C 一致
WSAEFAULT	EFAULT	10014	与标准 C 一致
WSAEINVAL	EINVAL	10022	与标准 C 一致
WSAEMFILE	EMFILE	10024	与标准 C 一致
WSAEWOULDBLOCK	EWOULDBLOCK	10035	与 BSD 一致
WSAEINPROGRESS	EINPROGRESS	10036	当应用程序调用 Windows Sockets API 函数时，如果一个阻塞函数正在运行，将返回该错误。
WSAEALREADY	EALREADY	10037	与 BSD 一致
WSAENOTSOCK	ENOTSOCK	10038	与 BSD 一致
WSAEDESTADDRREQ	EDESTADDRREQ	10039	与 BSD 一致
WSAEMSGSIZE	EMSGSIZE	10040	与 BSD 一致

WSAEPROTOTYPE	EPROTOTYPE	10041	与 BSD 一致
WSAENOPROTOOPT	ENOPROTOOPT	10042	与 BSD 一致
WSAEPROTONOSUPPORT	EPROTONOSUPPORT	10043	与 BSD 一致
WSAESOCKTNOSUPPORT	ESOCKTNOSUPPORT	10044	与 BSD 一致
WSAEOPNOTSUPPORT	EOPNOTSUPPORT	10045	与 BSD 一致
WSAEPFNOSUPPORT	EPFNOSUPPORT	10046	与 BSD 一致
WSAEAFNOSUPPORT	EAFNOSUPPORT	10047	与 BSD 一致
WSAEADDRINUSE	EADDRINUSE	10048	与 BSD 一致
WSAEADDRNOTAVAIL	EADDRNOTAVAIL	10049	与 BSD 一致
WSAENETDOWN	ENETDOWN	10050	与 BSD 一致，在 Windows Sockets 检测到任何底层的失败时，都有可能返回该错误。
WSAENETUNREACH	ENETUNREACH	10051	与 BSD 一致
WSAENETRESET	ENETRESET	10052	与 BSD 一致

WSAECONNABORTED	ECONNABORTED	10053	与 BSD 一致
WSAECONNRESET	ECONNRESET	10054	与 BSD 一致
WSAENOBUFS	ENOBUFS	10055	与 BSD 一致
WSAEISCONN	EISCONN	10056	与 BSD 一致
WSAENOTCONN	ENOTCONN	10057	与 BSD 一致
WSAESHUTDOWN	ESHUTDOWN	10058	与 BSD 一致
WSAETOOMANYREFS	ETOOMANYREFS	10059	与 BSD 一致
WSAETIMEDOUT	ETIMEDOUT	10060	与 BSD 一致
WSAECONNREFUSED	ECONNREFUSED	10061	与 BSD 一致
WSAELOOP	ELOOP	10062	与 BSD 一致
WSAENAMETOOLONG	ENAMETOOLONG	10063	与 BSD 一致
WSAEHOSTDOWN	EHOSTDOWN	10064	与 BSD 一致
WSAEHOSTUNREACH	EHOSTUNREACH	10065	与 BSD 一致
WSASYSNOTREADY		10091	由 WSAStartup() 函数返回，表明底层的网络子系统无法使用。
WSAVERNOTSUPPORTED		10092	由 WSAStartup() 函数返回，表明 Windows Sockets

			DLL 不支持这一应用。
WSANOTINITIALISED		10093	由除了 WSAStartup()的其他函数返回，表明没有对 WSAStartup()函数的成功调用。
WSAHOST_NOT_FOUND	HOST_NOT_FOUND	11001	与 BSD 一致
WSATRY_AGAIN	TRY_AGAIN	11001	与 BSD 一致
WSANO_RECOVERY	NO_RECOVERY	11003	与 BSD 一致
WSANO_DATA	NO_DATA	11004	与 BSD 一致

附录 B Windows Sockets 头文件

附录 B.1 Windows Sockets 1.1 头文件

```
/* WINSOCK.H—definitions to be used with the WINSOCK.DLL
 *
 * This header file corresponds to version 1.1 of the Windows Sockets
specification.
 *
 * This file includes parts which are Copyright (c) 1982-1986 Regents
 * of the University of California.  All rights reserved.  The
 * Berkeley Software License Agreement specifies the terms and
 * conditions for redistribution.
 */

#ifndef _WINSOCKAPI_
#define _WINSOCKAPI_

/*
 * Pull in WINDOWS.H if necessary
 */
#ifndef _INC_WINDOWS
#include <windows.h>
#endif /* _INC_WINDOWS */

/*
 * Basic system type definitions, taken from the BSD file sys/types.h.
 */
typedef unsigned char   u_char;
typedef unsigned short  u_short;
typedef unsigned int    u_int;
typedef unsigned long   u_long;

/*
```

```
 * The new type to be used in all
 * instances which refer to sockets.
 */
typedef u_int           SOCKET;

/*
 * Select uses arrays of SOCKETs.  These macros manipulate such
 * arrays.  FD_SETSIZE may be defined by the user before including
 * this file, but the default here should be >= 64.
 *
 * CAVEAT IMPLEMENTOR and USER: THESE MACROS AND TYPES MUST BE
 * INCLUDED IN WINSOCK.H EXACTLY AS SHOWN HERE.
 */
#ifndef FD_SETSIZE
#define FD_SETSIZE      64
#endif /* FD_SETSIZE */

typedef struct fd_set {
        u_short fd_count;               /* how many are SET? */
        SOCKET  fd_array[FD_SETSIZE];   /* an array of SOCKETs */
} fd_set;

extern int PASCAL FAR __WSAFDIsSet(SOCKET, fd_set FAR *);

#define FD_CLR(fd, set) do { \
    u_int __i; \
    for (__i = 0; __i < ((fd_set FAR *)(set))->fd_count ; __i++) { \
        if (((fd_set FAR *)(set))->fd_array[__i] == fd) { \
            while (__i < ((fd_set FAR *)(set))->fd_count-1) { \
                ((fd_set FAR *)(set))->fd_array[__i] = \
                    ((fd_set FAR *)(set))->fd_array[__i+1]; \
                __i++; \
            } \
            ((fd_set FAR *)(set))->fd_count—; \
            break; \
        } \
    } \
} while(0)
```

```
#define FD_SET(fd, set) do { \
    if (((fd_set FAR *)(set))->fd_count < FD_SETSIZE) \
        ((fd_set FAR *)(set))->fd_array[((fd_set FAR *)(set))-
>fd_count++]=fd;\
} while(0)

#define FD_ZERO(set) (((fd_set FAR *)(set))->fd_count=0)

#define FD_ISSET(fd, set) __WSAFDIsSet((SOCKET)fd, (fd_set FAR *)set)

/*
 * Structure used in select() call, taken from the BSD file sys/time.h.
 */
struct timeval {
        long    tv_sec;         /* seconds */
        long    tv_usec;        /* and microseconds */
};

/*
 * Operations on timevals.
 *
 * NB: timercmp does not work for >= or <=.
 */
#define timerisset(tvp)         ((tvp)->tv_sec || (tvp)->tv_usec)
#define timercmp(tvp, uvp, cmp) \
        ((tvp)->tv_sec cmp (uvp)->tv_sec || \
         (tvp)->tv_sec == (uvp)->tv_sec && (tvp)->tv_usec cmp (uvp)-
>tv_usec)
#define timerclear(tvp)         (tvp)->tv_sec = (tvp)->tv_usec = 0

/*
 * Commands for ioctlsocket(),  taken from the BSD file fcntl.h.
 *
 *
 * Ioctl's have the command encoded in the lower word,
 * and the size of any in or out parameters in the upper
 * word.  The high 2 bits of the upper word are used
```

```
 * to encode the in/out status of the parameter; for now
 * we restrict parameters to at most 128 bytes.
 */
#define IOCPARM_MASK    0x7f            /* parameters must be < 128
bytes */
#define IOC_VOID        0x20000000      /* no parameters */
#define IOC_OUT         0x40000000      /* copy out parameters */
#define IOC_IN          0x80000000      /* copy in parameters */
#define IOC_INOUT       (IOC_IN|IOC_OUT)
                                        /* 0x20000000 distinguishes
new &
                                           old ioctl's */
#define _IO(x,y)        (IOC_VOID|(x<<8)|y)

#define _IOR(x,y,t)
(IOC_OUT|(((long)sizeof(t)&IOCPARM_MASK)<<16)|(x<<8)|y)

#define _IOW(x,y,t)
(IOC_IN|(((long)sizeof(t)&IOCPARM_MASK)<<16)|(x<<8)|y)

#define FIONREAD    _IOR('f', 127, u_long) /* get # bytes to read */
#define FIONBIO    _IOW('f', 126, u_long) /* set/clear non-blocking i/o
*/
#define FIOASYNC    _IOW('f', 125, u_long) /* set/clear async i/o */

/* Socket I/O Controls */
#define SIOCSHIWAT  _IOW('s',  0, u_long)  /* set high watermark */
#define SIOCGHIWAT  _IOR('s',  1, u_long)  /* get high watermark */
#define SIOCSLOWAT  _IOW('s',  2, u_long)  /* set low watermark */
#define SIOCGLOWAT  _IOR('s',  3, u_long)  /* get low watermark */
#define SIOCATMARK  _IOR('s',  7, u_long)  /* at oob mark? */

/*
 * Structures returned by network data base library, taken from the
 * BSD file netdb.h.  All addresses are supplied in host order, and
 * returned in network order (suitable for use in system calls).
 */
```

```
struct  hostent {
        char    FAR * h_name;           /* official name of host */
        char    FAR * FAR * h_aliases;  /* alias list */
        short   h_addrtype;             /* host address type */
        short   h_length;               /* length of address */
        char    FAR * FAR * h_addr_list; /* list of addresses */
#define h_addr  h_addr_list[0]          /* address, for backward compat
*/
};

/*
 * It is assumed here that a network number
 * fits in 32 bits.
 */
struct  netent {
        char    FAR * n_name;           /* official name of net */
        char    FAR * FAR * n_aliases;  /* alias list */
        short   n_addrtype;             /* net address type */
        u_long  n_net;                  /* network # */
};

struct  servent {
        char    FAR * s_name;           /* official service name */
        char    FAR * FAR * s_aliases;  /* alias list */
        short   s_port;                 /* port # */
        char    FAR * s_proto;          /* protocol to use */
};

struct  protoent {
        char    FAR * p_name;           /* official protocol name */
        char    FAR * FAR * p_aliases;  /* alias list */
        short   p_proto;                /* protocol # */
};

/*
 * Constants and structures defined by the internet system,
 * Per RFC 790, September 1981, taken from the BSD file netinet/in.h.
