Visual C++ 黑客编程揭秘与防范

梁洋洋 编著

5大编程案例：

网络扫描器编程、远程控制软件开发、基于认证的网络程序、U盘防火墙工具实现、Windows底层RootKit编程

- 20多个黑客编程关键技术：Socket、监听、绑定、后门、扫描、线程、注入、拒绝服务、杀毒工具、远程控制等
- 从Socket、API基础到案例，全实例呈现黑客VC编程技术

人民邮电出版社
POSTS & TELECOM PRESS

Visual C++

黑客编程揭秘与防范

梁洋洋 编著

人民邮电出版社
北京

图书在版编目（CIP）数据

Visual C++黑客编程揭秘与防范 / 梁洋洋编著. —北京：人民邮电出版社，2009.7
ISBN 978-7-115-20664-0

Ⅰ. V… Ⅱ. 梁… Ⅲ. ①C语言－程序设计②计算机网络－安全技术 Ⅳ. TP312 TP393.08

中国版本图书馆CIP数据核字（2009）第050614号

内 容 提 要

本书全面介绍了在Visual C++环境中，使用Windows Socket及API开发各类防范黑客软件及安全防护工具的编程实现方法，深入剖析了目前热门的黑客编程技术。本书通过基础篇、提高篇、综合篇和拓展篇这种循序渐进地方式，向读者介绍了防范黑客攻击程序、安全防护工具、远程控制软件和网络安全管理软件的原理及具体编程实现方法。

本书内容丰富，实用性和实战性强，不仅包括读者必备的防范黑客的编程知识，更深入阐述了网络编程方面的高级技术。不仅适用于黑客程序开发，在读者掌握了本书介绍的各种编程技术后，还能用于开发各类网络安全防护软件。

本书适合初、中级网络安全爱好者学习网络安全知识时使用，同时也可作为程序员和网络高级安全工程师的参考用书。

Visual C++ 黑客编程揭秘与防范

◆ 编　　著　梁洋洋
责任编辑　屈艳莲
执行编辑　张　涛

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街14号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京鸿佳印刷厂印刷

◆ 开本：787×1092　1/16
印张：25
字数：599千字　　2009年7月第1版
印数：1－3 500册　　2009年7月北京第1次印刷

ISBN 978-7-115-20664-0/TP

定价：49.00元

读者服务热线：(010)67132692　印装质量热线：(010)67129223
反盗版热线：(010)67171154

前言

能编写出属于自己的防范黑客软件一直是很多网络安全爱好者梦寐以求的。然而由于各种编程技术的门槛较高，加之很少能找到相关的学习资料和具体技术细节，很多安全爱好者只能叹息黑客编程技术遥不可及。笔者在初学网络编程时也有同样的困惑，因此，非常能理解很多读者一直不得入门的心情。

为了让更多的网络安全爱好者能够迅速掌握防范黑客软件和安全工具的开发技术，也为了提高国内网络安全技术的整体水平，笔者精心编写了本书。

本书是根据自己多年的学习和工作经验，结合当前网络安全技术最新的发展趋势，循序渐进地为读者讲解了，如何在 Visual C++（简称 VC）环境下开发各种防范黑客工具和安全软件。本书旨在技术上为读者提供一种学习的方法和参考，以便在实践中逐步提高防范黑客的技能。

本书的特点

本书通过 4 篇（基础篇、提高篇、综合篇、拓展篇）内容，循序渐进地为读者讲解了当前流行的网络安全技术，以及黑客软件和安全工具的实现原理及编程实现方法。本书涵盖的内容丰富，从 Windows Socket 及 API 编程基础到最基本的网络扫描器编程，从基本防范黑客攻击程序到基于认证的网络程序破解，从下载者程序的编程实现到 U 盘防火墙等安全工具实现，从 Windows 底层的 RootKit 编程到远程控制软件开发，从网络准入控制技术到网络蜘蛛编程等。本书都逐个讲解各类技术的实现原理，并通过代码编程实现，其中很多代码具有很高的实用价值。

