Visual C++ 网络程序设计

实例详解

张 越 等编著

人民邮电出版社
POSTS & TELECOM PRESS

图书在版编目（CIP）数据

Visual C++网络程序设计实例详解 / 张越等编著.
—北京：人民邮电出版社，2006.7

ISBN 7-115-14914-3

Ⅰ. V... Ⅱ. 张... Ⅲ. C语言—程序设计 Ⅳ. TP312

中国版本图书馆CIP数据核字（2006）第067527号

内 容 提 要

本书全面介绍了使用Visual C++开发各种网络程序的方法与技巧，内容涵盖TCP和UDP的客户/服务器编程、Internet和LAN上的远程PC控制、链路层的计算机扫描技术、路由跟踪技术、IP欺骗技术、密码截获及保护技术、网络封包截获技术、串口通信技术、IOCP技术，以及分层协议、NDIS中间层网络驱动编程等。

本书实例新颖，有很强的实用性，既适合于有一定C、C++语言基础，欲深入了解Windows网络编程的读者快速提高，也适合于从事网络编程的工作人员参考借鉴。

Visual C++网络程序设计实例详解

◆ 编　　著　张　越　等
责任编辑　刘　浩

◆ 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
新华书店总店北京发行所经销

◆ 开本：787×1092　1/16
印张：22
字数：535千字　　2006年7月第1版
印数：1-5 000册　　2006年7月北京第1次印刷

ISBN 7-115-14914-3/TP・5494

定价：39.00元（附光盘）

读者服务热线：(010)67132692　印装质量热线：(010)67129223

前　言

笔者结合自己这几年的工作经验精心编写了本书，希望广大编程爱好者、网络程序开发人员通过阅读本书，分析配套光盘上的代码，能够解决自己在学习、工作过程中遇到的问题。

全书共分为 9 章，内容涵盖 TCP 和 UDP 的客户/服务器编程、Internet 和 LAN 上的远程 PC 控制、链路层的计算机扫描技术、路由跟踪技术、IP 欺骗技术、密码截获及保护技术、网络封包截获技术、IOCP 技术，以及分层协议、NDIS 中间层网络驱动编程等。各章内容安排如下。

第 1 章主要向读者介绍使用 Winsock 开发典型客户/服务器程序的方法。本章精选了 5 个实例程序，包括 TCP 客户/服务器程序、多线程 TCP 服务器、通过 Internet 传输文件以及时间协议的具体应用。

第 2 章讲述比较流行的计算机嗅探技术和远程控制技术。本章选取的实例有 Ping 程序实现、路由跟踪、网络嗅探器、远程进程和机器控制以及网络唤醒。

第 3 章（网络计算机扫描）详细讲述各种网络扫描技术的基本原理和实现方法。本章的例子包括原始 UDP 封包发送程序、原始以太封包发送程序、TCP/IP 端口扫描程序和高级 TCP 半开端口扫描程序。

第 4 章（网络封包过滤技术）讲述各种流行的封包过滤技术。本章首先概述 Windows 下各种网络数据包拦截技术，然后详细讲述使用挂钩 API、分层协议以及过滤钩子驱动过滤网络封包的方法。

第 5 章（NDIS 中间层驱动开发）将详细讲述 NDIS 中间层驱动的开发，带领读者一步步扩展 Windows DDK 自带的中间层驱动 PassThru 的功能，最终得到一个有着基本 IOCTL 用户接口，能够按照用户要求过滤数据，向用户报告网络活动状态的 NDIS 中间层驱动程序 PassThruEx。

第 6 章（网络安全）讲述网络时代人们最为关心的网络安全问题，本章的实例程序有局域网计算机诊测实例、ARP 欺骗与 ARP 表中毒实例、密码检测实例。

第 7 章（IP 帮助函数的使用）通过各种实例程序详细讲述常用 IP 帮助函数的使用方法。

第 8 章（串口通信编程技术）讲述编写串口通信 程序的方法，这章的例子有异步通信程序、文件传输程序和串口调试程序等。

第 9 章（综合实例）介绍一些综合性比较强的网络实例，如 IP 多播、组讨论会、IOCP 编程、基于 IOCP 的 IP 多播等。

参加本书编写的还有李樱、张巍、徐文、董肖、李亚平、杨云、李立雪、杨在海、汪玲、李渝平、张鲁萍、易逐非、李康平、周长安、刘健、于思锋、周一栗、何平、何亮、胡洁等，在此表示感谢！由于时间仓促，加之水平有限，不妥之处还望读者批评指正（电子函件：book_better@sina.com）。

