

[articles](#) [Q&A](#) [forums](#) [lounge](#)

Search for articles, questions, tips



Beginning Winsock Programming - Multithreaded TCP server with client

**Nish Nishant**, 6 Mar 2002

Rate:

★★★★★ 4.90 (102 votes)

Explains a multithreaded TCP file server, a custom TCP chat protocol and a custom TCP client



Is your email address OK? You are signed up for our newsletters but your email address is either unconfirmed, or has not been reconfirmed in a long time. Please [click here to have a confirmation email sent](#) so we can confirm your email address and start sending you newsletters again. Alternatively, you can [update your subscriptions](#).

[Download server project - 7 Kb](#)[Download client project - 7 Kb](#)

Introduction

In previous articles in this series we went through writing a [simple TCP server](#) that could accept a single connection at a time and also a [simple TCP client](#) that could download a file via HTTP. But it must have been obvious to most of you that a server program must surely be able to handle more than one client connection at any time. Otherwise if a client is currently connected, other clients wouldn't find the server to be of much utility to them.

In this article we shall write a multithreaded TCP server. In addition we shall also create our own custom TCP chat protocol, albeit a very simple one. We will also then write a client that will connect to this server and chat using this protocol. You might try running multiple versions of the client at the same time to test whether the server can indeed accept multiple connections. The server simply sends requested files to the client which then saves it on the client machine.

The method used here for handling multiple clients is the age-old method of one thread per client connection. There are several other more efficient mechanisms for handling multiple connections like IO completion ports. But if the chat protocol is elementary and each client connection is not very processor-intensive then this is a pretty reasonable method unless the number of clients that would connect at any one time is abnormally large.

Writing the multithreaded server

Those of you who have read my article on writing a [simple TCP server](#) would probably know how a TCP server is created. Others might be better off to go and read that article first. Others can continue. There is not much difference in our `main()` function. We start the server thread and loop endlessly on `_getch()` till someone presses ESC and then we close the server socket and exit.

[Hide](#) [Copy Code](#)

```
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;
}

```

Now lets take a look at the server thread. Up to the part where we call `listen()` the code is essentially the same. It's in the `accept()` part where we make our small change.

[Hide](#) [Copy Code](#)

```

UINT MTServerThread(LPVOID pParam)
{
    /*
     * Code has been snipped off.
     * Refer zipped source for actual code
     */

    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;
}

```

What happens is essentially simple. We accept a connection and the moment we accept it, we start off a new thread passing to the thread the client `SOCKET`. Then we return back to `accept()`. Thus the moment a client connects, a new thread is started to handle the client and thus the next client can also connect which will start off yet another thread and so on and so forth. Oh boy! And to think, some of you guys actually expected it to be a lot harder than that eh?

Our own custom protocol

Let's define the commands allowed in our own TCP chat protocol. We'll obviously need a command for closing the session. How about "QUIT"? That seems to be a nice obvious word for that. Now we don't want just about everyone to download files. So let's add an "AUTH" command that has a parameter which specifies the password. If the password is correct then we put the user into the authorized state. Both AUTH and QUIT may be used in the non-authorized state. But the command "FILE" which is used to retrieve files, is allowed only in the authorized state. Well, that's three commands, two of them allowed at all times and one of them allowed only during authorized state.

- **QUIT** :- This will close the connection
- **AUTH [password]** :- This will log the user into authorized mode
- **FILE [filename with full path]** :- Retrieves a file [works only in authorized mode]

Fancy that! Our own nice little protocol. Lets use # to denote success messages and ! to denote error messages. I'll now show you how a simple TCP chat session to our server will look like before I actually show you how it is implemented in code.

[Hide](#) [Shrink](#) ▲ [Copy Code](#)

```
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.

```

As you can see, once a user is logged in, he can request any number of files. I've shown only text files here, but he might as well demand binary files too. In the client program we will write later, we can do this too. Right now let's look at how the client thread handles this chat protocol.

Hide Shrink ▲ Copy Code

```

UINT ClientThread(LPVOID pParam)
{
    char buff[512];
    CString cmd;
    CString params;
    int n;
    int x;
    BOOL auth=false;
    SOCKET client=(SOCKET)pParam;
    strcpy(buff, "#Server Ready.\r\n");
    send(client, buff, strlen(buff), 0);
    while(true)
    {
        n=recv(client, buff, 512, 0);
        if(n==SOCKET_ERROR )
            break;
        buff[n]=0;
        if(ParseCmd(buff, cmd, params))
        {
            if(cmd=="QUIT")
                break;
            if(cmd=="AUTH")
            {
                if(params=="passwd")
                {
                    auth=true;
                    strcpy(buff, "#You are logged in.\r\n");
                }
                else
                {
                    strcpy(buff, "!Bad password.\r\n");
                }
                send(client, buff, strlen(buff), 0);
            }
            if(cmd=="FILE")
            {

