Winsock tutorial – Socket programming in C on windows

Winsock By Silver Moon On Dec 26, 2011 51 Comments

Socket programming with winsock

This is a quick guide/tutorial to learning socket programming in C language on Windows. "Windows" because the code snippets shown over here will work only on Windows. The windows api to socket programming is called winsock.

Sockets are the fundamental "things" behind any kind of network communications done by your computer. For example when you type www.google.com in your web browser, it opens a socket and connects to google.com to fetch the page and show it to you. Same with any chat client like gtalk or skype. Any network communication goes through a socket.

Before you begin

This tutorial assumes that you have basic knowledge of C and pointers. Also download Visual C++ 2010 Express Edition.

Initialising Winsock

Winsock first needs to be initialiased like this :

```c
/*
    Initialise Winsock
*/

#include<stdio.h>
#include<winsock2.h>

#pragma comment(lib,"ws2_32.lib") //Winsock Library

int main(int argc , char *argv[])
{
    WSADATA wsa;

    printf("\nInitialising Winsock...");
    if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
    {
        printf("Failed. Error Code : %d",WSAGetLastError());
        return 1;
    }

    printf("Initialised.");

    return 0;
}
```

winsock2.h is the header file to be included for winsock functions. ws2_32.lib is the library file to be linked with the program to be able to use winsock functions.

The WSAStartup function is used to start or initialise winsock library. It takes 2 parameters ; the first one is the version we want to load and second one is a WSADATA structure which will hold additional information after winsock has been loaded.

Connect with us

Other interesting stuff


Packet Sniffer Code in C using Winsock


UDP socket programming in winsock


Get mac address from ip in winsock

TCP Connect Port Scanner Source Code in C wit Winsock

If any error occurs then the WSAStartup function would return a non zero value and WSAGetLastError can be used to get more information about what error happened.

OK , so next step is to create a socket.

Creating a socket

The `socket()` function is used to create a socket.
Here is a code sample :

```c
/*
    Create a TCP socket
*/

#include<stdio.h>
#include<winsock2.h>

#pragma comment(lib,"ws2_32.lib") //Winsock Library

int main(int argc , char *argv[])
{
    WSADATA wsa;
    SOCKET s;

    printf("\nInitialising Winsock...");
    if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
    {
        printf("Failed. Error Code : %d",WSAGetLastError());
        return 1;
    }

    printf("Initialised.\n");


    if((s = socket(AF_INET , SOCK_STREAM , 0 )) == INVALID_SOCKET)
    {
        printf("Could not create socket : %d" , WSAGetLastError());
    }

    printf("Socket created.\n");

    return 0;
}
```

Function `socket()` creates a socket and returns a socket descriptor which can be used in other network commands. The above code will create a socket of :

Address Family : AF_INET (this is IP version 4)
Type : SOCK_STREAM (this means connection oriented TCP protocol)
Protocol : 0 [or IPPROTO_TCP , IPPROTO_UDP]

It would be a good idea to read some documentation here

Ok , so you have created a socket successfully. But what next ? Next we shall try to connect to some server using this socket. We can connect to www.google.com

Note

Apart from SOCK_STREAM type of sockets there is another type called SOCK_DGRAM which indicates the UDP protocol. This type of socket is non-connection socket. In this tutorial we shall stick to SOCK_STREAM or TCP sockets.

Connect to a Server

We connect to a remote server on a certain port number. So we need 2 things , IP address and port number to connect to.

To connect to a remote server we need to do a couple of things. First is create a sockaddr_in structure with proper values filled in. Lets create one for ourselves :

```c
struct sockaddr_in server;
```



Raw socket programming on windows with winsock


DNS Query Code in C with winsock

Have a look at the structures

```
// IPv4 AF_INET sockets:
struct sockaddr_in {
    short           sin_family;   // e.g. AF_INET, AF_INET6
    unsigned short   sin_port;     // e.g. htons(3490)
    struct in_addr   sin_addr;     // see struct in_addr, below
    char            sin_zero[8];  // zero this if you want to
};


typedef 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;
} IN_ADDR, *PIN_ADDR, FAR *LPIN_ADDR;


struct sockaddr {
    unsigned short    sa_family;    // address family, AF_xxx
    char              sa_data[14];  // 14 bytes of protocol address
};
```

The sockaddr_in has a member called sin_addr of type in_addr which has a s_addr which is nothing but a long. It contains the IP address in long format.

Function `inet_addr` is a very handy function to convert an IP address to a long format. This is how you do it :

```
server.sin_addr.s_addr = inet_addr("74.125.235.20");
```

So you need to know the IP address of the remote server you are connecting to. Here we used the ip address of google.com as a sample. A little later on we shall see how to find out the ip address of a given domain name.