 */
```

```
/*
 * Protocols
 */
#define IPPROTO_IP              0               /* dummy for IP */
#define IPPROTO_ICMP            1               /* control message
protocol */
#define IPPROTO_GGP             2               /* gateway^2
(deprecated) */
#define IPPROTO_TCP             6               /* tcp */
#define IPPROTO_PUP             12              /* pup */
#define IPPROTO_UDP             17              /* user datagram
protocol */
#define IPPROTO_IDP             22              /* xns idp */
#define IPPROTO_ND              77              /* UNOFFICIAL net disk
proto */

#define IPPROTO_RAW             255             /* raw IP packet */
#define IPPROTO_MAX             256

/*
 * Port/socket numbers: network standard functions
 */
#define IPPORT_ECHO             7
#define IPPORT_DISCARD          9
#define IPPORT_SYSTAT           11
#define IPPORT_DAYTIME          13
#define IPPORT_NETSTAT          15
#define IPPORT_FTP              21
#define IPPORT_TELNET           23
#define IPPORT_SMTP             25
#define IPPORT_TIMESERVER       37
#define IPPORT_NAMESERVER       42
#define IPPORT_WHOIS            43
#define IPPORT_MTP              57

/*
 * Port/socket numbers: host specific functions
```

```
 */
#define IPPORT_TFTP             69
#define IPPORT_RJE              77
#define IPPORT_FINGER           79
#define IPPORT_TTYLINK          87
#define IPPORT_SUPDUP           95

/*
 * UNIX TCP sockets
 */
#define IPPORT_EXECSERVER       512
#define IPPORT_LOGINSERVER      513
#define IPPORT_CMDSERVER        514
#define IPPORT_EFSSERVER        520

/*
 * UNIX UDP sockets
 */
#define IPPORT_BIFFUDP          512
#define IPPORT_WHOSERVER        513
#define IPPORT_ROUTESERVER      520
                                        /* 520+1 also used */

/*
 * Ports < IPPORT_RESERVED are reserved for
 * privileged processes (e.g. root).
 */
#define IPPORT_RESERVED         1024

/*
 * Link numbers
 */
#define IMPLINK_IP              155
#define IMPLINK_LOWEXPER        156
#define IMPLINK_HIGHEXPER       158

/*
 * Internet address (old style... should be updated)
```

```
 */
struct in_addr {
        union {
                struct { u_char s_b1,s_b2,s_b3,s_b4; } S_un_b;
                struct { u_short s_w1,s_w2; } S_un_w;
                u_long S_addr;
        } S_un;
#define s_addr  S_un.S_addr
                                /* can be used for most tcp & ip code */
#define s_host  S_un.S_un_b.s_b2
                                  /* host on imp */
#define s_net   S_un.S_un_b.s_b1
                                  /* network */
#define s_imp   S_un.S_un_w.s_w2
                                  /* imp */
#define s_impno S_un.S_un_b.s_b4
                                  /* imp # */
#define s_lh    S_un.S_un_b.s_b3
                                  /* logical host */
};

/*
 * Definitions of bits in internet address integers.
 * On subnets, the decomposition of addresses to host and net parts
 * is done according to subnet mask, not the masks here.
 */
#define IN_CLASSA(i)            (((long)(i) & 0x80000000) == 0)
#define IN_CLASSA_NET           0xff000000
#define IN_CLASSA_NSHIFT        24
#define IN_CLASSA_HOST          0x00ffffff
#define IN_CLASSA_MAX           128

#define IN_CLASSB(i)            (((long)(i) & 0xc0000000) ==
0x80000000)
#define IN_CLASSB_NET           0xffff0000
#define IN_CLASSB_NSHIFT        16
#define IN_CLASSB_HOST          0x0000ffff
#define IN_CLASSB_MAX           65536
```

```
#define IN_CLASSC(i)            (((long)(i) & 0xc0000000) ==
0xc0000000)
#define IN_CLASSC_NET           0xffffff00
#define IN_CLASSC_NSHIFT        8
#define IN_CLASSC_HOST          0x000000ff

#define INADDR_ANY              (u_long)0x00000000
#define INADDR_LOOPBACK         0x7f000001
#define INADDR_BROADCAST        (u_long)0xffffffff
#define INADDR_NONE             0xffffffff

/*
 * Socket address, internet style.
 */
struct sockaddr_in {
        short   sin_family;
        u_short sin_port;
        struct  in_addr sin_addr;
        char    sin_zero[8];
};

#define WSADESCRIPTION_LEN      256
#define WSASYS_STATUS_LEN       128

typedef struct WSAData {
        WORD                    wVersion;
        WORD                    wHighVersion;
        char                    szDescription[WSADESCRIPTION_LEN+1];
        char                    szSystemStatus[WSASYS_STATUS_LEN+1];
        unsigned short          iMaxSockets;
        unsigned short          iMaxUdpDg;
        char FAR *              lpVendorInfo;
} WSADATA;

typedef WSADATA FAR *LPWSADATA;

/*
```

```
 * Options for use with [gs]etsockopt at the IP level.
 */
#define IP_OPTIONS      1               /* set/get IP per-packet
options */

/*
 * Definitions related to sockets: types, address families, options,
 * taken from the BSD file sys/socket.h.
 */

/*
 * This is used instead of -1, since the
 * SOCKET type is unsigned.
 */
#define INVALID_SOCKET  (SOCKET)(~0)
#define SOCKET_ERROR            (-1)

/*
 * Types
 */
#define SOCK_STREAM     1               /* stream socket */
#define SOCK_DGRAM      2               /* datagram socket */
#define SOCK_RAW        3               /* raw-protocol interface */
#define SOCK_RDM        4               /* reliably-delivered message
*/
#define SOCK_SEQPACKET  5               /* sequenced packet stream */

/*
 * Option flags per-socket.
 */
#define SO_DEBUG        0x0001          /* turn on debugging info
recording */
#define SO_ACCEPTCONN   0x0002          /* socket has had listen() */
#define SO_REUSEADDR    0x0004         /* allow local address reuse */
#define SO_KEEPALIVE    0x0008          /* keep connections alive */
#define SO_DONTROUTE    0x0010         /* just use interface addresses
*/
#define SO_BROADCAST    0x0020          /* permit sending of broadcast
```

```
msgs */
#define SO_USELOOPBACK  0x0040          /* bypass hardware when
possible */
#define SO_LINGER       0x0080          /* linger on close if data
present */
#define SO_OOBINLINE    0x0100          /* leave received OOB data in
line */

#define SO_DONTLINGER   (u_int)(~SO_LINGER)

/*
 * Additional options.
 */
#define SO_SNDBUF       0x1001          /* send buffer size */
#define SO_RCVBUF       0x1002          /* receive buffer size */
#define SO_SNDLOWAT     0x1003          /* send low-water mark */
#define SO_RCVLOWAT     0x1004          /* receive low-water mark */
#define SO_SNDTIMEO     0x1005          /* send timeout */
#define SO_RCVTIMEO     0x1006          /* receive timeout */
#define SO_ERROR       0x1007         /* get error status and clear */
#define SO_TYPE         0x1008          /* get socket type */

/*
 * TCP options.
 */
#define TCP_NODELAY     0x0001

/*
 * Address families.
 */
#define AF_UNSPEC       0               /* unspecified */
#define AF_UNIX        1              /* local to host (pipes, portals)
*/
#define AF_INET         2               /* internetwork: UDP, TCP, etc.
*/
#define AF_IMPLINK      3               /* arpanet imp addresses */
#define AF_PUP          4               /* pup protocols: e.g. BSP */
#define AF_CHAOS        5               /* mit CHAOS protocols */
```

```
#define AF_NS           6               /* XEROX NS protocols */
#define AF_ISO          7               /* ISO protocols */
#define AF_OSI          AF_ISO          /* OSI is ISO */
#define AF_ECMA         8               /* european computer
manufacturers */
#define AF_DATAKIT      9               /* datakit protocols */
#define AF_CCITT        10             /* CCITT protocols, X.25 etc */
#define AF_SNA          11              /* IBM SNA */
#define AF_DECnet       12              /* DECnet */
#define AF_DLI          13             /* Direct data link interface */
#define AF_LAT          14              /* LAT */
#define AF_HYLINK       15              /* NSC Hyperchannel */
#define AF_APPLETALK    16              /* AppleTalk */
#define AF_NETBIOS      17              /* NetBios-style addresses */

#define AF_MAX          18

/*
 * Structure used by kernel to store most
 * addresses.
 */
struct sockaddr {
        u_short sa_family;              /* address family */
        char    sa_data[14];            /* up to 14 bytes of direct
address */
};

/*
 * Structure used by kernel to pass protocol
 * information in raw sockets.
 */
struct sockproto {
        u_short sp_family;              /* address family */
        u_short sp_protocol;            /* protocol */
};

/*
 * Protocol families, same as address families for now.
```

```
 */
#define PF_UNSPEC       AF_UNSPEC
#define PF_UNIX         AF_UNIX
#define PF_INET         AF_INET
#define PF_IMPLINK      AF_IMPLINK
#define PF_PUP          AF_PUP
#define PF_CHAOS        AF_CHAOS
#define PF_NS           AF_NS
#define PF_ISO          AF_ISO
#define PF_OSI          AF_OSI
#define PF_ECMA         AF_ECMA
#define PF_DATAKIT      AF_DATAKIT
#define PF_CCITT        AF_CCITT
#define PF_SNA          AF_SNA
#define PF_DECnet       AF_DECnet
#define PF_DLI          AF_DLI
#define PF_LAT          AF_LAT
#define PF_HYLINK       AF_HYLINK
#define PF_APPLETALK    AF_APPLETALK

#define PF_MAX          AF_MAX

/*
 * Structure used for manipulating linger option.
 */
struct  linger {
        u_short l_onoff;                /* option on/off */
        u_short l_linger;               /* linger time */
};

/*
 * Level number for (get/set)sockopt() to apply to socket itself.
 */
#define SOL_SOCKET      0xffff          /* options for socket level */

/*
 * Maximum queue length specifiable by listen.
 */
```

```
#define SOMAXCONN       5

#define MSG_OOB         0x1             /* process out-of-band data */
#define MSG_PEEK        0x2             /* peek at incoming message */
#define MSG_DONTROUTE   0x4             /* send without using routing
tables */

#define MSG_MAXIOVLEN   16

/*
 * Define constant based on rfc883, used by gethostbyxxxx() calls.
 */
#define MAXGETHOSTSTRUCT        1024

/*
 * Define flags to be used with the WSAAsyncSelect() call.