```

```

        if(auth)
        {
            if(SendFile(client,params))
                sprintf(buff,
                    "#File %s sent successfully.\r\n",
                    params);
            else
                sprintf(buff,
                    "!File %s could not be opened.\r\n",
                    params);
            x = send(client, buff,
                strlen(buff),0);
        }
        else
        {
            strcpy(buff,"!You are not logged in.\r\n");
            send(client,buff,strlen(buff),0);
        }
    }
}

else
{
    strcpy(buff,"!Invalid command.\r\n");
    send(client,buff,strlen(buff),0);
}
}

closesocket(client);
return 0;
}

```

Hopefully, the code is self-explanatory. I'll just run through it briefly. We first send a server greeting as is customary among TCP servers. Now we keep looping and accepting commands. We use our function **ParseCmd** to parse the command string entered into two **CString** objects, one containing the command and the other containing the arguments if any. If **ParseCmd** returns **true** it means an unknown command was sent, and we give back an error message.

If the command is QUIT we break out of the while loop, close the client socket and exit the thread. We also have a boolean flag called **auth** and we make this true only after the client has authorized itself. Till then, if we get a FILE command we send back a message saying that the client is not logged in. Right now I have hard coded "passwd" as our password, but in a real situation, the username/password will be taken from some database or configuration file.

Using the AUTH command the user can log in. We compare with our password and if they match we send a message telling them they are logged in and set the **auth** flag to true, else we give them a bad login error message. Once they are authorized they can request files. We use a function called **SendFile** to send the files across the TCP connection. I hope things are clear now.

Now lets look at the **ParseCmd** function

Hide Copy Code

```

BOOL ParseCmd(char *str, CString& cmd, CString& params)
{
    int n;
    CString tmp=str;
    tmp.TrimLeft();
    tmp.TrimRight();
    if((n=tmp.Find(' '))!=-1)
    {
        tmp.MakeUpper();
        if(tmp!="QUIT")
            return false;
        cmd=tmp;
        return true;
    }
    cmd=tmp.Left(n);
    params=tmp.Mid(n+1);
    cmd.MakeUpper();
    if((cmd!="AUTH") && (cmd!="FILE"))
        return false;
    return true;
}

```

Well as you can see, the function is pretty much straightforward. It splits the string and returns **true** if it encounters a valid command. Otherwise it returns **false** which indicates error.

Now let's take a look at the **SendFile** function.

[Hide](#) [Copy Code](#)

```

BOOL SendFile(SOCKET s, CString fname)
{
    CFile f;
    BOOL p=f.Open(fname,CFile::modeRead);
    char buff[1024];
    int y;
    int x;
    if(!p)
        return false;
    while(true)
    {
        y=f.Read(buff,1024);
        x=send(s,buff,y,0);
        if(y<1024)
        {
            f.Close();
            break;
        }
    }
    return true;
}

```

This is also a simple function. It returns **true** if the file was successfully sent and **false** if the file was not found. Bit confusing isn't it. I used **true** to represent an error in **ParseCmd** and here I am using **false** to represent an error. I guess I have a long way to go as far as coding standards are concerned. I hope you nice ladies and gentlemen will pardon this grave failing in my personality.

Alright now we have written our multithreaded TCP server with our own custom chat protocol. That's all very well I guess. But I bet some of you are now thinking that perhaps it would be nice to actually write a nice little client program that connects to the server, and chats with it using our protocol, and retrieves a few files as well. Hmmm. By an amazing coincidence I have had the very same idea too. Therefore we shall proceed to write a client program.

The custom client

I won't go through the source code this time. Those of you who have gone through my [simple TCP client](#) would have absolutely no problem in downloading and understanding the source code. Just a few points though. Do not use the string manipulation functions like **strcpy()** and **strchr()** on the buffers returned by **recv()**. These byte buffers may not be null terminated and calling functions like **strcpy()** will simply wreck your program. You may open the source and look into two functions I have used to search the buffer for a certain set of characters. I have not used the string manipulation functions at all.

Here is how to use the custom client. By the way I have hard coded 127.0.0.1 into the client source code as the server IP. You might want to change that if you are running the server and client on two different machines. Before running please make sure that the server is up and running.