The last thing needed is the `connect` function. It needs a socket and a sockaddr structure to connect to. Here is a code sample.

```
/*
    Create a TCP socket
*/

#include<stdio.h>
#include<winsock2.h>

#pragma comment(lib,"ws2_32.lib") //Winsock Library

int main(int argc , char *argv[])
{
    WSADATA wsa;
    SOCKET s;
    struct sockaddr_in server;

    printf("\nInitialising Winsock...");
    if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
    {
        printf("Failed. Error Code : %d",WSAGetLastError());
        return 1;
    }

    printf("Initialised.\n");

    //Create a socket
    if((s = socket(AF_INET , SOCK_STREAM , 0 )) == INVALID_SOCKET)
    {
        printf("Could not create socket : %d" , WSAGetLastError());
    }

    printf("Socket created.\n");


    server.sin_addr.s_addr = inet_addr("74.125.235.20");
    server.sin_family = AF_INET;
    server.sin_port = htons( 80 );

    //Connect to remote server
    if (connect(s , (struct sockaddr *)&server , sizeof(server)) < 0)
    {
        puts("connect error");
        return 1;
    }

    puts("Connected");

```

```
    return 0;
}
```

It cannot be any simpler. It creates a socket and then connects. If you run the program it should show Connected. Try connecting to a port different from port 80 and you should not be able to connect which indicates that the port is not open for connection.

OK , so we are now connected. Lets do the next thing , sending some data to the remote server.

Quick Note

The concept of "connections" apply to SOCK_STREAM/TCP type of sockets. Connection means a reliable "stream" of data such that there can be multiple such streams each having communication of its own. Think of this as a pipe which is not interfered by other data.

Other sockets like UDP , ICMP , ARP dont have a concept of "connection". These are non-connection based communication. Which means you keep sending or receiving packets from anybody and everybody.

Sending Data

Function `send` will simply send data. It needs the socket descriptor , the data to send and its size.
Here is a very simple example of sending some data to google.com ip :

```
/*
    Create a TCP socket
*/

#include<stdio.h>
#include<winsock2.h>

#pragma comment(lib,"ws2_32.lib") //Winsock Library

int main(int argc , char *argv[])
{
    WSADATA wsa;
    SOCKET s;
    struct sockaddr_in server;
    char *message;

    printf("\nInitialising Winsock...");
    if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
    {
        printf("Failed. Error Code : %d",WSAGetLastError());
        return 1;
    }

    printf("Initialised.\n");

    //Create a socket
    if((s = socket(AF_INET , SOCK_STREAM , 0 )) == INVALID_SOCKET)
    {
        printf("Could not create socket : %d" , WSAGetLastError());
    }

    printf("Socket created.\n");


    server.sin_addr.s_addr = inet_addr("74.125.235.20");
    server.sin_family = AF_INET;
    server.sin_port = htons( 80 );

    //Connect to remote server
    if (connect(s , (struct sockaddr *)&server , sizeof(server)) < 0)
    {
        puts("connect error");
        return 1;
    }

    puts("Connected");

    //Send some data
    message = "GET / HTTP/1.1\r\n\r\n";
    if( send(s , message , strlen(message) , 0) < 0)
    {
        puts("Send failed");
        return 1;
    }
    puts("Data Send\n");

    return 0;
}
```

In the above example , we first connect to an ip address and then send the string message "GET / HTTP/1.1\r\n\r\n" to it.

The message is actually a http command to fetch the mainpage of a website.

Now that we have send some data , its time to receive a reply from the server. So lets do it.

Receiving Data

Function `recv` is used to receive data on a socket. In the following example we shall send the same message as the last example and receive a reply from the server.

```
/*
    Create a TCP socket
*/

#include<stdio.h>
#include<winsock2.h>

#pragma comment(lib,"ws2_32.lib") //Winsock Library

int main(int argc , char *argv[])
{
    WSADATA wsa;
    SOCKET s;
    struct sockaddr_in server;
    char *message , server_reply[2000];
    int recv_size;

    printf("\nInitialising Winsock...");
    if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
    {
        printf("Failed. Error Code : %d",WSAGetLastError());
        return 1;
    }

    printf("Initialised.\n");

    //Create a socket
    if((s = socket(AF_INET , SOCK_STREAM , 0 )) == INVALID_SOCKET)
    {
        printf("Could not create socket : %d" , WSAGetLastError());
    }

    printf("Socket created.\n");


    server.sin_addr.s_addr = inet_addr("74.125.235.20");
    server.sin_family = AF_INET;
    server.sin_port = htons( 80 );

    //Connect to remote server
    if (connect(s , (struct sockaddr *)&server , sizeof(server)) < 0)
    {
        puts("connect error");
        return 1;
    }

    puts("Connected");

    //Send some data
    message = "GET / HTTP/1.1\r\n\r\n";
    if( send(s , message , strlen(message) , 0) < 0)
    {
        puts("Send failed");
        return 1;
    }
    puts("Data Send\n");

    //Receive a reply from the server
    if((recv_size = recv(s , server_reply , 2000 , 0)) == SOCKET_ERROR)
    {
        puts("recv failed");
    }

    puts("Reply received\n");

    //Add a NULL terminating character to make it a proper string before printing
    server_reply[recv_size] = '\0';
    puts(server_reply);

    return 0;
}
```