 */
#define FD_READ         0x01
#define FD_WRITE        0x02
#define FD_OOB          0x04
#define FD_ACCEPT       0x08
#define FD_CONNECT      0x10
#define FD_CLOSE        0x20

/*
 * All Windows Sockets error constants are biased by WSABASEERR from
 * the "normal"
 */
#define WSABASEERR              10000
/*
 * Windows Sockets definitions of regular Microsoft C error constants
 */
#define WSAEINTR                (WSABASEERR+4)
#define WSAEBADF                (WSABASEERR+9)
#define WSAEACCES               (WSABASEERR+13)
#define WSAEFAULT               (WSABASEERR+14)
#define WSAEINVAL               (WSABASEERR+22)
#define WSAEMFILE               (WSABASEERR+24)
```

```
/*
 * Windows Sockets definitions of regular Berkeley error constants
 */
#define WSAEWOULDBLOCK          (WSABASEERR+35)
#define WSAEINPROGRESS          (WSABASEERR+36)
#define WSAEALREADY             (WSABASEERR+37)
#define WSAENOTSOCK             (WSABASEERR+38)
#define WSAEDESTADDRREQ         (WSABASEERR+39)
#define WSAEMSGSIZE             (WSABASEERR+40)
#define WSAEPROTOTYPE           (WSABASEERR+41)
#define WSAENOPROTOOPT          (WSABASEERR+42)
#define WSAEPROTONOSUPPORT      (WSABASEERR+43)
#define WSAESOCKTNOSUPPORT      (WSABASEERR+44)
#define WSAEOPNOTSUPP           (WSABASEERR+45)
#define WSAEPFNOSUPPORT         (WSABASEERR+46)
#define WSAEAFNOSUPPORT         (WSABASEERR+47)
#define WSAEADDRINUSE           (WSABASEERR+48)
#define WSAEADDRNOTAVAIL        (WSABASEERR+49)
#define WSAENETDOWN             (WSABASEERR+50)
#define WSAENETUNREACH          (WSABASEERR+51)
#define WSAENETRESET            (WSABASEERR+52)
#define WSAECONNABORTED         (WSABASEERR+53)
#define WSAECONNRESET           (WSABASEERR+54)
#define WSAENOBUFS              (WSABASEERR+55)
#define WSAEISCONN              (WSABASEERR+56)
#define WSAENOTCONN             (WSABASEERR+57)
#define WSAESHUTDOWN            (WSABASEERR+58)
#define WSAETOOMANYREFS         (WSABASEERR+59)
#define WSAETIMEDOUT            (WSABASEERR+60)
#define WSAECONNREFUSED         (WSABASEERR+61)
#define WSAELOOP                (WSABASEERR+62)
#define WSAENAMETOOLONG         (WSABASEERR+63)
#define WSAEHOSTDOWN            (WSABASEERR+64)
#define WSAEHOSTUNREACH         (WSABASEERR+65)
#define WSAENOTEMPTY            (WSABASEERR+66)
#define WSAEPROCLIM             (WSABASEERR+67)
#define WSAEUSERS               (WSABASEERR+68)
```

```
#define WSAEDQUOT               (WSABASEERR+69)
#define WSAESTALE               (WSABASEERR+70)
#define WSAEREMOTE              (WSABASEERR+71)

/*
 * Extended Windows Sockets error constant definitions
 */
#define WSASYSNOTREADY          (WSABASEERR+91)
#define WSAVERNOTSUPPORTED      (WSABASEERR+92)
#define WSANOTINITIALISED       (WSABASEERR+93)

/*
 * Error return codes from gethostbyname() and gethostbyaddr()
 * (when using the resolver). Note that these errors are
 * retrieved via WSAGetLastError() and must therefore follow
 * the rules for avoiding clashes with error numbers from
 * specific implementations or language run-time systems.
 * For this reason the codes are based at WSABASEERR+1001.
 * Note also that [WSA]NO_ADDRESS is defined only for
 * compatibility purposes.
 */

#define h_errno         WSAGetLastError()

/* Authoritative Answer: Host not found */
#define WSAHOST_NOT_FOUND       (WSABASEERR+1001)
#define HOST_NOT_FOUND          WSAHOST_NOT_FOUND

/* Non-Authoritative: Host not found, or SERVERFAIL */
#define WSATRY_AGAIN            (WSABASEERR+1002)
#define TRY_AGAIN               WSATRY_AGAIN

/* Non recoverable errors, FORMERR, REFUSED, NOTIMP */
#define WSANO_RECOVERY          (WSABASEERR+1003)
#define NO_RECOVERY             WSANO_RECOVERY

/* Valid name, no data record of requested type */
#define WSANO_DATA              (WSABASEERR+1004)
```

```
#define NO_DATA                 WSANO_DATA

/* no address, look for MX record */
#define WSANO_ADDRESS           WSANO_DATA
#define NO_ADDRESS              WSANO_ADDRESS

/*
 * Windows Sockets errors redefined as regular Berkeley error constants
 */
#define EWOULDBLOCK             WSAEWOULDBLOCK
#define EINPROGRESS             WSAEINPROGRESS
#define EALREADY                WSAEALREADY
#define ENOTSOCK                WSAENOTSOCK
#define EDESTADDRREQ            WSAEDESTADDRREQ
#define EMSGSIZE                WSAEMSGSIZE
#define EPROTOTYPE              WSAEPROTOTYPE
#define ENOPROTOOPT             WSAENOPROTOOPT
#define EPROTONOSUPPORT         WSAEPROTONOSUPPORT
#define ESOCKTNOSUPPORT         WSAESOCKTNOSUPPORT
#define EOPNOTSUPP              WSAEOPNOTSUPP
#define EPFNOSUPPORT            WSAEPFNOSUPPORT
#define EAFNOSUPPORT            WSAEAFNOSUPPORT
#define EADDRINUSE              WSAEADDRINUSE
#define EADDRNOTAVAIL           WSAEADDRNOTAVAIL
#define ENETDOWN                WSAENETDOWN
#define ENETUNREACH             WSAENETUNREACH
#define ENETRESET               WSAENETRESET
#define ECONNABORTED            WSAECONNABORTED
#define ECONNRESET              WSAECONNRESET
#define ENOBUFS                 WSAENOBUFS
#define EISCONN                 WSAEISCONN
#define ENOTCONN                WSAENOTCONN
#define ESHUTDOWN               WSAESHUTDOWN
#define ETOOMANYREFS            WSAETOOMANYREFS
#define ETIMEDOUT               WSAETIMEDOUT
#define ECONNREFUSED            WSAECONNREFUSED
#define ELOOP                   WSAELOOP
#define ENAMETOOLONG            WSAENAMETOOLONG
```

```
#define EHOSTDOWN               WSAEHOSTDOWN
#define EHOSTUNREACH            WSAEHOSTUNREACH
#define ENOTEMPTY               WSAENOTEMPTY
#define EPROCLIM                WSAEPROCLIM
#define EUSERS                  WSAEUSERS
#define EDQUOT                  WSAEDQUOT
#define ESTALE                  WSAESTALE
#define EREMOTE                 WSAEREMOTE

/* Socket function prototypes */

#ifdef __cplusplus
extern "C" {
#endif

SOCKET PASCAL FAR accept (SOCKET s, struct sockaddr FAR *addr,
                          int FAR *addrlen);

int PASCAL FAR bind (SOCKET s, const struct sockaddr FAR *addr, int
namelen);

int PASCAL FAR closesocket (SOCKET s);

int PASCAL FAR connect (SOCKET s, const struct sockaddr FAR *name, int
namelen);

int PASCAL FAR ioctlsocket (SOCKET s, long cmd, u_long FAR *argp);

int PASCAL FAR getpeername (SOCKET s, struct sockaddr FAR *name,
                            int FAR * namelen);

int PASCAL FAR getsockname (SOCKET s, struct sockaddr FAR *name,
                            int FAR * namelen);

int PASCAL FAR getsockopt (SOCKET s, int level, int optname,
                           char FAR * optval, int FAR *optlen);

u_long PASCAL FAR htonl (u_long hostlong);
```

```
u_short PASCAL FAR htons (u_short hostshort);

unsigned long PASCAL FAR inet_addr (const char FAR * cp);

char FAR * PASCAL FAR inet_ntoa (struct in_addr in);

int PASCAL FAR listen (SOCKET s, int backlog);

u_long PASCAL FAR ntohl (u_long netlong);

u_short PASCAL FAR ntohs (u_short netshort);

int PASCAL FAR recv (SOCKET s, char FAR * buf, int len, int flags);

int PASCAL FAR recvfrom (SOCKET s, char FAR * buf, int len, int flags,
                         struct sockaddr FAR *from, int FAR * fromlen);

int PASCAL FAR select (int nfds, fd_set FAR *readfds, fd_set FAR
*writefds,
                       fd_set FAR *exceptfds, const struct timeval FAR
*timeout);

int PASCAL FAR send (SOCKET s, const char FAR * buf, int len, int flags);

int PASCAL FAR sendto (SOCKET s, const char FAR * buf, int len, int flags,
                       const struct sockaddr FAR *to, int tolen);

int PASCAL FAR setsockopt (SOCKET s, int level, int optname,
                           const char FAR * optval, int optlen);

int PASCAL FAR shutdown (SOCKET s, int how);

SOCKET PASCAL FAR socket (int af, int type, int protocol);

/* Database function prototypes */

struct hostent FAR * PASCAL FAR gethostbyaddr(const char FAR * addr,
```

```
                                        int len, int type);

struct hostent FAR * PASCAL FAR gethostbyname(const char FAR * name);

int PASCAL FAR gethostname (char FAR * name, int namelen);

struct servent FAR * PASCAL FAR getservbyport(int port, const char FAR
* proto);

struct servent FAR * PASCAL FAR getservbyname(const char FAR * name,
                                        const char FAR * proto);

struct protoent FAR * PASCAL FAR getprotobynumber(int proto);

struct protoent FAR * PASCAL FAR getprotobyname(const char FAR * name);

/* Microsoft Windows Extension function prototypes */

int PASCAL FAR WSAStartup(WORD wVersionRequired, LPWSADATA lpWSAData);

int PASCAL FAR WSACleanup(void);

void PASCAL FAR WSASetLastError(int iError);

int PASCAL FAR WSAGetLastError(void);

BOOL PASCAL FAR WSAIsBlocking(void);

int PASCAL FAR WSAUnhookBlockingHook(void);

FARPROC PASCAL FAR WSASetBlockingHook(FARPROC lpBlockFunc);

int PASCAL FAR WSACancelBlockingCall(void);

HANDLE PASCAL FAR WSAAsyncGetServByName(HWND hWnd, u_int wMsg,
                                    const char FAR * name,
                                    const char FAR * proto,
                                    char FAR * buf, int buflen);
```

```
HANDLE PASCAL FAR WSAAsyncGetServByPort(HWND hWnd, u_int wMsg, int port,
                                        const char FAR * proto, char FAR
* buf,
                                          int buflen);

HANDLE PASCAL FAR WSAAsyncGetProtoByName(HWND hWnd, u_int wMsg,
                                       const char FAR * name, char FAR
* buf,
                                         int buflen);

HANDLE PASCAL FAR WSAAsyncGetProtoByNumber(HWND hWnd, u_int wMsg,
                                           int number, char FAR * buf,
                                            int buflen);

HANDLE PASCAL FAR WSAAsyncGetHostByName(HWND hWnd, u_int wMsg,
                                       const char FAR * name, char FAR
* buf,
                                         int buflen);

HANDLE PASCAL FAR WSAAsyncGetHostByAddr(HWND hWnd, u_int wMsg,
                                       const char FAR * addr, int len,
int type,
                                         const char FAR * buf, int
buflen);

int PASCAL FAR WSACancelAsyncRequest(HANDLE hAsyncTaskHandle);

int PASCAL FAR WSAAsyncSelect(SOCKET s, HWND hWnd, u_int wMsg,
                              long lEvent);

#ifdef __cplusplus
}
#endif

/* Microsoft Windows Extended data types */
typedef struct sockaddr SOCKADDR;
typedef struct sockaddr *PSOCKADDR;
```

```
typedef struct sockaddr FAR *LPSOCKADDR;

typedef struct sockaddr_in SOCKADDR_IN;
typedef struct sockaddr_in *PSOCKADDR_IN;
typedef struct sockaddr_in FAR *LPSOCKADDR_IN;

typedef struct linger LINGER;
typedef struct linger *PLINGER;
typedef struct linger FAR *LPLINGER;

typedef struct in_addr IN_ADDR;
typedef struct in_addr *PIN_ADDR;
typedef struct in_addr FAR *LPIN_ADDR;

typedef struct fd_set FD_SET;
typedef struct fd_set *PFD_SET;
typedef struct fd_set FAR *LPFD_SET;

typedef struct hostent HOSTENT;
typedef struct hostent *PHOSTENT;
typedef struct hostent FAR *LPHOSTENT;

typedef struct servent SERVENT;
typedef struct servent *PSERVENT;
typedef struct servent FAR *LPSERVENT;

typedef struct protoent PROTOENT;
typedef struct protoent *PPROTOENT;
typedef struct protoent FAR *LPPROTOENT;

typedef struct timeval TIMEVAL;
typedef struct timeval *PTIMEVAL;
typedef struct timeval FAR *LPTIMEVAL;

/*
 * Windows message parameter composition and decomposition
 * macros.