本书的特点主要体现在以下几个方面。

- 本书的编排采用循序渐进的方式，适合于对 VC 程序开发有一定了解，并对黑客程序开发抱有兴趣的网络安全爱好者。
- 本书在介绍大量网络安全技术实现原理时，均提供了典型的案例和参考图例。读者通过对原理的学习，能够掌握用 VC 开发黑客工具的具体技术，同时，也能更加深入地理解网络安全技术的具体细节。
- 本书除了介绍主流的安全技术及编程方法外，还涉及到 RootKit、SSDT 恢复等系统底层编程技术，对于希望提高黑客软件开发技术的读者无疑是一个很大的帮助。
- 本书虽然以剖析黑客软件开发为基本出发点，但是并不限于黑客技术，更多的是从技术角度探讨技术原理及实现方法，同时将网络安全思想时刻灌输其中。如书中涉及的 U 盘防火墙、网络准入技术等都是对当前互联网黑客攻击泛滥的思考和防范方法的具体实现。

本书的内容安排

本书分为 4 篇，共 15 章，以网络编程最基本的 Windows Socket 及 API 开始，逐步介绍简单的网络扫描器编程技术，让读者轻松入门。通过常见黑客工具及下载者防范程序的编写，让读者对编程技术有一个更大地提高。在读者掌握了一定黑客软件开发基础后，开始介绍 RootKit 编程技术及远程控制编程技术，让读者通过一个完整综合的实例来学习用 VC 开发防范黑客软件。最后结合笔者的工作经验，介绍了网络准入技术和网络蜘蛛等拓展技术，供有兴趣的读者深入学习。

第一篇（第 1 章～第 3 章）基础篇。

讲述了使用 VC 开发防范黑客软件，尤其是开发基于网络的防范黑客软件必须具备的理论基础及入门级的编程知识。通过本章学习，读者可以掌握 Windows Socket API 编程开发的技术、网络扫描程序及认证程序破解的编程实现，从而为进一步提高编程水平打下基础。

第二篇（第 4 章～第 7 章）提高篇。

讲述了拒绝服务攻击技术的原理，感染型下载者程序的功能及原理，RootKit 技术的编程实现。通过本篇的学习，读者的防范黑客编程技术将得到很大提高。本篇介绍的 3 类典型程序是当前互联网最为流行的防范黑客攻击技术。同时针对下载者程序，还讲解了如何有针对地防范，并通过 U 盘病毒防火墙的形式予以实现。

第三篇（第 8 章～第 14 章）综合篇。

本篇通过一个完整的远程控制软件的功能、原理、设计、实现及优化等方面的讲解，为读者深入剖析了一个完整软件的开发流程。本篇是前几章编程技术的综合，是各种技术的实战运用。在本篇详细地为读者介绍了编程中的各个细节，同时首次公开了部分远程控制软件的关键代码。

第四篇（第 15 章）拓展篇。

结合笔者工作和学习的经验，为读者介绍了网络准入技术、网络蜘蛛、SSDT 恢复等技术的原理及实现方式。对于希望进一步提高自己安全软件开发技术的读者无疑是一个知识延展的机会，为读者将来开发属于自己的网络安全工具或软件提供了必要的铺垫作用。

本书由浅入深，由理论到实践，尤其适合对 VC 开发有一定了解，同时对黑客软件开发抱有极大兴趣的读者学习。

适合阅读本书的读者

本书适合初、中级网络安全爱好者学习网络安全知识时使用，同时也可作为程序员和网络高级安全工程师的参考用书。

编者

目录

第一篇 基础篇

第二篇 提高篇

第三篇 综合篇

第四篇 拓展篇

第一篇

基础篇

第1章 开发网络安全程序基础

Visual C++对于其他的 Windows 编程工具而言，不但提供了可视化的编程方法，而且提供了强大的 MFC（微软基础类库），使得开发者可以使用完全面向对象的方法来开发应用程序。事实上，MFC 提供的类库和控件都是构架在 Win32 API 函数基础之上的，是封装了 API 函数的集合。它们把常用的 API 函数结合在一起，构成一个控件或类库，给出方便的使用方法和属性，从而加速了 Windows 应用程序开发的过程。

1.1 认识 Windows API 和 Socket

稍微有过 Windows API 编程经验的人员都知道，强大的 Windows 网络程序都是通过灵活的 API 编程实现的。这些 Win32 API 函数是网络编程中最基础的函数，在任何网络程序中都会用到。下面各小节中，将介绍这些基础的 Win32 API 函数，掌握此类函数，对后面的网络编程很有帮助。