Here is the output of the above code :

```
Initialising Winsock...Initialised.
Socket created.
Connected
Data Send

Reply received
```

```
HTTP/1.1 302 Found
Location: http://www.google.co.in/
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Set-Cookie: PREF=ID=7da819edfd7af808:FF=0:TM=1324882923:LM=1324882923:S=PdlMu0TE
E3QKrmdB; expires=Wed, 25-Dec-2013 07:02:03 GMT; path=/; domain=.google.com
Date: Mon, 26 Dec 2011 07:02:03 GMT
Server: gws
Content-Length: 221
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.co.in/">here</A>.
</BODY></HTML>

Press any key to continue
```

We can see what reply was send by the server. It looks something like Html, well IT IS html. Google.com replied with the content of the page we requested. Quite simple!

Now that we have received our reply, its time to close the socket.

Close socket

Function `closesocket` is used to close the socket. Also WSACleanup must be called to unload the winsock library (ws2_32.dll).

```
closesocket(s);
WSACleanup();
```

Thats it.

Lets Revise

So in the above example we learned how to :
1. Create a socket
2. Connect to remote server
3. Send some data
4. Receive a reply

Its useful to know that your web browser also does the same thing when you open www.google.com
This kind of socket activity represents a **CLIENT**. A client is a system that connects to a remote system to fetch or retrieve data.

The other kind of socket activity is called a **SERVER**. A server is a system that uses sockets to receive incoming connections and provide them with data. It is just the opposite of Client. So www.google.com is a server and your web browser is a client. Or more technically www.google.com is a HTTP Server and your web browser is an HTTP client.

Now its time to do some server tasks using sockets. But before we move ahead there are a few side topics that should be covered just incase you need them.

Get IP address of a hostname/domain

When connecting to a remote host , it is necessary to have its IP address. Function `gethostbyname` is used for this purpose. It takes the domain name as the parameter and returns a structure of type hostent. This structure has the ip information. It is present in `netdb.h`. Lets have a look at this structure

```
/* Description of data base entry for a single host.  */
struct hostent
{
  char *h_name;         /* Official name of host.  */
  char **h_aliases;     /* Alias list.  */
  int h_addrtype;       /* Host address type.  */
  int h_length;         /* Length of address.  */
  char **h_addr_list;       /* List of addresses from name server.  */
};
```

The `h_addr_list` has the IP addresses. So now lets have some code to use them.

```c
/*
    Get IP address from domain name
*/

#include<stdio.h>
#include<winsock2.h>

#pragma comment(lib,"ws2_32.lib") //Winsock Library

int main(int argc , char *argv[])
{
    WSADATA wsa;
    char *hostname = "www.google.com";
    char ip[100];
    struct hostent *he;
    struct in_addr **addr_list;
    int i;

    printf("\nInitialising Winsock...");
    if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
    {
        printf("Failed. Error Code : %d",WSAGetLastError());
        return 1;
    }

    printf("Initialised.\n");


    if ( (he = gethostbyname( hostname ) ) == NULL)
    {
        //gethostbyname failed
        printf("gethostbyname failed : %d" , WSAGetLastError());
        return 1;
    }

    //Cast the h_addr_list to in_addr , since h_addr_list also has the ip address in l
    addr_list = (struct in_addr **) he->h_addr_list;

    for(i = 0; addr_list[i] != NULL; i++)
    {
        //Return the first one;
        strcpy(ip , inet_ntoa(*addr_list[i]) );
    }

    printf("%s resolved to : %s\n" , hostname , ip);
    return 0;
    return 0;
}
```

Output of the code would look like :

```
www.google.com resolved to : 74.125.235.20
```

So the above code can be used to find the ip address of any domain name. Then the ip address can be used to make a connection using a socket.

Function `inet_ntoa` will convert an IP address in long format to dotted format. This is just the opposite of `inet_addr`.

So far we have see some important structures that are used. Lets revise them :

1. `sockaddr_in` - Connection information. Used by connect , send , recv etc.
2. `in_addr` - Ip address in long format
3. `sockaddr`
4. `hostent` - The ip addresses of a hostname. Used by gethostbyname

Server Concepts

OK now onto server things. Servers basically do the following :

1. Open a socket
2. Bind to a address(and port).
3. Listen for incoming connections.
4. Accept connections
5. Read/Send

We have already learnt how to open a socket. So the next thing would be to bind it.