编者

2006 年 6 月

目　录

第 1 章　客户机/服务器开发

本章主要向读者介绍使用 Winsock 开发典型客户机/服务器程序的方法。本章精选了 5 个实例程序从易到难，包括 TCP 客户机/服务器程序、多线程 TCP 服务器、通过 Internet 传输文件，以及时间协议的具体应用。

1.1　Winsock 编程入门——简单的 TCP 服务器

本节通过一个简单的 TCP 服务器实例来讲述基本的 Winsock 编程。实例程序在配套光盘的 SimpleTcpServerSrc 工程下。

1.1.1　Winsock 编程简介

Winsock 是 Windows 下网络编程的标准接口，它允许两个或多个应用程序在同一台电脑上，或者是通过网络相互交流。Winsock 是真正的协议无关的接口，本节详细介绍 Winsock 编程的方法。

如果仅有一台电脑，也仍然能够进行 Winsock 编程。可以使用本地回环地址 127.0.0.1。这样的话，如果有一个 TCP 服务器运行在你的电脑上，那么运行在此电脑上的客户程序使用这个回环地址，即可连接到服务器。

1.1.2　简单的 TCP 服务器

下面一步步创建简单的 TCP 服务器。在此之前，还要做下面这些准备工作，以便可以真正的进行 Winsock 编程。

- 使用 VC++ 6.0 App Wizard 创建一个 Win32 控制台应用程序。
- 记住要设置选项，以便添加对 MFC 的支持。
- 打开文件 stdafx.h，添加语句：#include <winsock2.h>。
- 在 winsock2.h 之后，还要包含 conio.h 和 iostream.h。
- 打开菜单 Project\Settings，切换到 Link 选项卡，向库模块列表中添加 ws2_32.lib。或者，直接在代码中添加语句：#pragma comment(lib, "WS2_32")，以便链接到 WS2_32.DLL 库。因为所有的 Winsock 函数都是从 WS2_32.DLL 导出的。

1. main 函数

在 main 函数中启动一个线程，然后循环调用_getch()。_getch()仅简单地等待键盘输入，返回被按键的 ASCII 值。ESCAPE 键的 ASCII 值是 27，所以函数返回 27 后，程序才退出循环。当 main 函数返回，进程将终止，进程内的线程也将会终止。上述过程的具体程序代码如下。

```
int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
```

```
{
    int nRetCode = 0;
    cout << "Press ESCAPE to terminate program\r\n";
    AfxBeginThread(ServerThread,0);
    while(_getch()!=27);
    return nRetCode;
}
```