 *
```

```
 * WSAMAKEASYNCREPLY is intended for use by the Windows Sockets
implementation
 * when constructing the response to a WSAAsyncGetXByY() routine.
 */
#define WSAMAKEASYNCREPLY(buflen,error)     MAKELONG(buflen,error)
/*
 * WSAMAKESELECTREPLY is intended for use by the Windows Sockets
implementation
 * when constructing the response to WSAAsyncSelect().
 */
#define WSAMAKESELECTREPLY(event,error)     MAKELONG(event,error)
/*
 * WSAGETASYNCBUFLEN is intended for use by the Windows Sockets
application
 * to extract the buffer length from the lParam in the response
 * to a WSAGetXByY().
 */
#define WSAGETASYNCBUFLEN(lParam)           LOWORD(lParam)
/*
 * WSAGETASYNCERROR is intended for use by the Windows Sockets application
 * to extract the error code from the lParam in the response
 * to a WSAGetXByY().
 */
#define WSAGETASYNCERROR(lParam)            HIWORD(lParam)
/*
 * WSAGETSELECTEVENT is intended for use by the Windows Sockets
application
 * to extract the event code from the lParam in the response
 * to a WSAAsyncSelect().
 */
#define WSAGETSELECTEVENT(lParam)           LOWORD(lParam)
/*
 * WSAGETSELECTERROR is intended for use by the Windows Sockets
application
 * to extract the error code from the lParam in the response
 * to a WSAAsyncSelect().
 */
#define WSAGETSELECTERROR(lParam)           HIWORD(lParam)
```

```
#endif  /* _WINSOCKAPI_ */
```

附录 B.2 Windows Sockets 2 头文件

```
/* Winsock2.h—definitions to be used with the WinSock 2.0
DLL and
 *             WinSock 2.0 applications.
 *
 * This header file corresponds to version 2.0 of the
WinSock specification.
 *
 * This file includes parts which are Copyright (c) 1982-
1986 Regents
 * of the University of California.  All rights reserved.
The
 * Berkeley Software License Agreement specifies the terms
and
 * conditions for redistribution.
 */

#ifndef _WINSOCK2API_
#define _WINSOCK2API_

/*
 * Pull in WINDOWS.H if necessary
 */
#ifndef _INC_WINDOWS
#include <windows.h>
#endif /* _INC_WINDOWS */

/*
 * Basic system type definitions, taken from the BSD file
sys/types.h.
 */
typedef unsigned char   u_char;
typedef unsigned short  u_short;
typedef unsigned int    u_int;
typedef unsigned long   u_long;
```

```
/*
 * The new type to be used in all
 * instances which refer to sockets.
 */
typedef u_int           SOCKET;

/*
 * Select uses arrays of SOCKETs.  These macros manipulate
such
 * arrays.  FD_SETSIZE may be defined by the user before
including
 * this file, but the default here should be >= 64.
 *
 * CAVEAT IMPLEMENTOR and USER: THESE MACROS AND TYPES MUST
BE
 * INCLUDED IN WINSOCK.H EXACTLY AS SHOWN HERE.
 */
#ifndef FD_SETSIZE
#define FD_SETSIZE      64
#endif /* FD_SETSIZE */

typedef struct fd_set {
        u_short fd_count;               /* how many are SET?
*/
        SOCKET  fd_array[FD_SETSIZE];   /* an array of
SOCKETs */
} fd_set;

extern int PASCAL FAR __WSAFDIsSet(SOCKET, fd_set FAR *);

#define FD_CLR(fd, set) do { \
    u_int __i; \
    for (__i = 0; __i < ((fd_set FAR *)(set))->fd_count ;
__i++) { \
        if (((fd_set FAR *)(set))->fd_array[__i] == fd) { \
            while (__i < ((fd_set FAR *)(set))->fd_count-1)
{ \
                ((fd_set FAR *)(set))->fd_array[__i] = \
```

```
                    ((fd_set FAR *)(set))->fd_array[__i+1];
\
                __i++; \
            } \
            ((fd_set FAR *)(set))->fd_count—; \
            break; \
        } \
    } \
} while(0)

#define FD_SET(fd, set) do { \
    if (((fd_set FAR *)(set))->fd_count < FD_SETSIZE) \
        ((fd_set FAR *)(set))->fd_array[((fd_set FAR
*)(set))->fd_count++]=fd;\
} while(0)

#define FD_ZERO(set) (((fd_set FAR *)(set))->fd_count=0)

#define FD_ISSET(fd, set) __WSAFDIsSet((SOCKET)fd, (fd_set
FAR *)set)

/*
 * Structure used in select() call, taken from the BSD file
sys/time.h.
 */
struct timeval {
        long    tv_sec;         /* seconds */
        long    tv_usec;        /* and microseconds */
};

/*
 * Operations on timevals.
 *
 * NB: timercmp does not work for >= or <=.
 */
#define timerisset(tvp)         ((tvp)->tv_sec || (tvp)-
>tv_usec)
#define timercmp(tvp, uvp, cmp) \
```

```
        ((tvp)->tv_sec cmp (uvp)->tv_sec || \
         (tvp)->tv_sec == (uvp)->tv_sec && (tvp)->tv_usec
cmp (uvp)->tv_usec)
#define timerclear(tvp)         (tvp)->tv_sec = (tvp)-
>tv_usec = 0

/*
 * Commands for ioctlsocket(),  taken from the BSD file
fcntl.h.
 *
 *
 * Ioctl's have the command encoded in the lower word,
 * and the size of any in or out parameters in the upper
 * word.  The high 2 bits of the upper word are used
 * to encode the in/out status of the parameter; for now
 * we restrict parameters to at most 128 bytes.
 */
#define IOCPARM_MASK    0x7f            /* parameters must
be < 128 bytes */
#define IOC_VOID        0x20000000      /* no parameters */
#define IOC_OUT         0x40000000      /* copy out
parameters */
#define IOC_IN          0x80000000      /* copy in
parameters */
#define IOC_INOUT       (IOC_IN|IOC_OUT)
                                        /* 0x20000000
distinguishes new &
                                           old ioctl's */
#define _IO(x,y)        (IOC_VOID|(x<<8)|y)

#define _IOR(x,y,t)
(IOC_OUT|(((long)sizeof(t)&IOCPARM_MASK)<<16)|(x<<8)|y)

#define _IOW(x,y,t)
(IOC_IN|(((long)sizeof(t)&IOCPARM_MASK)<<16)|(x<<8)|y)

#define FIONREAD    _IOR('f', 127, u_long) /* get # bytes to
read */
```

```
#define FIONBIO     _IOW('f', 126, u_long) /* set/clear non-
blocking i/o */
#define FIOASYNC    _IOW('f', 125, u_long) /* set/clear
async i/o */

/* Socket I/O Controls */
#define SIOCSHIWAT  _IOW('s',  0, u_long)  /* set high
watermark */
#define SIOCGHIWAT  _IOR('s',  1, u_long)  /* get high
watermark */
#define SIOCSLOWAT  _IOW('s',  2, u_long)  /* set low
watermark */
#define SIOCGLOWAT  _IOR('s',  3, u_long)  /* get low
watermark */
#define SIOCATMARK  _IOR('s',  7, u_long)  /* at oob mark?
*/

/*
 * Structures returned by network data base library, taken
from the
 * BSD file netdb.h.  All addresses are supplied in host
order, and
 * returned in network order (suitable for use in system
calls).
 */

struct  hostent {
        char    FAR * h_name;           /* official name of
host */
        char    FAR * FAR * h_aliases;  /* alias list */
        short   h_addrtype;             /* host address type
*/
        short   h_length;               /* length of address
*/
        char    FAR * FAR * h_addr_list; /* list of
addresses */
#define h_addr  h_addr_list[0]          /* address, for
backward compat */
```

```
};

/*
 * It is assumed here that a network number
 * fits in 32 bits.
 */
struct  netent {
        char    FAR * n_name;           /* official name of
net */
        char    FAR * FAR * n_aliases;  /* alias list */
        short   n_addrtype;             /* net address type
*/
        u_long  n_net;                  /* network # */
};

struct  servent {
        char    FAR * s_name;           /* official service
name */
        char    FAR * FAR * s_aliases;  /* alias list */
        short   s_port;                 /* port # */
        char    FAR * s_proto;          /* protocol to use
*/
};

struct  protoent {
        char    FAR * p_name;           /* official protocol
name */
        char    FAR * FAR * p_aliases;  /* alias list */
        short   p_proto;                /* protocol # */
};

/*
 * Constants and structures defined by the internet system,
 * Per RFC 790, September 1981, taken from the BSD file
netinet/in.h.