[Hide](#) [Copy Code](#)

```

E:\work\MTSCClient\Debug>mtscclient
Usage :- mtscclient [file1] [file2] [file3] ....

E:\work\MTSCClient\Debug>mtscclient c:\cu.gif c:\cp.gif c:\g.gif c:\ddd
File c:\cu.gif not found on server
cp.gif has been saved.
g.gif has been saved.
File c:\ddd not found on server

E:\work\MTSCClient\Debug>

```

Conclusion

This is probably the last in my 3 part series of introductory articles on TCP programming using Winsock. By now you must be able to write basic TCP clients and TCP servers [multithreaded]. I suggest that you try writing a simple program that connects to a POP server, logs in and checks whether there is any mail in there. Or you might try writing a small program that uses SMTP chat to send a mail. To test your server coding skills, you might want to write a simple HTTP server. Or think up something and write your own custom server with your own custom protocol. Anyway, have fun...

License

This article has no explicit license attached to it but may contain usage terms in the article text or the download files themselves. If in doubt please contact the author via the discussion board below.

A list of licenses authors might use can be found [here](#)

Share

EMAIL

About the Author



Nish Nishant

United States 

Nish Nishant is a Software Architect/Consultant based out of **Columbus, Ohio**. He has over 16 years of software industry experience in various roles including **Lead Software Architect**, **Principal Software Engineer**, and **Product Manager**. Nish is a recipient of the annual **Microsoft Visual C++ MVP** Award since 2002 (14 consecutive awards as of 2015).

Nish is an industry acknowledged expert in the Microsoft technology stack. He authored [C++/CLI in Action](#) for Manning Publications in 2005, and had previously co-authored [Extending MFC Applications with the .NET Framework](#) for Addison Wesley in 2003. In addition, he has over 140 published technology articles on CodeProject.com and another 250+ blog articles on his [WordPress blog](#). Nish is vastly experienced in team management, mentoring teams, and directing all stages of software development.

Contact Nish : You can reach Nish on his google email id **voidnish**.

Website and Blog

- www.voidnish.com
- voidnish.wordpress.com


You may also be interested in...

[Kendo UI in SharePoint Online](#)


[SAPrefs - Netscape-like Preferences Dialog](#)

[The Best Angular 2 Data Grid: FlexGrid](#)[Generate and add keyword variations using AdWords API](#)[10 Ways to Boost COBOL Application Development](#)[Window Tabs \(WndTabs\) Add-In for DevStudio](#)


Comments and Discussions




[First](#)
[Prev](#)
[Next](#)

My vote of 5 


hoseinhero 19-Dec-12 21:35

My vote of 5 


son.phuoc 20-Apr-12 16:06

send data from one client to another client throgh server 


vahidhwp 1-Oct-09 23:55

A Universal C++ TCP Socket Class for Non-blocking Server/Clients 


Elmue 23-Mar-09 3:03

How to listen / control multiple ports/sockets 


texnguy 15-Feb-08 3:07

send message to specific client 


psychope 6-Sep-07 17:59

How to close tcp connection? 


donj82 6-Jun-07 21:22

any suggestion for sending unsigned char * messages? 


arbel kfir 11-Jan-07 17:00

Re: any suggestion for sending unsigned char * messages? 


arbel kfir 16-Jan-07 15:00

Is Winsock thread safe ?? 

j_cheng 1-Apr-06 11:37

Re: Is Winsock thread safe ?? 

Diego.Martinez 3-May-06 20:21

chat server 

aftab ali shah 19-Jul-05 1:23

multithreded server **cool_vc++** 4-Mar-05 23:34**more than one message or less than one message** **Moshe Gershberg** 24-Dec-04 4:12

Re: more than one message or less than one message

Mike O'Neill 7-Feb-05 2:50**how to assign a specific ip address to server and client** **syaks** 26-Oct-04 17:44**Help** **asv** 28-Aug-04 23:02**CSocket Issues** **dharani** 22-Jun-04 11:26**I cant connect to this server** **Anonymous** 5-Jun-04 7:34**about timeout** **LIMO** 5-Mar-04 18:53**telnet** **Gadmovorg** 12-Jan-04 17:04

Re: telnet

Christos Ioannou 9-Aug-04 22:43**Authorisation** **Ancient Scientist** 10-Jan-04 6:51**memory leak in server?** **bryce** 29-Aug-03 12:10**Proxy server-help !** **dharani** 8-Aug-03 17:44[Refresh](#)[1](#) [2](#) [3](#) [4](#) [5](#) [Next »](#)
[General](#)
[News](#)
[Suggestion](#)
[Question](#)
[Bug](#)
[Answer](#)
[Joke](#)
[Praise](#)
[Rant](#)
[Admin](#)

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.

[Permalink](#) | [Advertise](#) | [Privacy](#) | [Terms of Use](#) | [Mobile](#)

Web02 | 2.8.160621.1 | Last Updated 6 Mar 2002

請選取語言 ▼

Layout: [fixed](#) | [fluid](#)Article Copyright 2002 by Nish Nishant
Everything else Copyright © [CodeProject](#), 1999-2016