2. ServerThread 函数

下面要做的是列出 ServerThread 函数的实现代码，使用注释来解释每行代码的作用。TCP 服务器启动之后在端口 20248 监听。当有客户连接时，服务器向客户发回一个消息，告诉客户自己的 IP 地址，然后关闭连接，并继续到端口 20248 去接受连接。上述过程的具体程序代码如下。

```
UINT   ServerThread(LPVOID pParam)
{
    cout << "Starting up TCP server\r\n";
    // SOCKET 实际上是一个 unsigned int 类型
    SOCKET server;
    // WSADATA 结构将有 WSAStartup 函数来填充
    WSADATA wsaData;
    // 对 TCP/IP 套接字来说，sockaddr_in 结构指定了套接字的地址
    // 其他协议使用相似的结构
    sockaddr_in local;
    // WSAStartup 函数负责加载 Winsock 库的函数
    // 第一个参数用来指定想要加载的 Winsock 库的版本
    int wsaret=WSAStartup(0x101,&wsaData);
    // 调用成功，WSAStartup 函数返回 0
    // 失败的话，程序就退出
    if(wsaret!=0)
    {
        return 0;
    }
    // 现在我们填充 sockaddr_in 结构
    local.sin_family=AF_INET;                    // 地址家族 Address family
    local.sin_addr.s_addr=INADDR_ANY;            // 要求使用当前主机配置的所有 IP 地址
    local.sin_port=htons((u_short)20248);        // 使用的端口
    // socket 函数用来创建我们的 SOCKET
    server=socket(AF_INET,SOCK_STREAM,0);
    // 如果 socket()函数调用失败，我们退出
```

```
if(server==INVALID_SOCKET)
{
    return 0;
}
// 为套接字关联本地地址的函数是 bind.。
// bind 函数用在没有建立连接的套接字上，它的作用是绑定面向连接的或者无连接的套接字。
// 套接字被 socket 函数创建以后，存在于指定的地址家族里，但它是未命名的。
// bind 函数通过安排一个本地名称到未命名的 socket 建立此 socket 的本地关联。
// 本地名称包含 3 个部分：主机地址、协议号（分别为 UDP 或 TCP）和端口号。
if(bind(server,(sockaddr*)&local,sizeof(local))!=0)
{
    return 0;
}
// listen 函数设置套接字进入监听状态
// 为了接受连接，首先使用 socket 函数创建一个套接字，然后使用 bind 函数将它绑定到一个本地地址，
// 再用 listen 函数为到达的连接指定一个 backlog，最后使用 accept 接受请求的连接。
// 函数的第二个参数是 backlog 值
if(listen(server,10)!=0)
{
    return 0;
}
// 我们需要变量保存客户套接字，下面是这些变量的定义
SOCKET client;
sockaddr_in from;
int fromlen=sizeof(from);
// 下面进入无限循环
while(true)
{
    char temp[512];
    // accept() 将接受到了的客户连接
    client=accept(server,(struct sockaddr*)&from,&fromlen);
    sprintf(temp,"Your IP is %s\r\n",inet_ntoa(from.sin_addr));
    // 我们简单地将这串字符发送给客户
    send(client,temp,strlen(temp),0);
    cout << "Connection from " << inet_ntoa(from.sin_addr) <<"\r\n";
    // 关闭客户套接字
    closesocket(client);
}
```

```
    // closesocket()关闭套接字，并且释放套接字描述表
    closesocket(server);
    // 每一个对 WSAStartup 的调用必须对应一个对 WSACleanup 的调用，这个函数释放 Winsock 库
    WSACleanup();
    return 0;
}
```

1.1.3 测试程序

运行此服务器，使用 telnet 连接到服务器所在机器的 20248 端口。下面是服务器端的输出：

```
E:\work\Server\Debug>server
Press ESCAPE to terminate program
Starting up TCP server
Connection from 203.200.100.122
Connection from 127.0.0.1
E:\work\Server\Debug>
```

下面是客户端的输出：

```
nish@sumida:~$ telnet 202.89.211.88 20248
Trying 202.89.211.88...
Connected to 202.89.211.88.
Escape character is '^]'.
Your IP is 203.200.100.122
Connection closed by foreign host.
nish@sumida:~$
```

1.2 Winsock 编程入门——简单的 TCP 客户端

本节讲述如何编写 TCP 客户端程序。本节的实例程序将连接到 HTTP 服务器，获取一个文件。实例程序在配套光盘的 SimpleTcpClientSrc 工程下，运行效果如图 1.1 所示。

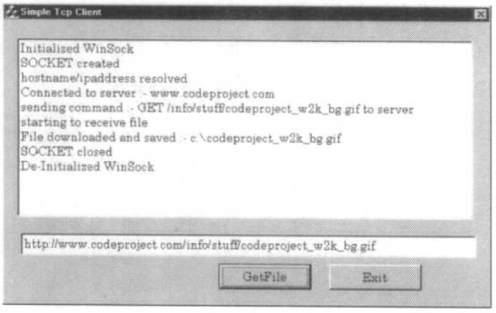

图 1.1 TCP 客户端运行效果

1.2.1　TCP 客户程序的编写流程

下面先介绍 TCP 客户程序的编写流程，然后再详细讨论每一步的具体实现。

（1）使用 WSAStartup()初始化 Winsock 库。

（2）使用 socket()创建一个 IPPROTO_TCP 类型的套接字。

（3）使用 gethostbyname()/gethostbyaddr()获取主机信息。

（4）使用 connect()连接到服务器。

（5）使用 send()/recv()发送和接收数据，直到 TCP 通信结束。

（6）使用 closesocket()关闭套接字连接。

（7）使用 WSACleanup()释放 Winsock 库。

1.2.2　初始化 Winsock 库

和其他所有的 Winsock 程序一样，首先初始化 Winsock 库。这也是检查 Winsock 在该系统上是否可用的一种方法。

```
int wsaret=WSAStartup(0x101,&wsaData);
if(wsaret)
    return;
```