1.1.1 Windows API 编程的优点

大多数 Windows 程序员已经熟悉 MFC 的编程开发。实际上，如果要开发出更灵活、更实用、更有效率的应用程序，尤其是网络程序，必然要直接使用 API 函数进行编程。与使用 MFC 编写出来的程序相比，使用 Win32 API 编写出来的程序有很多优势：

- 生成的可执行程序体积小；
- 执行效率高；
- 更适用于编写直接对系统进行底层操作的程序，其所生成的代码质量也更加高效简洁。

目前，大多数的黑客程序都依赖于网络，因此，开发黑客程序必然离不开网络通信，即在两台计算机间进行通信。在网络编程中应用最广泛的是 Winsock 编程接口，即 Windows Socket API。所有在 Win32 平台上的 Winsock 编程都要经过下列步骤：定义变量→获得 Winsock 版本→加载 Winsock 库→初始化→创建套接字→设置套接字选项→关闭套接字→卸载 Winsock 库→释放所有资源，如图 1.1 所示。

为了方便网络编程，20 世纪 90 年代初，由微软公司联合了其他几家公司共同制定了一套 Windows 下的网络编程接口，即 Windows Sockets 规范。它不是一种网络协议，而是一套开放的、支持多种协议的 Windows 下的网络编程接口。现在的 Winsock 已经基本上实现了与协议无关，可以使用 Winsock 来调用多种协议的功能，但较常使用的是 TCP/IP。Socket 实际

是提供了一个通信端口，可以通过这个端口与任何一个具有 Socket 接口的计算机通信。应用程序在网络上传输，接收的信息都通过这个 Socket 接口来实现。

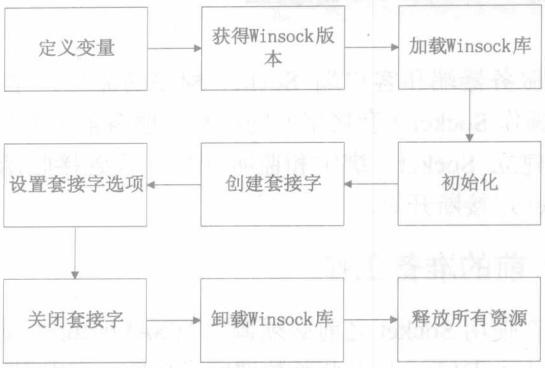

图 1.1 WinSock 编程步骤

注意：程序最后不释放资源是学习网络通信编程新手常犯的错误。如果只是一款简单的程序来释放资源，看不出有什么影响；如果是多线程或者创建了多个 Socket 对象而不去释放这些资源，那么就会对系统造成较大的影响。所以，程序最后务必要释放所有资源。

1.1.2 Socket 通信流程

传统的网络通信模式是 C/S 模式，即客户端、服务器端模式。客户端、服务器端模式中至少需要一台服务端、一台客户端。服务端先启动，建立一个 Socket，并对相应的 IP 和端口进行绑定、监听；客户端后启动，启动后也建立一个 Socket，直接链接服务端监听的端口，双方建立连接后，服务端和客户端就可以互相传输数据了。

整个程序架构可以分为两大部分，服务器端和客户端，如图 1.2 所示。

服务器端和客户端启动后，各自执行自己的任务。服务器端 Socket 程序流程：socket()→bind()→listen→accept()→recv()/send()→closesocket()，如图 1.3 所示。

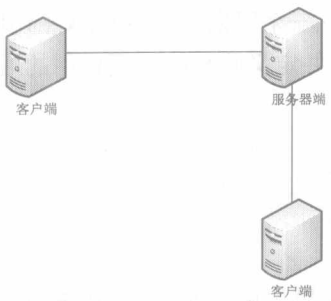

图 1.2 服务器端与客户端通信架构

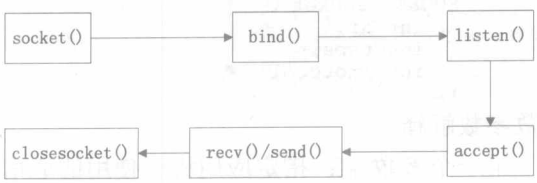

图 1.3 服务器端 Socket 程序流程

客户端 Socket 程序流程：socket()→connect()→send()/recv()→closesocket()，如图 1.4 所示。

图 1.4 客户端 Socket 程序流程

1.2 服务器端 Socket 编程

上一节简单介绍了服务器端和客户端 Socket 程序的流程，本节将详细阐述 Windows Socket 编程中服务器端操作 Socket（套接字）的编程。服务器端编程主要涉及初始化阶段调用 WSAStartup 函数、建立 Socket、绑定和监听端口、设置接收函数、异步过程处理函数 WSAAsyncSelect、Socket 连接断开等。