 */

/*
```

```
 * Protocols
 */
#define IPPROTO_IP              0               /* dummy for
IP */
#define IPPROTO_ICMP            1               /* control
message protocol */
#define IPPROTO_IGMP            2               /* internet
group management protocol */
#define IPPROTO_GGP             3               /* gateway^2
(deprecated) */
#define IPPROTO_TCP             6               /* tcp */
#define IPPROTO_PUP             12              /* pup */
#define IPPROTO_UDP             17              /* user
datagram protocol */
#define IPPROTO_IDP             22              /* xns idp
*/
#define IPPROTO_ND              77              /*
UNOFFICIAL net disk proto */

#define IPPROTO_RAW             255             /* raw IP
packet */
#define IPPROTO_MAX             256

/*
 * Port/socket numbers: network standard functions
 */
#define IPPORT_ECHO             7
#define IPPORT_DISCARD          9
#define IPPORT_SYSTAT           11
#define IPPORT_DAYTIME          13
#define IPPORT_NETSTAT          15
#define IPPORT_FTP              21
#define IPPORT_TELNET           23
#define IPPORT_SMTP             25
#define IPPORT_TIMESERVER       37
#define IPPORT_NAMESERVER       42
#define IPPORT_WHOIS            43
#define IPPORT_MTP              57
```

```
/*
 * Port/socket numbers: host specific functions
 */
#define IPPORT_TFTP             69
#define IPPORT_RJE              77
#define IPPORT_FINGER           79
#define IPPORT_TTYLINK          87
#define IPPORT_SUPDUP           95

/*
 * UNIX TCP sockets
 */
#define IPPORT_EXECSERVER       512
#define IPPORT_LOGINSERVER      513
#define IPPORT_CMDSERVER        514
#define IPPORT_EFSSERVER        520

/*
 * UNIX UDP sockets
 */
#define IPPORT_BIFFUDP          512
#define IPPORT_WHOSERVER        513
#define IPPORT_ROUTESERVER      520
                                        /* 520+1 also used
*/

/*
 * Ports < IPPORT_RESERVED are reserved for
 * privileged processes (e.g. root).
 */
#define IPPORT_RESERVED         1024

/*
 * Link numbers
 */
#define IMPLINK_IP              155
#define IMPLINK_LOWEXPER        156
```

```
#define IMPLINK_HIGHEXPER       158

/*
 * Internet address (old style... should be updated)
 */
struct in_addr {
        union {
                struct { u_char s_b1,s_b2,s_b3,s_b4; }
S_un_b;
                struct { u_short s_w1,s_w2; } S_un_w;
                u_long S_addr;
        } S_un;
#define s_addr  S_un.S_addr
                                /* can be used for most tcp
& ip code */
#define s_host  S_un.S_un_b.s_b2
                                /* host on imp */
#define s_net   S_un.S_un_b.s_b1
                                /* network */
#define s_imp   S_un.S_un_w.s_w2
                                /* imp */
#define s_impno S_un.S_un_b.s_b4
                                /* imp # */
#define s_lh    S_un.S_un_b.s_b3
                                /* logical host */
};

/*
 * Definitions of bits in internet address integers.
 * On subnets, the decomposition of addresses to host and
net parts
 * is done according to subnet mask, not the masks here.
 */
#define IN_CLASSA(i)            (((long)(i) & 0x80000000) ==
0)
#define IN_CLASSA_NET           0xff000000
#define IN_CLASSA_NSHIFT        24
#define IN_CLASSA_HOST          0x00ffffff
```

```
#define IN_CLASSA_MAX           128

#define IN_CLASSB(i)            (((long)(i) & 0xc0000000) ==
0x80000000)
#define IN_CLASSB_NET           0xffff0000
#define IN_CLASSB_NSHIFT        16
#define IN_CLASSB_HOST          0x0000ffff
#define IN_CLASSB_MAX           65536

#define IN_CLASSC(i)            (((long)(i) & 0xc0000000) ==
0xc0000000)
#define IN_CLASSC_NET           0xffffff00
#define IN_CLASSC_NSHIFT        8
#define IN_CLASSC_HOST          0x000000ff

#define INADDR_ANY              (u_long)0x00000000
#define INADDR_LOOPBACK         0x7f000001
#define INADDR_BROADCAST        (u_long)0xffffffff
#define INADDR_NONE             0xffffffff

/*
 * Socket address, internet style.
 */
struct sockaddr_in {
        short   sin_family;
        u_short sin_port;
        struct  in_addr sin_addr;
        char    sin_zero[8];
};

#define WSADESCRIPTION_LEN      256
#define WSASYS_STATUS_LEN       128

typedef struct WSAData {
        WORD                    wVersion;
        WORD                    wHighVersion;
        char
szDescription[WSADESCRIPTION_LEN+1];
```

```
        char
szSystemStatus[WSASYS_STATUS_LEN+1];
        unsigned short          iMaxSockets;
        unsigned short          iMaxUdpDg;
        char FAR *              lpVendorInfo;
} WSADATA, FAR * LPWSADATA;

typedef WSADATA FAR *LPWSADATA;

#if !defined(MAKEWORD)
     #define MAKEWORD(low,high) \
          ((WORD)((BYTE)(low)) | (((WORD)(BYTE)(high))<<8)))
#endif


/*
 * Options for use with [gs]etsockopt at the IP level.
 */
#define IP_OPTIONS      1               /* set/get IP per-
packet options */

/*
 * Definitions related to sockets: types, address families,
options,
 * taken from the BSD file sys/socket.h.
 */

/*
 * This is used instead of -1, since the
 * SOCKET type is unsigned.
 */
#define INVALID_SOCKET  (SOCKET)(~0)
#define SOCKET_ERROR            (-1)

/*
 * Types
 */
#define SOCK_STREAM     1               /* stream socket */
```

```
#define SOCK_DGRAM      2               /* datagram socket
*/
#define SOCK_RAW        3               /* raw-protocol
interface */
#define SOCK_RDM        4               /* reliably-
delivered message */
#define SOCK_SEQPACKET  5               /* sequenced packet
stream */

/*
 * Option flags per-socket.
 */
#define SO_DEBUG        0x0001          /* turn on debugging
info recording */
#define SO_ACCEPTCONN   0x0002          /* socket has had
listen() */
#define SO_REUSEADDR    0x0004          /* allow local
address reuse */
#define SO_KEEPALIVE    0x0008          /* keep connections
alive */
#define SO_DONTROUTE    0x0010          /* just use
interface addresses */
#define SO_BROADCAST    0x0020          /* permit sending of
broadcast msgs */
#define SO_USELOOPBACK  0x0040          /* bypass hardware
when possible */
#define SO_LINGER       0x0080          /* linger on close
if data present */
#define SO_OOBINLINE    0x0100          /* leave received
OOB data in line */

#define SO_DONTLINGER   (int)(~SO_LINGER)

/*
 * Additional options.
 */
#define SO_SNDBUF       0x1001          /* send buffer size
*/
```

```
#define SO_RCVBUF       0x1002          /* receive buffer
size */
#define SO_SNDLOWAT     0x1003          /* send low-water
mark */
#define SO_RCVLOWAT     0x1004          /* receive low-water
mark */
#define SO_SNDTIMEO     0x1005          /* send timeout */
#define SO_RCVTIMEO     0x1006          /* receive timeout
*/
#define SO_ERROR        0x1007          /* get error status
and clear */
#define SO_TYPE         0x1008          /* get socket type
*/

/*
 * WinSock 2.0 extension — new options
 */
#define SO_GROUP_ID       0x2001          /* ID of a socket
group */
#define SO_GROUP_PRIORITY 0x2002          /* the relative
priority within a group */
#define SO_MAX_MSG_SIZE   0x2003          /* maximum message
size */
#define SO_PROTOCOL_INFO  0x2004          /* PROTOCOL_INFO
structure */
#define PVD_CONFIG        0x3001          /* configuration
info for service provider */

/*
 * TCP options.
 */
#define TCP_NODELAY     0x0001

/*
 * Address families.
 */
#define AF_UNSPEC       0               /* unspecified */
#define AF_UNIX         1               /* local to host
```

```
(pipes, portals) */
#define AF_INET         2               /* internetwork:
UDP, TCP, etc. */
#define AF_IMPLINK      3               /* arpanet imp
addresses */
#define AF_PUP          4               /* pup protocols:
e.g. BSP */
#define AF_CHAOS        5               /* mit CHAOS
protocols */
#define AF_NS           6               /* XEROX NS
protocols */
#define AF_IPX          AF_NS           /* IPX protocols:
IPX, SPX, etc. */
#define AF_ISO          7               /* ISO protocols */
#define AF_OSI          AF_ISO          /* OSI is ISO */
#define AF_ECMA         8               /* european computer
manufacturers */
#define AF_DATAKIT      9               /* datakit protocols
*/
#define AF_CCITT        10              /* CCITT protocols,
X.25 etc */
#define AF_SNA          11              /* IBM SNA */
#define AF_DECnet       12              /* DECnet */
#define AF_DLI          13              /* Direct data link
interface */
#define AF_LAT          14              /* LAT */
#define AF_HYLINK       15              /* NSC Hyperchannel
*/
#define AF_APPLETALK    16              /* AppleTalk */
#define AF_NETBIOS      17              /* NetBios-style
addresses */
#define AF_FIREFOX      18              /* Protocols from
Firefox */
#define AF_VOICEVIEW    19              /* VoiceView */

#define AF_MAX          20

/*
```

```
 * Structure used by kernel to store most
 * addresses.
 */
struct sockaddr {
        u_short sa_family;              /* address family */
        char    sa_data[14];            /* up to 14 bytes of
direct address */
};

/*
 * Structure used by kernel to pass protocol
 * information in raw sockets.
 */
struct sockproto {
        u_short sp_family;              /* address family */
        u_short sp_protocol;            /* protocol */
};

/*
 * Protocol families, same as address families for now.
 */
#define PF_UNSPEC       AF_UNSPEC
#define PF_UNIX         AF_UNIX
#define PF_INET         AF_INET
#define PF_IMPLINK      AF_IMPLINK
#define PF_PUP          AF_PUP
#define PF_CHAOS        AF_CHAOS
#define PF_NS           AF_NS
#define PF_IPX          AF_IPX
#define PF_ISO          AF_ISO
#define PF_OSI          AF_OSI
#define PF_ECMA         AF_ECMA
#define PF_DATAKIT      AF_DATAKIT
#define PF_CCITT        AF_CCITT
#define PF_SNA          AF_SNA
#define PF_DECnet       AF_DECnet
#define PF_DLI          AF_DLI
#define PF_LAT          AF_LAT
```

```
#define PF_HYLINK       AF_HYLINK
#define PF_APPLETALK    AF_APPLETALK
#define PF_FIREFOX      AF_FIREFOX
#define PF_VOICEVIEW    AF_VOICEVIEW

#define PF_MAX          AF_MAX

/*
 * Structure used for manipulating linger option.