1.2.3　创建套接字

套接字是在客户和服务器之间通信的实体。当客户连接到服务器时，就会有两个套接字：客户方套接字和对应的服务器端套接字，这里将他们称作 CLIENTSOCK 和 SERVERSOCK。当客户方在 CLIENTSOCK 使用 send()函数时，服务器方可以在 SERVERSOCK 上使用 recv()函数来获取客户发送的数据。反之亦然。以下是创建套接字的代码。

```
SOCKET conn;
conn=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
if(conn==INVALID_SOCKET)
    return;
```

1.2.4　获取主机信息

在连接到主机（服务器）之前，必须首先获取主机的信息。可以使用两个函数：gethostbyname()和 gethostbyaddr()。如果有服务器的 DNS 名称（例如，codeproject.com 或 ftp.myserver.org），要使用 gethostbyname()函数；当我们有服务器实际的 IP 地址（例如，192.168.1.1 或者 202.54.1.100），要使用 gethostbyaddr()函数。

为了方便使用，既允许用户输入 DNS 名称，也允许用户输入 IP 地址。程序中对用户的输入调用了 inet_addr()函数，此函数将 IP 地址转换为标准的网络地址格式。如果该函数返回失败，说明用户输入的不是 IP 地址，反之，说明用户输入了有效的 IP 地址。此过程的具体实现代码如下。

```
if(inet_addr(servername)==INADDR_NONE)
{
    hp=gethostbyname(servername);
```

```
}
else
{
    addr=inet_addr(servername);
    hp=gethostbyaddr((char*)&addr,sizeof(addr),AF_INET);
}
if(hp==NULL)
{
    closesocket(conn);
    return;
}
```

1.2.5 连接到服务器

connect()函数用来和目的服务器建立连接。参数为已创建的套接字和一个包含服务器地址信息的 sockaddr 结构。sockaddr 结构中的主机地址用 gethostbyname()/gethostbyaddr()的返回值来填充，同时也要指定一个有效的端口号。具体程序代码如下。

```
server.sin_addr.s_addr=*((unsigned long*)hp->h_addr);
server.sin_family=AF_INET;
server.sin_port=htons(80);
if(connect(conn,(struct sockaddr*)&server,sizeof(server)))
{
    closesocket(conn);
    return;
}
```

1.2.6 TCP 通信

套接字连接建立之后，客户和服务器可以使用 send()和 recv()函数交换数据。这个过程一般称作 TCP 通信。本节的例子中，我们需要 HTTP 通信，这比其他协议（如 SMTP 或 POP3）简单多了。HTTP GET 命令从 HTTP 服务器获取文件，可以是 HTML 文件、图像文件、MP3 等。GET 命令的使用格式如下（这里是最简单的情况）：

```
GET http-path-to-file\r\n\r\n
```

具体地使用如下代码发送 GET 命令。

```
sprintf(buff,"GET %s\r\n\r\n",filepath);
send(conn,buff,strlen(buff),0);
```

服务器接收到此命令，就开始向客户发送指定的文件。正如使用 send()发送命令一样，程序使用 recv()接收服务器发来的数据。下面是接收数据的程序代码。循环调用 recv()函数，直到它返回 0 为止。recv()函数返回 0 说明服务器已经完成了数据的发送。程序将接收到的数据写入到文件中。

```
while(y=recv(conn,buff,512,0))
```

```
{
    f.Write(buff,y);
}
```

1.2.7　关闭连接、释放 Winsock 库

数据传输完毕之后，必须关闭连接。本例中，HTTP 连接是由服务器端在传输完文件之后关闭的，客户端也要关闭套接字，释放资源。在更加复杂的程序中，通常在调用 closesocket()之前先调用 shutdown()函数，以确保缓冲区中的数据传输完毕。下面是关闭连接的程序代码。

```
closesocket(conn);
```

最后，还要调用 WSACleanup()释放 Winsock 库，代码如下所示。

```
WSACleanup();
```

1.3　多线程 TCP 服务器和客户端实例

本节的实例是一个多线程的 TCP 文件服务器，它使用自定义通信协议向客户方传递文件。实例程序保存在配套光盘的 MTSClientSrc（客户端）和 MTServerSrc（服务器端）工程下。