1.2.1 使用 Socket 前的准备工作

使用 Socket 的程序在使用 Socket 之前必须调用 WSAStartup 函数，此函数在应用程序中用来初始化 Windows Sockets DLL，只有此函数调用成功后，应用程序才可以再调用 Windows Sockets DLL 中的其他 API 函数，否则后面的任何函数都将调用失败。

WSAStartup 函数的原型：

```
int WSAStartup(
     WORD wVersionRequested,
     LPWSADATA lpWSAData
);
```

函数参数解释：

- 第一个参数 wVersionRequested：应用程序欲使用的 Windows Socket 版本号。
- 第二个参数 lpWSAData：指向 WSADATA 的指针。

函数调用后返回一个 int 型的值，通过检查这个值来确定初始化是否成功。该函数执行成功后返回 0。在程序中调用该函数的形式如下：

```
WSAStartup(MAKEWORD(2,2), (LPWSADATA) &WSAData)
```

其中 MAKEWORD(2,2)表示程序使用的是 WinSocket 2 版本，WSAData 用来存储系统传回的关于 WinSocket 的结构。

1.2.2 建立 Socket

初始化 WinSock 的动态连接库后，需要在服务器端建立一个用来监听的 Socket 句柄，为此可以调用 Socket()函数，用来建立这个监听的 Socket 句柄，并定义此 Socket 所使用的通信协议。

Socket 的原型：

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

函数参数解释。

- 第一个参数 af：指定应用程序使用的通信协议的协议族。对于 TCP/IP 族，该参数设置为 FF_INET。
- 第二个参数 type：指定要创建的套接字类型。
- 第三个参数 protocol：指定应用程序所使用的通信协议。

在 Winsock 2 中，type 支持以下 3 种类型。

- SOCK_STREAM：流式套接字。

- SOCK_DGRAM：数据报套接字。
- SOCK_RAW：原始套接字。

在 Winsock 2 中，protocol 支持以下 3 种类型。

- IPPROTO_UDP：UDP 协议，用于无连接数据报套接字。
- IPPROTO_TCP：TCP 协议，用于流套接字。
- IPPROTO_ICMP：ICMP 协议，用于原始套接字。

该函数调用成功则返回 Socket 对象，函数调用失败则返回 INVALID_SOCKET（调用 WSAGetLastError()可得知函数调用失败的原因，所有 WinSocket 的函数都可以使用这个函数来获取失败的原因）。

在 Windows 程序中，并不是用内存的物理地址来标志内存块、文件、任务和动态装入的模块；相反的，Windows API 给这些项目分配确定的句柄，并将句柄返回给应用程序，然后通过句柄来进行操作。

提示：句柄是 Windows 用来标志被应用程序所建立或使用的对象的惟一整数，Windows 使用各种各样的句柄标志，诸如应用程序实例、窗口、控件、位图、GDI 对象等。一个 Windows 应用程序可以用不同的方法获得一个特定项的句柄。通常，Windows 通过应用程序的引出函数将一个句柄作为参数传给应用程序，应用程序一旦获得了一个确定的句柄，便可以在 Windows 环境下的任何地方对这个句柄进行操作。

1.2.3 绑定端口

当创建了一个 Socket 以后，套接字数据结构中有一个默认的 IP 地址和默认的端口号。一个服务器端程序必须调用 bind 函数来为其绑定一个 IP 地址和一个特定的端口号，这样客户端才知道连接服务器端哪一个 IP 地址的哪个端口。客户端程序一般不必调用 bind 函数来为其 Socket 绑定 IP 地址和端口号。该函数调用成功返回 0，否则返回 SOCKET_ERROR。

bind 函数原型：

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

函数参数解释。

- 第一个参数 s：指定待绑定的 Socket 描述符。
- 第二个参数 name：指定一个 sockaddr 结构。
- 第三个参数 namelen：是 name 结构体的大小。

