电脑技术指导丛书

张金辉　崔文生　罗益民　尹立娟　编著

电脑网络编程指导

大连出版社

电脑网络编程指导

(Windows)

编著 张金辉 崔文生
罗益民 尹立娟
审校 林维成

大连出版社

电脑网络编程指导

张金辉　崔文生　罗益民　尹立娟　编著

大连出版社出版
（大连市西岗区长白街12号　　邮政编码　　116011）
大连理工大学印刷厂印刷　　新华书店发行

开本：850×1168毫米 1/32　　字数：250千字　印张：11½
印数：1－5000册
1999年6月第1版　　1999年6月第1次印刷

责任编辑：任雪芹　　责任校对：王恒田
封面设计：李克峻　　版式设计：蓝　雪

ISBN 7－80612－650－3/TP·8
定价：15.00元

第 一 章

TCP/IP 编程接口

TCP/IP (Transmission Control Protocol/Internet Protocol)是 70 年代中期美国国防部为其 ARPANET 广域网开发的网络体系结构和协议标准，以其为基础的 Internet 是目前国际上规模最大的计算机网络，现已有上万个网络连入 Internet，用户数以亿计。TCP/IP 虽不是国际标准，但已无可争议地成为“事实上的标准”，由于 TCP/IP 的地位和重要性，许多网络应用的开发是基于 TCP/IP 的，但作为 TCP/IP 核心的 TCP/UDP 和 IP 中下层协议，向外提供的只是原始的编程界面，而不是直接的用户服务，为此，开发了一些应用程序接口，socket(套接字)编程界面是常用的一种。本书主要介绍基于 socket 的网络程序设计。

1.1 Windows 套接字

我们以生活上浅显的例子来说明一下什么是 socket。将电话系统和面向连接的 socket 机制相比，有惊人相似的地方，以一个电话网为例，电话通信的双方相当于相互通信的两个进程；通话的双方所在的地区相当于一个网络区号，即网络地址；区内一个单位的交换机相当于一台主机，主机分配给每个用户的号码相当于 socket 号。任何用户在通信前，必须要有一台电话，相当于申请了一个 socket 号。同时要知道对方的电话号码，相当于对方有一个固定的 socket 号。然后向对方拨号呼叫，相当于发出一个连接请求(若对方不在同一个区，则要拨区号，相当于给出网络地址)。若对方空闲(相当于通信的另一台主机可接受新的

连接请求)，拿起话筒，双方就可通信了，相当于连接成功。双方通话的过程相当于向 socket 发出数据和从 socket 接受数据。通话结束，一方挂机，相当于关闭 socket 撤消连接。

在电话系统中，一般用户只能感受到本地话机和对方话机号码的存在，建立通话的过程、话音传输的过程及整个电话系统的技术细节对用户都是透明的，这与 socket 机制很相似。当利用 socket 进行通信时，对通信的细节不必关心。

从以上对 socket 的描述，我们看到 socket 实质是提供了进程通信的端点，进程通信之前，双方首先必须各自创建一个端点，否则无法建立联系并相互通信，正如打电话前双方必须有一台电话一样。

一个 socket 有三个参数与之相联系：协议、本地地址和本地端口，即一个半相关。每一个 socket 都有一个本地惟一的 socket 号，由操作系统统一分配。一个完整的 socket 连接需要有两个 socket 参与，可用一个相关描述：协议、本地地址、本地端口、远地地址和远地端口。

socket 编程界面由 4BSD UNIX 首先提出，目的是解决网络上进程通信问题。Windows Socket(WinSock)是基于 BSD 版本的套接字，但它针对 Windows 进行了扩展，它不仅包含了 Berkeley Socket 风格的库函数，也包含了一组针对 Windows 的扩展库函数，使得程序员能充分地利用 Windows 消息驱动机制进行编程。

在 Windows 套接字中，有三种形式的套接字：数据报套接字(Datagram Socket)、流式套接字(Stream Socket)和原始套接字(Raw Socket)。