1.3.1　实例介绍

1.1 节讲述的 TCP 服务器在同一时刻仅能接收一个连接。我们知道，一个服务器程序必须能够在任何时间处理多个客户连接。本节将写一个多线程 TCP 服务器。并且还将创建自定义的基于 TCP 的通信协议。还要编写一个客户端程序，它将连接到服务器，使用该自定义协议与服务器通信。服务器程序的功能是向客户程序发送请求的文件，客户程序将这个文件保存到本地计算机中。

这里使用的处理多个客户的方法是最原始的一个客户连接一个线程的方法。有许多其他更加有效的处理多个连接的机制，如 I/O 完成端口模型等。但是，如果通信协议比较简单，每个客户连接对速度要求并不太苛刻的话，使用本节的方法还是很合理的，除非同一时间连接的客户数量特别巨大。对这方面的内容感兴趣的读者可以参考《Windows 网络与通信程序设计》一书（人民邮电出版社出版）。

1.3.2　多线程服务器

在 main()函数中，本节例子的代码和前面的 TCP 服务器没有太大的不同，都是先启动服务器线程，然后循环调用_getch()函数，直到用户按下 ESC 键。具体代码如下：

```
int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
    int nRetCode = 0;
    cout << "Press ESCAPE to terminate program\r\n";
    AfxBeginThread(MTServerThread,0);
    while(_getch()!=27);
```

```
    closesocket(server);
    WSACleanup();
    return nRetCode;
}
```

下面集中精力研究服务器线程。在服务器线程中，直到调用 listen()为止，程序代码都与1.1 节的服务器程序完全相同。在调用 accept()部分我们做了一些微小的改动，代码如下所示。

```
UINT  MTServerThread(LPVOID pParam)
{
    /* 这里的代码被省略了
    ……
    */
    if(listen(server,10)!=0)
    {
        return 0;
    }
    SOCKET client;
    sockaddr_in from;
    int fromlen=sizeof(from);
    while(true)
    {
        client=accept(server,
            (struct sockaddr*)&from,&fromlen);
        AfxBeginThread(ClientThread,(LPVOID)client);
    }
    return 0;
}
```

MTServerThread 接收到连接后，启动一个新的线程来处理这个连接，并将客户的套接字句柄传递给新的线程，然后再返回继续调用 accept()。这样，客户连接的时候，一个新的线程就会启动来处理这个客户的 I/O 请求，再有客户连接的时候，又会创建一个新的线程，如此反复下去。

1.3.3 自定义传输协议

为了实施文件传输，必须首先定义一个简单的协议，下面来定义这个协议使用的命令。显然需要一个命令来关闭会话，这个命令是“QUIT”。因为不想每个人都可以下载文件，所以添加“AUTH”命令，此命令的参数用来指定密码。如果密码正确，我们就将用户设为授权状态。AUTH 和 QUIT 命令都可以在未授权状态使用。但是用来获取文件的命令“FILE”仅允许在授权状态使用。总结一下，共定义了如下 3 个命令。

- QUIT：此命令将关闭连接。
- AUTH [password]：此命令将使用户进入授权状态。

- FILE [filename with full path]：获取指定文件（仅在授权状态才可用）。

为了说明自定义协议是如何工作的，使用#来表示成功的消息，使用!来表示错误消息，下面完整地显示了一个 TCP 会话过程（在客户端看来）。

```
Trying 192.168.1.44...
Connected to 192.168.1.44.
Escape character is '^]'.
#Server Ready.
file c:\config.sys
!You are not logged in.
auth yellow
!Bad password.
auth passwd
#You are logged in.
file c:\config.sys
DEVICE=C:\WINDOWS\HIMEM.SYS
DEVICE=C:\WINDOWS\EMM386.EXE
#File c:\config.sys sent successfully.
file c:\setup.log
[InstallShield Silent]
Version=v6.00.000
File=Log File

[ResponseResult]
ResultCode=0

[Application]
Name=Intel Ultra ATA Storage Driver
Version=6.03.007
Company=Intel
Lang=0009
#File c:\setup.log sent successfully.
file d:\g5.doc
!File d:\g5.doc could not be opened.
quit
Connection closed by foreign host.
```

可以看到，用户登录之后，他可以请求任意数量的文件。文本文件和二进制文件均可。这些命令的具体实现是在 ClientThread 线程函数中进行的，下面是具体程序代码。

```
UINT   ClientThread(LPVOID pParam)
{
```