这里需要简单介绍一下第二个参数 name，这个参数是一个 sockaddr 结构类型，sockaddr 结构类型定义如下。参数 sa_family 指定地址族，对于 TCP/IP 族的套接字，将其设置为 AF_INET。

```
struct sockaddr(
     u_short sa_family;
     char sa_data[14];
);
```

对于 TCP/IP 族的套接字进行绑定时，通常使用另一个地址结构：

```
struct sockaddr_in(
     short sin_famil;
     u_short sin_port;
     struct in_addr sin_addr;
     char sin_zero[8];
);
```

函数参数解释。

- 第一个参数 sin_famil：设置 TCP/IP 协议族类型 AF_INET。
- 第二个参数 sin_port：指明端口号。
- 第三个参数 sin_addr：结构体中只有一个惟一的字段 s_addr，表示 IP 地址，该字段是一个整数，一般用函数 inet_addr()把字符串形式的 IP，转换成 unsigned long 型的整数值后再发送给 s_addr。

提示：有的服务器是多宿主机，一般有两个网卡。运行在这样的服务器上的服务端程序在为其 Socket 绑定 IP 地址时，可以把 htonl(INADDR_ANY) 置给 s_addr。这样做的好处是，不论哪个网段上的客户程序都能与该服务端程序通信。如果只给运行在多宿主机上的服务端程序的 Socket 绑定一个固定的 IP 地址，那么只有与该 IP 地址处于同一个网段上的客户程序才能与该服务端程序通信。使用 0 来填充 sin_zero 数组，目的是让 sockaddr_in 结构的大小与 sockaddr 结构的大小一致。

下面是一个 bind 函数调用的例子，如代码 1.1 所示。

代码 1.1 bind 函数调用示例

```
//…
    struct sockaddr_in name;
    name.sin_family = AF_INET;
    name.sin_port = htons(80);
    name.sin_addr.s_addr = htonl(INADDR_ANY);
    int namelen = sizeof(name);
    bind(sSocket, (struct sockaddr *)&name, namelen);
//…
```

以上代码中，如果不需要特别指明 IP 地址和端口号的值，那么，可以设定 IP 地址为 INADDR_ANY，Port 为 0。Windows Sockets 会自动为其设定适当的 IP 地址及 Port（1024～5000 之间的值）。如果想得到该 IP 地址和端口，可以调用 getsockname()函数来获知其被设定的值。

1.2.4 监听端口

服务器端的 Socket 对象绑定完成后，服务器端必须建立一个监听的队列，来接收客户端发来的连接请求。listen()函数使服务器端的 Socket 进入监听状态，并设定可以建立的最大连接数（一次同时连接不能超过 5 个 IP）。

listen()函数原型：

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

函数参数解释。

- 第一个参数 s：指定监听的 Socket 描述符。
- 第二个参数 backlog：为一次同时连接的最大数目。

该函数调用成功返回 0，否则返回 SOCKET_ERROR。服务器端的 Socket 调用完 listen()后使其套接字 s 处于监听状态，处于监听状态的流套接字 s 将维护一个客户连接请求队列，最多容纳 backlog 个客户连接请求。

1.2.5 创建服务器端接受客户端请求

当客户端发出连接请求时，服务器端 hwnd 视窗会收到 Winsock Stack 送来自定义的一个消息，为了使服务器端接受客户端的连接请求，就要使用 accept()函数。处于监听状态的流套

接字 s 从客户端连接请求队列中取出排在最前面的一个客户请求，并且创建一个新的套接字来与客户端套接字共同创建连接通道。原先处于监听状态的套接字继续处于监听状态，等待客户端的连接，这样服务器端和客户端才算正式完成通信程序的连接动作。如果创建连接通道成功，就返回新创建套接字的描述符，以后与客户端套接字交换数据的是新创建的套接字；如果失败就返回 INVALID_SOCKET。

accept 函数原型：

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

函数参数解释。

- 第一个参数 s：处于监听状态的流套接字。
- 第二个参数 addr：用来返回新创建的套接字的地址结构。
- 第三个参数 addrlen：指明用来返回新创建的套接字的地址结构的长度。