数据报套接字提供了一种不可靠的、非连接的数据包通信方式。“不可靠”是指发送一个数据包不能获得担保到达目的地，也不能保证数据包按照发送的顺序到达。实际上，同一个数据包可能被数次发送。在同一计算机上或在负载较轻的局域网上两台

计算机上的两个进程通信时，使用数据报套接字发送数据包通常会到达目的地，而且常按发送的顺序收到到数据包，但其他情况下，很难保证数据包按发送顺序到达。所以，使用数据报套接字应具备处理这些情况的能力。

尤其上在非常复杂的网络如 Internet 上设计应用程序时，应当考虑到数据报套接字的不可靠性。若这个问题处理不当，应用程序可能会崩溃。尽管数据报服务有这些缺陷，但它在发送数据包或记录型数据时仍常用。另外，使用数据报可实现广播或组播服务。

流式套接字提供了一种可靠的面向连接的数据传输服务，它使用传输控制协议(TCP)实现了一种流式数据传输，能保证用户发送的数据包按发送的顺序无重复地到达目的地，所以，流式套接字适用于发送大批量的数据或对传输质量要求较高的数据。

原始套接字提供了使用底层网络协议的接口。

1.2 使用 Winsock API

不管 socket 的内部机制如何，它提供给应用程序的最终界面是一组系统调用，本节我们主要介绍一些重要的 Winsock 系统调用。Winsock API 中使用的大多数函数与 UNIX 中的 Berkeley Socket 实现是一样的。然而它们之间还是有差别的，这些差别主要在于为 Windows 环境进行异步事件处理而专门提供的附加函数。

Winsock API 与 UNIX socket 系统调用一个很大的差别是 Winsock 在使用前必须进行初始化，即在使用其他 Winsock API 前首先调用函数 WSAStartup( )。WSAStartup( )允许协商应用程序可以使用的 Winsock 版本。

1.2.1 初始化 Winsock——WSAStartup( )

任何使用 Winsock API 的应用程序必须首先调用 WSAStartup( )函数来检查 Windows Sockets 协议栈安装情况，即系统中是否有 Windows Sockets 的实现。WSAStartup( )函数说明如下：

WSAStartup( )——连接应用程序与 Winsock.DLL 的第一个调用。

格式：int PASCAL FAR WSAStartup (WORD wVersionReuested, LPWSADATA lpWSAData);

第一个参数是应用程序欲使用的 Windows Sockets 版本号，低位字节是主版本号，高位字节是副版本号。

第二个参数是一个指向 WSADATA 的指针。

其使用方法如下：

```
#include <winsock2.h>
        .
        .
        .
WSADATA wsaData;
if (WSAStartup(MAKEWORD(2,1),&wsaData) != 0)
{
  //Winsock 初始化错误
        fprintf(stderr,"WSAStartup failed: %d\n",GetLastError( ));
        ExitProcess(STATUS_FAILED);
  }
```

```
if (wsaData.wVersiong!=MAKEWORD(2,1)
{
    //Winsock 版本号不匹配
      fprintf(stderr,"WSAStartup failed: %d\n",GetLastError( ));
      WSACleanup( );
      ExitProcess(STATUS_FAILED);
}
//Winsock 初始化正确，继续…
```

若 Winsock.DLL 不存在或底层网络子系统没有被正确初始化，WSAStartup( )函数将返回 WSASYSNOTREADY 错误码。它还允许应用程序协商使用某个 Winsock 版本号，若这个版本号比任何当前系统 Winsock.DLL 支持的版本低，WSAStartup( )将返回错误码 WSAVERNOTSUPPORTED。若所要求的版本不低于当前系统 DLL 所支持的版本，wsaData 的 wVersiong 成员将返回应用程序应该使用的版本，另一个成员 wHighVersiong 将返回 DLL 所支持的最高版本。

应用程序若对当前的版本不接受，下一步应当调用 WSACleanup( )函数清除 Winsock 协议栈并退出应用程序。另外，wsaData 还返回了一些别的信息如 Winsock 开发商、一个进程可使用的最多套接字数等。