 */
struct  linger {
        u_short l_onoff;                /* option on/off */
        u_short l_linger;               /* linger time */
};

/*
 * Level number for (get/set)sockopt() to apply to socket
itself.
 */
#define SOL_SOCKET      0xffff          /* options for
socket level */

/*
 * Maximum queue length specifiable by listen.
 */
#define SOMAXCONN       5

#define MSG_OOB         0x1             /* process out-of-
band data */
#define MSG_PEEK        0x2             /* peek at incoming
message */
#define MSG_DONTROUTE   0x4             /* send without
using routing tables */

#define MSG_MAXIOVLEN   16

/*
 * Define constant based on rfc883, used by gethostbyxxxx()
```

```
calls.
 */
#define MAXGETHOSTSTRUCT        1024

/*
 * Define flags to be used with the WSAAsyncSelect() and
WSAEventSelect() call.
 */
#define FD_READ         0x01
#define FD_WRITE        0x02
#define FD_OOB          0x04
#define FD_ACCEPT       0x08
#define FD_CONNECT      0x10
#define FD_CLOSE        0x20

/*
 * WinSock 2.0 extension — new flags for WSAAsyncSelect()
and WSAEventSelect()
 */
#define FD_QOS          0x40
#define FD_GROUP_QOS    0x80

/*
 * All Windows Sockets error constants are biased by
WSABASEERR from
 * the "normal"
 */
#define WSABASEERR              10000
/*
 * Windows Sockets definitions of regular Microsoft C error
constants
 */
#define WSAEINTR                (WSABASEERR+4)
#define WSAEBADF                (WSABASEERR+9)
#define WSAEACCES               (WSABASEERR+13)
#define WSAEFAULT               (WSABASEERR+14)
#define WSAEINVAL               (WSABASEERR+22)
#define WSAEMFILE               (WSABASEERR+24)
```

```
/*
 * Windows Sockets definitions of regular Berkeley error
constants
 */
#define WSAEWOULDBLOCK          (WSABASEERR+35)
#define WSAEINPROGRESS          (WSABASEERR+36)
#define WSAEALREADY             (WSABASEERR+37)
#define WSAENOTSOCK             (WSABASEERR+38)
#define WSAEDESTADDRREQ         (WSABASEERR+39)
#define WSAEMSGSIZE             (WSABASEERR+40)
#define WSAEPROTOTYPE           (WSABASEERR+41)
#define WSAENOPROTOOPT          (WSABASEERR+42)
#define WSAEPROTONOSUPPORT      (WSABASEERR+43)
#define WSAESOCKTNOSUPPORT      (WSABASEERR+44)
#define WSAEOPNOTSUPP           (WSABASEERR+45)
#define WSAEPFNOSUPPORT         (WSABASEERR+46)
#define WSAEAFNOSUPPORT         (WSABASEERR+47)
#define WSAEADDRINUSE           (WSABASEERR+48)
#define WSAEADDRNOTAVAIL        (WSABASEERR+49)
#define WSAENETDOWN             (WSABASEERR+50)
#define WSAENETUNREACH          (WSABASEERR+51)
#define WSAENETRESET            (WSABASEERR+52)
#define WSAECONNABORTED         (WSABASEERR+53)
#define WSAECONNRESET           (WSABASEERR+54)
#define WSAENOBUFS              (WSABASEERR+55)
#define WSAEISCONN              (WSABASEERR+56)
#define WSAENOTCONN             (WSABASEERR+57)
#define WSAESHUTDOWN            (WSABASEERR+58)
#define WSAETOOMANYREFS         (WSABASEERR+59)
#define WSAETIMEDOUT            (WSABASEERR+60)
#define WSAECONNREFUSED         (WSABASEERR+61)
#define WSAELOOP                (WSABASEERR+62)
#define WSAENAMETOOLONG         (WSABASEERR+63)
#define WSAEHOSTDOWN            (WSABASEERR+64)
#define WSAEHOSTUNREACH         (WSABASEERR+65)
#define WSAENOTEMPTY            (WSABASEERR+66)
#define WSAEPROCLIM             (WSABASEERR+67)
```

```
#define WSAEUSERS               (WSABASEERR+68)
#define WSAEDQUOT               (WSABASEERR+69)
#define WSAESTALE               (WSABASEERR+70)
#define WSAEREMOTE              (WSABASEERR+71)

/*
 * Extended Windows Sockets error constant definitions
 */
#define WSASYSNOTREADY          (WSABASEERR+91)
#define WSAVERNOTSUPPORTED      (WSABASEERR+92)
#define WSANOTINITIALISED       (WSABASEERR+93)

/*
 * Error return codes from gethostbyname() and
gethostbyaddr()
 * (when using the resolver). Note that these errors are
 * retrieved via WSAGetLastError() and must therefore follow
 * the rules for avoiding clashes with error numbers from
 * specific implementations or language run-time systems.
 * For this reason the codes are based at WSABASEERR+1001.
 * Note also that [WSA]NO_ADDRESS is defined only for
 * compatibility purposes.
 */

#define h_errno         WSAGetLastError()

/* Authoritative Answer: Host not found */
#define WSAHOST_NOT_FOUND       (WSABASEERR+1001)
#define HOST_NOT_FOUND          WSAHOST_NOT_FOUND

/* Non-Authoritative: Host not found, or SERVERFAIL */
#define WSATRY_AGAIN            (WSABASEERR+1002)
#define TRY_AGAIN               WSATRY_AGAIN

/* Non-recoverable errors, FORMERR, REFUSED, NOTIMP */
#define WSANO_RECOVERY          (WSABASEERR+1003)
#define NO_RECOVERY             WSANO_RECOVERY
```

```
/* Valid name, no data record of requested type */
#define WSANO_DATA              (WSABASEERR+1004)
#define NO_DATA                 WSANO_DATA

/* no address, look for MX record */
#define WSANO_ADDRESS           WSANO_DATA
#define NO_ADDRESS              WSANO_ADDRESS

/*
 * Windows Sockets errors redefined as regular Berkeley
error constants
 */
#define EWOULDBLOCK             WSAEWOULDBLOCK
#define EINPROGRESS             WSAEINPROGRESS
#define EALREADY                WSAEALREADY
#define ENOTSOCK                WSAENOTSOCK
#define EDESTADDRREQ            WSAEDESTADDRREQ
#define EMSGSIZE                WSAEMSGSIZE
#define EPROTOTYPE              WSAEPROTOTYPE
#define ENOPROTOOPT             WSAENOPROTOOPT
#define EPROTONOSUPPORT         WSAEPROTONOSUPPORT
#define ESOCKTNOSUPPORT         WSAESOCKTNOSUPPORT
#define EOPNOTSUPP              WSAEOPNOTSUPP
#define EPFNOSUPPORT            WSAEPFNOSUPPORT
#define EAFNOSUPPORT            WSAEAFNOSUPPORT
#define EADDRINUSE              WSAEADDRINUSE
#define EADDRNOTAVAIL           WSAEADDRNOTAVAIL
#define ENETDOWN                WSAENETDOWN
#define ENETUNREACH             WSAENETUNREACH
#define ENETRESET               WSAENETRESET
#define ECONNABORTED            WSAECONNABORTED
#define ECONNRESET              WSAECONNRESET
#define ENOBUFS                 WSAENOBUFS
#define EISCONN                 WSAEISCONN
#define ENOTCONN                WSAENOTCONN
#define ESHUTDOWN               WSAESHUTDOWN
#define ETOOMANYREFS            WSAETOOMANYREFS
#define ETIMEDOUT               WSAETIMEDOUT
```

```
#define ECONNREFUSED            WSAECONNREFUSED
#define ELOOP                   WSAELOOP
#define ENAMETOOLONG            WSAENAMETOOLONG
#define EHOSTDOWN               WSAEHOSTDOWN
#define EHOSTUNREACH            WSAEHOSTUNREACH
#define ENOTEMPTY               WSAENOTEMPTY
#define EPROCLIM                WSAEPROCLIM
#define EUSERS                  WSAEUSERS
#define EDQUOT                  WSAEDQUOT
#define ESTALE                  WSAESTALE
#define EREMOTE                 WSAEREMOTE

/*
 * WinSock 2.0 extension — new error codes and type
definition
 */

#ifdef WIN32

        #define WSAAPI                  FAR PASCAL
        #define WSATASK                 HANDLE
        #define WSAEVENT                HANDLE
        #define LPWSAEVENT              LPHANDLE
        #define WSAOVERLAPPED           OVERLAPPED
        #define LPWSAOVERLAPPED         LPOVERLAPPED

        #define WSA_IO_PENDING          (ERROR_IO_PENDING)
        #define WSA_IO_INCOMPLETE
(ERROR_IO_INCOMPLETE)
        #define WSA_INVALID_HANDLE
(ERROR_INVALID_HANDLE)
        #define WSA_INVALID_PARAMETER
(ERROR_INVALID_PARAMETER)
        #define WSA_NOT_ENOUGH_MEMORY
(ERROR_NOT_ENOUGH_MEMORY)
        #define WSAEDISCON              (WSABASEERR + 94)

        #define WSA_INVALID_EVENT       ((WSAEVENT)NULL)
```

```
        #define WSA_MAXIMUM_WAIT_EVENTS
(MAXIMUM_WAIT_OBJECTS)
        #define WSA_WAIT_FAILED         ((DWORD)-1L)
        #define WSA_WAIT_EVENT_0        (WAIT_OBJECT_0)
        #define WSA_WAIT_IO_COMPLETION  (WAIT_IO_COMPLETION)
        #define WSA_WAIT_TIMEOUT        (WAIT_TIMEOUT)
        #define WSA_INFINITE            (INFINITE)

#else // WIN16

        #define WSAAPI                  FAR PASCAL
        #define WSATASK                 HTASK
        typedef DWORD                   WSAEVENT, FAR *
LPWSAEVENT;

        typedef struct _WSAOVERLAPPED
        {
            DWORD    Internal;
            DWORD    InternalHigh;
            DWORD    Offset;
            DWORD    OffsetHigh;
            WSAEVENT hEvent;
        } WSAOVERLAPPED, FAR * LPWSAOVERLAPPED;

        #define WSA_IO_PENDING          (WSAEWOULDBLOCK)
        #define WSA_IO_INCOMPLETE       (WSAEWOULDBLOCK)
        #define WSA_INVALID_HANDLE      (WSAENOTSOCK)
        #define WSA_INVALID_PARAMETER   (WSAEINVAL)
        #define WSA_NOT_ENOUGH_MEMORY   (WSAENOBUFS)
        #define WSAEDISCON              (WSABASEERR + 94)

        #define WSA_INVALID_EVENT       ((WSAEVENT)NULL)
        #define WSA_MAXIMUM_WAIT_EVENTS
(MAXIMUM_WAIT_OBJECTS)
        #define WSA_WAIT_FAILED         ((DWORD)-1L)
        #define WSA_WAIT_EVENT_0        ((DWORD)0)
        #define WSA_WAIT_TIMEOUT        ((DWORD)0x102L)
        #define WSA_INFINITE            ((DWORD)-1L)
```

```
#endif  // WIN32

/*
 * WinSock 2.0 extension — WSABUF and QOS struct
 */

typedef struct _WSABUF {
    int         len;     // the length of the buffer
    char FAR *  buf;     // the pointer to the buffer
} WSABUF, FAR * LPWSABUF;

typedef enum
{
    BestEffortService,
    PredictiveService,
    GuaranteedService
} GUARANTEE;

typedef long int32;

typedef struct _flowspec
{
    int32        TokenRate;              // In Bytes/sec
    int32        TokenBucketSize;        // In Bytes
    int32        PeakBandwidth;          // In Bytes/sec