下面是 accept 函数示例，如代码 1.2 所示。

代码 1.2 accept 函数示例

```
//…
    struct sockaddr_in addr;
    int addrlen;
    addrlen = sizeof(addr);
    addr = accept(sSocket, (struct sockaddr *)&addr, &addrlen);
//…
```

1.2.6 服务器端响应客户端连接请求

当客户端向服务器端发出连接请求，服务器端即可用 accept 函数实现与客户端建立连接。为了达到服务器端的 Socket 在恰当的时候与从客户端发来的连接请求建立连接，服务器端需要使用 WSAAsyncSelect 函数，让系统主动来通知服务器端程序有客户端提出连接请求了。该函数调用成功返回 0，否则返回 SOCKET_ERROR。

WSAAsyncSelect 函数原型：

```
int WSAAsynSelect(
     SOCKET s,
     HWND hWnd,
     unsigned int wMsg.
     long lEvent
);
```

函数参数解释。

- 第一个参数 s：Socket 对象。
- 第二个参数 hWnd：接收消息的窗口句柄。
- 第三个参数 wMsg：传给窗口的消息。
- 第四个参数 lEvent：被注册的网络事件，即是应用程序向窗口发送消息的网络事件。

lEvent 值为下列值 FD_READ、FD_WRITE、FD_OOB、FD_ACCEPT、FD_CONNECT、FD_CLOSE 的组合。各个值的具体含意如下所述。

- FD_READ：希望在套接字 S 收到数据时收到消息。
- FD_WRITE：希望在套接字 S 发送数据时收到消息。
- FD_ACCEPT：希望在套接字 S 收到连接请求时收到消息。

- FD_CONNECT：希望在套接字 S 连接成功时收到消息。
- FD_CLOSE：希望在套接字 S 连接关闭时收到消息。
- FD_OOB：希望在套接字 S 收到带外数据时收到消息。

该函数在具体应用时，wMsg 应是在应用程序中定义的消息名称，而消息结构中的 lParam 则为以上各种网络事件名称。所以，可以在窗口处理自定义消息函数中使用以下结构代码来响应 Socket 的不同事件，如代码 1.3 所示。

代码 1.3 相应 Socket 事件的结构代码

```
switch(lParam)
{
// 响应 FD_READ 的函数方法实现
case FD_READ:
    ...
    break;
// 响应 FD_WRITE 的函数方法实现
case FD_WRITE、
    ...
    break;
// 响应 FD_ACCEPT：E 的函数方法实现
case FD_ACCEPT：、
    ...
    break;
//...
}
```

1.2.7 完成服务端与客户端 Socket 连接

结束服务器端与客户端的通信连接，可以由服务器端或客户端的任一端发出请求，只要调用 closesocket()就可以了。同样的，要关闭服务器端监听状态的 Socket，也是利用该函数。在调用 closesocket()函数关闭 Socket 之前，与程序启动时调用 WSAStartup()函数相对应。程序结束前，需要调用 WSACleanup()来通知 Winsock Stack 释放 Socket 所占用的资源。该函数调用成功返回 0，否则返回 SOCKET_ERROR。

WSAStartup()函数原型：

```
int WSACleanup();
```

该函数在应用程序完成对请求的 Socket 库的使用后调用，来解除与 Socket 库的绑定并且释放 Socket 库所占用的资源。该函数一般用在网络程序结束的地方调用。

closesocket()原型：

```
int closesocket(
    SOCKET s
);
```

函数参数解释。

- 第一个参数 s：表示要关闭的套接字。

该函数如果成功执行就返回 0，否则返回 SOCKET_ERROR。

每个进程中都有一个套接字描述符表，表中的每个套接字描述符都对应了一个位于操作系统缓冲区中的套接字数据结构。因此，可能有几个套接字描述符指向同一个套接字数据结构。套接字数据结构中专门有一个字段存放该结构被引用的次数，即有多少个套接字描述符指向该结构。当调用 closesocket 函数时，操作系统先检查套接字数据结构中的该字段的值。如果值为 1，就表明只有一个套接字描述符指向它，因此操作系统就先把 s 在套接字描述符表中对应的那条表项清除，并且释放 s 对应的套接字数据结构；如果该字段值大于 1，那么操作系统仅仅清除 s 在套接