1.2.2 创建 Socket——socket( )

应用程序在使用 socket 通信前，必须拥有一个 socket，系统调用 socket( )可用来创建一个套接字。其调用格式如下：

```
SOCKET mysocketid=socket(af,type,protocol)
```

返回值 mysocketid 是一个无符号整数，即 socket 号，创建一个 socket 实际上是向操作系统申请一个 socket 号。

Socket( )有三个参数：

1) af——address family 地址族，指本 socket 所用的地址类型。AF_INET 标识 TCP/IP 地址族。

2) type——服务类型，是指创建 socket 的应用程序所希望使用的通信服务类型。在 Winsock2 中，socket 支持以下三种服务类型(TCP/IP 协议)：

SOCK_STREAM 流式套接字

SOCK_DGRAM 数据报套接字

SOCK_RAW 原始套接字

3) protocol——协议，指本 socket 请求的协议。在 Winscok2 中主要有如下几种协议：

IPPROTO_UDP 表示 UDP 协议，用于数据报套接字。

IPPROTO_TCP 表示 TCP 协议，用于流套接字。

IPPROTO_ICMP 表示 ICMP 协议，用于原始套接字。

Socket( )系统调用实际上仅指定了 socket 必须的三个属性之一的协议，本地地址和端口号需通过下面的系统调用来设置，另外，远地地址和远地端口也要通过下面的调用设置，一旦设置完这些参数，一个完整的 socket 连接即可建立起来。

当 socket( )调用成功后，将返回新建的 socket 号，若不成功则返回 INVALID_SOCKET，并可通过 WSAGetLastError( )确定错误原因。在 Winsock 中，常用的 socket( )调用方式为如下形式：

```
//创建一个数据报套接字
SOCKET sockid =socket (AF_INET, SOCK_DGRAM,
IPPROTO_UDP);
```

或

```
//创建一个流套接字
SOCKET sockid =socket (AF_INET, SOCK_STREAM,
IPPROTO_TCP);
```

或

```
//创建一个原始套接字
SOCKET sockid =socket (AF_INET, SOCK_RAW,
IPPROTO_ICMP);
```

1.2.3 指定本地地址——bind( )

调用 socket( )是 socket 通信的第一步，目的是创建一个通信所用的 socket 端口。Socket( )系统调用创建套接字时仅指定了所希望的服务类型，下一步还须指明哪一台计算机、哪一个进程需要通信，这些工作是由 bind( )系统调用来完成的。

系统调用 bind( )将本机地址及标识通信进程的端口号(port)与所创建的套接字联系起来，即将本地 socket 地址赋予套接字。其调用格式为：

```
bind(sockid,localaddress,addresslength)
```

其中：

1) socketid 是由系统调用 socket( )返回的 socket 号。

2) 第二个参数 localaddress 是本地地址，由它指定本机地址和端口号。其定义为：

```
struct sockaddr_in{
    short sin_family;
    u_short sin_port;
```

```
    struct in_addr sin_addr;
    char sin_zero[8];
  }
```

对于 TCP/IP 来说，sin_family 成员将是 AF_INET；成员 sin_port 指定的是将要分配给套接字的端口号；成员 sin_addr 给出的是套接字的本机地址，即本机的 IP 地址。

3) 第三个参数 addresslength 是地址(localaddress)长度，指出以字节为单位的地址结构(sockaddr_in)的长度，常用的取值是：

```
sizeof(localaddress)
```

其中，localaddress 为指向 socket 地址的指针。下面举例说明其用法：

```
SOCKET aSock;//套接字 ID
Sockaddr_in addr;//套接字地址
Int nSockErr;//错误码
  .
  .
  .
//创建套接字
aSock=socket(AF_INET,SOCK_STREAM,0);
//指定端口号和本机地址
addr.sin_family=AF_INET;
addr.sin_port=htons(5000);
addr.sin_addr.s_addr=htonl(INADDR_ANY);
```

```
        //将本地地址和套接字联系起来
        if (bind (aSock, (LPSOCKADDR) &addr, sizeof (addr) )
==SOCKET_ERROR)
            {
                //调用失败
                nSockErr=WSAGetLastError( );
                //错误处理…
            }
        //调用成功,继续…
```