    int32        Latency;                // In microseconds
    int32        DelayVariation;         // In microseconds
    GUARANTEE    LevelOfGuarantee;       // Guaranteed,
Predictive or Best Effort
    int32        CostOfCall;             // Reserved for
future use, must be set to 0 now
    int32        NetworkAvailability;    // read-only: 1 if
accessible, 0 if not
} FLOWSPEC, FAR * LPFLOWSPEC;

typedef struct _QualityOfService
{
```

```
    WSABUF        SendingFlowspec;       // the flow spec
for data sending
    WSABUF        ReceivingFlowspec;     // the flow spec
for data receiving
} QOS, FAR * LPQOS;

/*
 * WinSock 2.0 extension — manifest constants for return
values of the condition function
 */
#define CF_ACCEPT       0x0000
#define CF_REJECT       0x0001
#define CF_DEFER        0x0002

/*
 * WinSock 2.0 extension — manifest constants for
shutdown()
 */
#define SD_RECEIVE      0x00
#define SD_SEND         0x01
#define SD_BOTH         0x02

/*
 * WinSock 2.0 extension — data type and manifest constants
for socket groups
 */
typedef unsigned int            GROUP;

#define SG_UNCONSTRAINED_GROUP   0x01
#define SG_CONSTRAINED_GROUP     0x02

/*
 * WinSock 2.0 extension — data type for
WSAEnumNetworkEvents()
 */
typedef struct _WSANETWORKEVENTS {
       long lNetworkEvent,
       int iErrorCode
```

```
} WSANETWORKEVENTS, FAR * LPWSANETWORKEVENTS;

/*
 * WinSock 2.0 extension — PROTOCOL_INFO structure and
associated manifest constants
 */
typedef struct _PROTOCOL_INFO {
    DWORD dwServiceFlags1;
    DWORD dwServiceFlags2;
    DWORD dwServiceFlags3;
    DWORD dwServiceFlags4;
    int iProviderID;
    int iVersion;
    int iAddressFamily;
    int iMaxSockAddr;
    int iMinSockAddr;
    int iSocketType;
    int iProtocol;
    int iNetworkByteOrder;
    int iSecurityScheme;
    BOOL bMultiple;
    BOOL bFirst;
    DWORD dwMessageSize;
    LPSTR lpProtocol;
} PROTOCOL_INFO, FAR * LPPROTOCOL_INFO;

#define XP1_CONNECTIONLESS                0x00000001
#define XP1_GUARANTEED_DELIVERY           0x00000002
#define XP1_GUARANTEED_ORDER              0x00000004
#define XP1_MESSAGE_ORIENTED              0x00000008
#define XP1_PSEUDO_STREAM                 0x00000010
#define XP1_GRACEFUL_CLOSE                0x00000020
#define XP1_EXPEDITED_DATA                0x00000040
#define XP1_CONNECT_DATA                  0x00000080
#define XP1_DISCONNECT_DATA               0x00000100
#define XP1_SUPPORTS_BROADCAST            0x00000200
#define XP1_SUPPORT_MULTIPOINT            0x00000400
#define XP1_MULTIPOINT_CONTROL_PLANE      0x00000800
```

```
#define XP1_MULTIPOINT_DATA_PLANE           0x00001000
#define XP1_QOS_SUPPORTED                   0x00002000
#define XP1_INTERRUPT                       0x00004000
#define XP1_UNI_SEND                        0x00008000
#define XP1_UNI_RECV                        0x00010000

/*
 * WinSock 2.0 extension — manifest constants for
WSAJoinLeaf()
 */
#define JL_SENDER_ONLY    0x01
#define JL_RECEIVER_ONLY  0x02
#define JL_BOTH           0x04

/*
 * WinSock 2.0 extension — manifest constants for
WSASocket()
 */
#define WSA_FLAG_OVERLAPPED           0x01
#define WSA_FLAG_MULTIPOINT_C_ROOT    0x02
#define WSA_FLAG_MULTIPOINT_C_LEAF    0x04
#define WSA_FLAG_MULTIPOINT_D_ROOT    0x08
#define WSA_FLAG_MULTIPOINT_D_LEAF    0x10

/* Socket function prototypes */

#ifdef __cplusplus
extern "C" {
#endif

SOCKET PASCAL FAR accept (SOCKET s, struct sockaddr FAR
*addr,
                          int FAR *addrlen);

int PASCAL FAR bind (SOCKET s, const struct sockaddr FAR
*addr, int namelen);

int PASCAL FAR closesocket (SOCKET s);
```

```
int PASCAL FAR connect (SOCKET s, const struct sockaddr FAR
*name, int namelen);

int PASCAL FAR ioctlsocket (SOCKET s, long cmd, u_long FAR
*argp);

int PASCAL FAR getpeername (SOCKET s, struct sockaddr FAR
*name,
                           int FAR * namelen);

int PASCAL FAR getsockname (SOCKET s, struct sockaddr FAR
*name,
                           int FAR * namelen);

int PASCAL FAR getsockopt (SOCKET s, int level, int optname,
                          char FAR * optval, int FAR
*optlen);

u_long PASCAL FAR htonl (u_long hostlong);

u_short PASCAL FAR htons (u_short hostshort);

unsigned long PASCAL FAR inet_addr (const char FAR * cp);

char FAR * PASCAL FAR inet_ntoa (struct in_addr in);

int PASCAL FAR listen (SOCKET s, int backlog);

u_long PASCAL FAR ntohl (u_long netlong);

u_short PASCAL FAR ntohs (u_short netshort);

int PASCAL FAR recv (SOCKET s, char FAR * buf, int len, int
flags);

int PASCAL FAR recvfrom (SOCKET s, char FAR * buf, int len,
int flags,
```

```
                        struct sockaddr FAR *from, int FAR
* fromlen);

int PASCAL FAR select (int nfds, fd_set FAR *readfds, fd_set
FAR *writefds,
                       fd_set FAR *exceptfds, const struct
timeval FAR *timeout);

int PASCAL FAR send (SOCKET s, const char FAR * buf, int
len, int flags);

int PASCAL FAR sendto (SOCKET s, const char FAR * buf, int
len, int flags,
                       const struct sockaddr FAR *to, int
tolen);

int PASCAL FAR setsockopt (SOCKET s, int level, int optname,
                           const char FAR * optval, int
optlen);

int PASCAL FAR shutdown (SOCKET s, int how);

SOCKET PASCAL FAR socket (int af, int type, int protocol);

/* Database function prototypes */

struct hostent FAR * PASCAL FAR gethostbyaddr(const char FAR
* addr,
                                              int len, int
type);

struct hostent FAR * PASCAL FAR gethostbyname(const char FAR
* name);

int PASCAL FAR gethostname (char FAR * name, int namelen);

struct servent FAR * PASCAL FAR getservbyport(int port,
const char FAR * proto);
```

```
struct servent FAR * PASCAL FAR getservbyname(const char FAR
* name,
                                              const char FAR
* proto);

struct protoent FAR * PASCAL FAR getprotobynumber(int
proto);

struct protoent FAR * PASCAL FAR getprotobyname(const char
FAR * name);

/* Microsoft Windows Extension function prototypes */

int PASCAL FAR WSAStartup(WORD wVersionRequired, LPWSADATA
lpWSAData);

int PASCAL FAR WSACleanup(void);

void PASCAL FAR WSASetLastError(int iError);

int PASCAL FAR WSAGetLastError(void);

BOOL PASCAL FAR WSAIsBlocking(void);

int PASCAL FAR WSAUnhookBlockingHook(void);

FARPROC PASCAL FAR WSASetBlockingHook(FARPROC lpBlockFunc);

int PASCAL FAR WSACancelBlockingCall(void);

HANDLE PASCAL FAR WSAAsyncGetServByName(HWND hWnd, u_int
wMsg,
                                        const char FAR *
name,
                                        const char FAR *
proto,
                                        char FAR * buf, int
```

```
buflen);

HANDLE PASCAL FAR WSAAsyncGetServByPort(HWND hWnd, u_int
wMsg, int port,
                                        const char FAR *
proto, char FAR * buf,
                                        int buflen);

HANDLE PASCAL FAR WSAAsyncGetProtoByName(HWND hWnd, u_int
wMsg,
                                         const char FAR *
name, char FAR * buf,
                                         int buflen);

HANDLE PASCAL FAR WSAAsyncGetProtoByNumber(HWND hWnd, u_int
wMsg,
                                           int number, char
FAR * buf,
                                           int buflen);

HANDLE PASCAL FAR WSAAsyncGetHostByName(HWND hWnd, u_int
wMsg,
                                        const char FAR *
name, char FAR * buf,
                                        int buflen);

HANDLE PASCAL FAR WSAAsyncGetHostByAddr(HWND hWnd, u_int
wMsg,
                                        const char FAR *
addr, int len, int type,
                                        const char FAR *
buf, int buflen);

int PASCAL FAR WSACancelAsyncRequest(HANDLE
hAsyncTaskHandle);

int PASCAL FAR WSAAsyncSelect(SOCKET s, HWND hWnd, u_int
wMsg,
```

```
                                long lEvent);

/*
 * WinSock 2.0 extension — data type for the condition
function in WSAAccept()
 */

typedef int (CALLBACK * LPCONDITIONPROC) (LPWSABUF
lpCallerId,
                                          LPWSABUF
lpCallerData,
                                          LPQOS lpSQOS,
                                          LPQOS lpGQOS,
                                          LPWSABUF
lpCalleeId,
                                          LPWSABUF
lpCalleeData,
                                          GROUP FAR * g,
                                          DWORD
dwCallbackData);

/* WinSock 2.0 API new function prototypes */

SOCKET WSAAPI WSAAccept (SOCKET s,
                         struct sockaddr FAR *addr,
                         int FAR *addrlen,
                         LPCONDITIONPROC lpfnCondition,
                         DWORD dwCallbackData);

BOOL WSAAPI WSACloseEvent (WSAEVENT hEvent);

int WSAAPI WSAConnect (SOCKET s,
                       const struct sockaddr FAR *name,
                       int namelen,
                       LPWSABUF lpCallerData,
                       LPWSABUF lpCalleeData,
                       LPQOS lpSQOS,
                       LPQOS lpGQOS);
```

```
WSAEVENT WSAAPI WSACreateEvent (VOID);

SOCKET WSAAPI WSADuplicateSocket(SOCKET s,
                                 WSATASK hTargetTask);

int WSAAPI WSAEnumNetworkEvents (SOCKET s,
                                 WSAEVENT hEventObject,
                                 LPWSANETWORKEVENTS
lpNetworkEvents,
                                 LPINT lpiCount);

int WSAAPI WSAEnumProtocols (LPDWORD lpdwProtocols,