1.2.4 建立 socket 连接——请求连接 connect( )与接受连接 accept( )系统调用

这两个系统调用用于将本地套接字和远地套接字连接起来。其中 connect( )用于建立连接。connect( )的调用格式如下：

```
connect(sockid,destaddress,addresslength)
```

其中：

1) sockid 是本地 socket 号。

2) destaddress 是一个指向对方 socket 地址结构的指针。

3) addresslength 是对方 socket 地址的长度。

若连接失败，则返回 SOCKET_ERROR，程序可调用 WSAGetLastError( )，获知失败的具体原因。

connect( )调用主要是为面向连接的客户设计的，客户调用此服务指定远地主机地址和远地端口。无连接通信也可调用 connect( )，但这时在本地系统和远地系统之间不进行实际的报文

交换，调用将从本地操作系统直接返回，此时并没有建立真正的 socket 连接。无连接服务调用此函数的目的在于通知操作系统，来自指定地址的数据应送往本 socket。

accept( )系统调用是专为面向连接的服务器设计的，当客户调用 connect( )时服务器调用 accept( )同意连接，这样，一个真正的连接就建立起来了，就象通话的双方都已提起了话筒。accept( )的调用格式如下：

```
Newsock=accept(sockid,clientaddress,addresslength)
```

其中：

1) sockid 是本地 socket 号。

2) clientaddress 是一个指向客户 socket 地址结构的指针，其值将由操作系统赋予。

3) addresslength 是客户 socket 地址的实际长度。服务器用 addresslength 和 clientaddress 获知客户的地址信息。

4) newsock 是一个新的 socket 号。accept( )除将连接请求方即客户的 socket 地址与长度放入上面两个变量外，还返回一个新的 socket 号。newsock 可用于服务器处理并发请求，对应于一个从服务器，从服务器利用此新的 socket 回答 accept( )所接受的客户请求。

若 accept( )调用失败，则返回 INVALID_SOCKET。

1.2.5 侦听连接——listen( )系统调用

listen( )是用于面向连接服务器的，它表示愿意接受连接。listen( )在 accept( )之前调用，其调用格式为：

```
listen(sockid,queuelen)
```

其中：

1) sockid 是本地 socket 号，服务器从它上面接听客户的请求。

2) queuelen 是请求队列的长度，listen( )以此参数限制排队请求连接的客户。

1.2.6 发送数据——send( )和 sendto( )

上面讨论的函数是用于建立连接的，成功地建立连接后就可发送数据了。用于 socket 发送数据的系统调用共有两个，send( )用于面向连接的传输，它可以不指定信宿地址；sendto( )用于无连接传输，当调用 sendto( )时必须指定信宿地址。若无连接 socket 的双方都已调用了 connect( )，可以认为是建立了面向连接的 socket，此时可使用 send( )函数发送数据。

send( )的调用格式为：

```
send(sockid,buffer,bufferlen,flags)
```

其中：

1) sockid 是本地 socket 号。

2) buffer 是一个指向发送缓冲区的指针。

3) bufferlen 是发送缓冲区的大小。

4) flags 是传输控制标志，其可取下列两个值之一：

MSG_DONTROUTE：寻径控制选项，在信宿地址的网络号部分指定数据发送须经过的网络接口，使其不经过本地寻径机制而直接将数据发送出去，此选项常用于网络调试软件。

MSG_OOB：带外数据发送

sendto( )的调用的格式为：

sendto(sockid,buffer,bufferlen,flags,destaddress,addresslen)

其中：sockid,buffer,bufferlen,flags 参数的意义同上，其他参数的意义为：

1) destaddress 是指向信宿 socket 地址的指针。

2) addresslen 是信宿 socket 地址的长度。

1.2.7 接收数据——recv( )和 recvfrom( )

recv( )系统调用是用于面向连接传输的，当用 recv( )发送数据是不必每次都指定对方的地址，其调用格式为：

recv(sockid,buffer,bufferlen,flags)

其中：参数 sockid，buffer 和 bufferlen 的含义同 send( )函数，flags 与 send( )的 flags 参数略有不同，其可选值为：

MSG_OOB ：带外数据接受。

MSG_PEEK：允许调用者从 socket 中读出下一输入数据块的拷贝，而不将它从 socket 中删除，以供提前查阅。一般情况下，如果从 socket 中读取一个数据区，调用返回后，socket 将不在保存该数据块。

另外，bufferlen 是一个指针，其所指单元的初值为欲读数据的长度，调用后的值是实际读出的值。

recvfrom( )系统调用用于无连接的传输，其调用格式为：

recvfrom(sockid,buffer,bufferlen,flags,srcaddress,addrlen)

其中：sockid,buffer,bufferlen,flags 的含义如 recv( )相同，不同之处在于其参数中有一个指向信源地址的指针 srcaddress 和地

址长度 addrlen，信源地址的初始值为空。当 recvfrom( )返回时，信源地址将被填充。

1.2.8 关闭套接字——closesocket( )

当使用完套接字就可调用 closesocket( )关闭套接字，以释放与套接字有关的系统资源。closesocket( )的调用格式为：

```
int closesocket(socketid);
```

其中：socketid 为系统调用 socket( )返回的套接字。

此外，也可以使用系统调用 shutdown( )对于一个给定的套接字不能发送数据、不能接收数据或既不能发送数据也不能接收数据。通常只有一个套接字没有用时才调用此函数，因为套接字不能被重用，但调用完 shutdown 后，还必须调用 closesocket( )来真正释放和套接字有关的系统资源。shutdown( ) 的调用格式为：

```
int shutdown(socketid,how);
```

其中：

1) socketid 是套接字。

2) how 是一个整数，表示使用方式，其可取下列值之一：SD_RECEIVE，SD_SEND 或 SD_BOTH。若 how 取 SD_RECEIVE，则调用 shutdown(socketid, SD_RECEIVE)后，应用程序将不能从套接字 socketid 读入数据，即接收函数将被禁止。若 how 取值 SD_SEND, 则调用 shutdown (socketid, SD_SEND) 后，应用程序将不能使用套接字 socketid 发送数据，即发送函数将被禁止。若 how 取值 SD_BOTH, 则调用 shutdown (socketid,

SD_BOTH)后，应用程序将不能使用套接字 socketid 发送数据，也不能从套接字中读入数据，即发送函数和接收函数都将被禁止。

1.2.9 异步处理函数

在缺省状态下，当套接字由 socket( )函数创建后，它被设置为对 I/O 操作处于阻塞的状态，即 connect( )，recv( )，recvfrom( )等调用将会暂停一个进程或线程的执行，直到它接收到一个数据报，但有时希望程序在连接或等待数据时能继续做一些其他的事情，如取消连接、退出程序等，要做到这些可使用 Winsock 提供的异步接收数据的方法：一种是使用是 BSD 接口的函数 select( )，另外一种是使用 Winsock 的专用 WSAAsncSelect( )调用。

1. select( )

当使用 select( )时，必须先调用 ioctlsocket( )。函数 ioctlsocket( )可将一个套接字设置为非阻塞模式，其调用格式为：

```
ioctlsocket(socketid,FIONBIO,1)
```

其中：socketid 是套接字 ID。一旦一个套接字变为非套接字模式，任何不能立即完成的调用将返回一个错误，调用 WSAGetLastError( )将返回 WSAEWOULDDBLOCK。

select( )函数允许应用程序在等待一个事件发生时检测一组套接字的状态，可使用 select( )函数指定想等待数据的套接字。无论什么时候，当数据被套接字接收到以后，select( )将返回，并确定在输入队列中哪个套接字在等待数据，然后，就可取数据了。

select( )的调用格式为：