                             LPVOID lpProtocolBuffer,
                             LPDWORD lpdwBufferLength);

int WSAAPI WSAEventSelect (SOCKET s,
                           WSAEVENT hEventObject,
                           long lNetworkEvents);

BOOL WSAAPI WSAGetOverlappedResult (SOCKET s,
                                    LPWSAOVERLAPPED
lpOverlapped,
                                    LPDWORD lpcbTransfer,
                                    BOOL fWait,
                                    LPDWORD lpdwFlags);

BOOL WSAAPI WSAGetQOSByname (SOCKET s,
                             LPWSABUF lpQOSName,
                             LPQOS lpQOS);

u_long WSAAPI WSAHtonl (SOCKET s,
                        u_long hostlong);

u_short WSAAPI WSAHtons (SOCKET s,
                         u_short hostshort );

int WSAAPI WSAIoctl(SOCKET s,
```

```
                    DWORD dwIoControlCode,
                    LPVOID lpvInBuffer,
                    DWORD cbInBuffer,
                    LPVOID lpvOutBuffer,
                    DWORD cbOutBuffer,
                    LPDWORD lpcbBytesReturned,
                    LPWSAOVERLAPPED lpOverlapped,
                    LPWSAOVERLAPPED_COMPLETION_ROUTINE
lpCompletionRoutine);

SOCKET WSAAPI WSAJoinLeaf (SOCKET s,
                           const struct sockaddr FAR * name,
                           int namelen,
                           LPWSABUF lpCallerData,
                           LPWSABUF lpCalleeData,
                           LPQOS lpSQOS,
                           LPQOS lpGQOS,
                           int iFlags);

u_long WSAAPI WSANtohl (SOCKET s,
                        u_long netlong );

u_short WSAAPI WSANtohs (SOCKET s,
                         u_short netshort );

int WSAAPI WSARecv (SOCKET s,
                    LPWSABUF lpBuffesr,
                    DWORD dwBufferCount,
                    LPDWORD lpNumberOfBytesRecvd,
                    LPINT lpFlags,
                    LPWSAOVERLAPPED lpOverlapped,
                    LPWSAOVERLAPPED_COMPLETION_ROUTINE
lpCompletionRoutine);

int WSAAPI WSARecvDisconnect (SOCKET s,
                              LPWSABUF
lpInboundDisconnectData);
```

```
int WSAAPI WSARecvFrom (SOCKET s,
                        LPWSABUF lpBuffers,
                        DWORD dwBufferCount,
                        LPDWORD lpNumberOfBytesRecvd,
                        LPINT lpFlags,
                        LPVOID lpFrom,
                        LPINT lpFromlen,
                        LPWSAOVERLAPPED lpOverlapped,
                        LPWSAOVERLAPPED_COMPLETION_ROUTINE
lpCompletionRoutine);

BOOL WSAAPI WSAResetEvent(WSAEVENT hEvent);

int WSAAPI WSASend (SOCKET s,
                    LPWSABUFlpBuffers,
                    DWORD dwBufferCount,
                    LPDWORD lpNumberOfBytesSent,
                    int iFlags,
                    LPWSAOVERLAPPED lpOverlapped,
                    LPWSAOVERLAPPED_COMPLETION_ROUTINE
lpCompletionRoutine);

int WSAAPI WSASendDisconnect (SOCKET s,
                              LPWSABUF
lpOutboundDisconnectData );

int WSAAPI WSASendTo (SOCKET s,
                      LPWSABUFlpBuffers,
                      DWORD dwBufferCount,
                      LPDWORD lpNumberOfBytesSent,
                      int iFlags,
                      LPVOID lpTo,
                      int iTolen,
                      LPWSAOVERLAPPED lpOverlapped,
                      LPWSAOVERLAPPED_COMPLETION_ROUTINE
lpCompletionRoutine);

BOOL WSAAPI WSASetEvent(WSAEVENT hEvent);
```

```
SOCKET WSAAPI WSASocket (int af,
                         int type,
                         int protocol,
                         LPPROTOCOL_INFO lpProtocolInfo,
                         GROUP g,
                         int iFlags);

DWORD WSAAPI WSAWaitForMultipleEvents(DWORD cEvents,
                                      const WSAEVENT FAR *
lphEvents,
                                      BOOL fWaitAll,
                                      DWORD dwTimeout,
                                      BOOL fAlertable);

#ifdef __cplusplus
}
#endif

/* Microsoft Windows Extended data types */
typedef struct sockaddr SOCKADDR;
typedef struct sockaddr *PSOCKADDR;
typedef struct sockaddr FAR *LPSOCKADDR;

typedef struct sockaddr_in SOCKADDR_IN;
typedef struct sockaddr_in *PSOCKADDR_IN;
typedef struct sockaddr_in FAR *LPSOCKADDR_IN;

typedef struct linger LINGER;
typedef struct linger *PLINGER;
typedef struct linger FAR *LPLINGER;

typedef struct in_addr IN_ADDR;
typedef struct in_addr *PIN_ADDR;
typedef struct in_addr FAR *LPIN_ADDR;

typedef struct fd_set FD_SET;
typedef struct fd_set *PFD_SET;
```

```
typedef struct fd_set FAR *LPFD_SET;

typedef struct hostent HOSTENT;
typedef struct hostent *PHOSTENT;
typedef struct hostent FAR *LPHOSTENT;

typedef struct servent SERVENT;
typedef struct servent *PSERVENT;
typedef struct servent FAR *LPSERVENT;

typedef struct protoent PROTOENT;
typedef struct protoent *PPROTOENT;
typedef struct protoent FAR *LPPROTOENT;

typedef struct timeval TIMEVAL;
typedef struct timeval *PTIMEVAL;
typedef struct timeval FAR *LPTIMEVAL;

/*
 * Windows message parameter composition and decomposition
 * macros.
 *
 * WSAMAKEASYNCREPLY is intended for use by the Windows
Sockets implementation
 * when constructing the response to a WSAAsyncGetXByY()
routine.
 */
#define WSAMAKEASYNCREPLY(buflen,error)
MAKELONG(buflen,error)
/*
 * WSAMAKESELECTREPLY is intended for use by the Windows
Sockets implementation
 * when constructing the response to WSAAsyncSelect().
 */
#define WSAMAKESELECTREPLY(event,error)
MAKELONG(event,error)
/*
 * WSAGETASYNCBUFLEN is intended for use by the Windows
```

```
Sockets application
 * to extract the buffer length from the lParam in the
response
 * to a WSAAsyncGetXByY().
 */
#define WSAGETASYNCBUFLEN(lParam)           LOWORD(lParam)
/*
 * WSAGETASYNCERROR is intended for use by the Windows
Sockets application
 * to extract the error code from the lParam in the response
 * to a WSAGetXByY().
 */
#define WSAGETASYNCERROR(lParam)            HIWORD(lParam)
/*
 * WSAGETSELECTEVENT is intended for use by the Windows
Sockets application
 * to extract the event code from the lParam in the response
 * to a WSAAsyncSelect().
 */
#define WSAGETSELECTEVENT(lParam)           LOWORD(lParam)
/*
 * WSAGETSELECTERROR is intended for use by the Windows
Sockets application
 * to extract the error code from the lParam in the response
 * to a WSAAsyncSelect().
 */
#define WSAGETSELECTERROR(lParam)           HIWORD(lParam)

#endif  /* _WINSOCK2API_ */
```

附录 B.3 Winsock.def 文件

```
;
;         File: winsock.def
;       System: MS-Windows 3.x
;      Summary: Module definition file for Windows Sockets DLL.
;

LIBRARY         WINSOCK         ; Application's module name

DESCRIPTION     'BSD Socket API for Windows'

EXETYPE         WINDOWS         ; required for all windows
applications

STUB            'WINSTUB.EXE'   ; generates error message if
application
                                ; is run without Windows

;CODE can be FIXED in memory because of potential upcalls
CODE            PRELOAD         FIXED

;DATA must be SINGLE and at a FIXED location since this is a DLL
DATA            PRELOAD         FIXED           SINGLE

HEAPSIZE        1024
STACKSIZE       16384

; All functions that will be called by any Windows routine
; must be exported.  Any additional exports beyond those defined
; here must have ordinal numbers 1000 or above.

EXPORTS
    accept                      @1
        bind                        @2
        closesocket                 @3
        connect                     @4
```

```
    getpeername              @5
    getsockname              @6
    getsockopt               @7
    htonl                    @8
    htons                    @9
    inet_addr                @10
    inet_ntoa                @11
    ioctlsocket              @12
    listen                   @13
    ntohl                    @14
ntohs                        @15
    recv                     @16
    recvfrom                 @17
    select                   @18
    send                     @19
    sendto                   @20
setsockopt                   @21
    shutdown                 @22
    socket                   @23

    gethostbyaddr            @51
    gethostbyname            @52
    getprotobyname           @53
    getprotobynumber         @54
    getservbyname            @55
getservbyport                @56
    gethostname              @57

    WSAAsyncSelect           @101
    WSAAsyncGetHostByAddr    @102
    WSAAsyncGetHostByName    @103
WSAAsyncGetProtoByNumber     @104
    WSAAsyncGetProtoByName   @105
    WSAAsyncGetServByPort    @106
    WSAAsyncGetServByName    @107
    WSACancelAsyncRequest    @108
    WSASetBlockingHook       @109
    WSAUnhookBlockingHook    @110
```

```
        WSAGetLastError             @111
        WSASetLastError             @112
        WSACancelBlockingCall       @113
        WSAIsBlocking               @114
        WSAStartup                  @115
        WSACleanup                  @116

    __WSAFDIsSet                    @151

    WEP                             @500    RESIDENTNAME

;eof
```

附录 C 参考文献

(1) 周明天 汪文勇 编著，“TCP/IP 网络原理与技术”，清华大学出版社,1993 年
(2) 朱三元等编著，“网络通信软件设计指南”，清华大学出版社，1994 年
(3) Martin Hall 等，“Windows Sockets－An Open Interface for Network Programming under Microsoft Windows”，Document，1993 年
(4) Martin Hall 等，“Windows Sockets 2 Application Programming Interface”，Document，1995 年