Bind a socket

Function `bind` can be used to bind a socket to a particular address and port. It needs a sockaddr_in structure similar to connect function.

Lets see a code example :

```
/*
    Bind socket to port 8888 on localhost
*/

#include<stdio.h>
#include<winsock2.h>

#pragma comment(lib,"ws2_32.lib") //Winsock Library

int main(int argc , char *argv[])
{
    WSADATA wsa;
    SOCKET s;
    struct sockaddr_in server;

    printf("\nInitialising Winsock...");
    if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
    {
        printf("Failed. Error Code : %d",WSAGetLastError());
        return 1;
    }

    printf("Initialised.\n");

    //Create a socket
    if((s = socket(AF_INET , SOCK_STREAM , 0 )) == INVALID_SOCKET)
    {
        printf("Could not create socket : %d" , WSAGetLastError());
    }

    printf("Socket created.\n");

    //Prepare the sockaddr_in structure
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons( 8888 );

    //Bind
    if( bind(s ,(struct sockaddr *)&server , sizeof(server)) == SOCKET_ERROR)
    {
        printf("Bind failed with error code : %d" , WSAGetLastError());
    }

    puts("Bind done");

    closesocket(s);

    return 0;
}
```

Now that bind is done, its time to make the socket listen to connections. We bind a socket to a particular IP address and a certain port number. By doing this we ensure that all incoming data which is directed towards this port number is received by this application.

This makes it obvious that you cannot have 2 sockets bound to the same port.

Listen for connections

After binding a socket to a port the next thing we need to do is listen for connections. For this we need to put the socket in listening mode. Function `listen` is used to put the socket in listening mode. Just add the following line after bind.

```
//Listen
listen(s , 3);
```

Thats all. Now comes the main part of accepting new connections.

Accept connection

Function `accept` is used for this. Here is the code

```
/*
    Bind socket to port 8888 on localhost
```

```
*/

#include<stdio.h>
#include<winsock2.h>

#pragma comment(lib,"ws2_32.lib") //Winsock Library

int main(int argc , char *argv[])
{
    WSADATA wsa;
    SOCKET s , new_socket;
    struct sockaddr_in server , client;
    int c;

    printf("\nInitialising Winsock...");
    if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
    {
        printf("Failed. Error Code : %d",WSAGetLastError());
        return 1;
    }

    printf("Initialised.\n");

    //Create a socket
    if((s = socket(AF_INET , SOCK_STREAM , 0 )) == INVALID_SOCKET)
    {
        printf("Could not create socket : %d" , WSAGetLastError());
    }

    printf("Socket created.\n");

    //Prepare the sockaddr_in structure
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons( 8888 );

    //Bind
    if( bind(s ,(struct sockaddr *)&server , sizeof(server)) == SOCKET_ERROR)
    {
        printf("Bind failed with error code : %d" , WSAGetLastError());
    }

    puts("Bind done");


    //Listen to incoming connections
    listen(s , 3);

    //Accept and incoming connection
    puts("Waiting for incoming connections...");

    c = sizeof(struct sockaddr_in);
    new_socket = accept(s , (struct sockaddr *)&client, &c);
    if (new_socket == INVALID_SOCKET)
    {
        printf("accept failed with error code : %d" , WSAGetLastError());
    }

    puts("Connection accepted");

    closesocket(s);
    WSACleanup();

    return 0;
}
```

Output

Run the program. It should show

```
Initialising Winsock...Initialised.
Socket created.
Bind done
Waiting for incoming connections...
```

So now this program is waiting for incoming connections on port 8888. Dont close this program , keep it running. Now a client can connect to it on this port. We shall use the telnet client for testing this. Open a terminal and type

```
telnet localhost 8888
```

And the server output will show

```
Initialising Winsock...Initialised.
Socket created.
Bind done
Waiting for incoming connections...
Connection accepted
Press any key to continue
```

So we can see that the client connected to the server. Try the above process till you get it perfect.

Note

You can get the ip address of client and the port of connection by using the sockaddr_in structure passed to accept function. It is very simple :

```
char *client_ip = inet_ntoa(client.sin_addr);
int client_port = ntohs(client.sin_port);
```

We accepted an incoming connection but closed it immediately. This was not very productive. There are lots of things that can be done after an incoming connection is established. Afterall the connection was established for the purpose of communication. So lets reply to the client.

Here is an example :

```
/*
    Bind socket to port 8888 on localhost
*/
#include<io.h>
#include<stdio.h>
#include<winsock2.h>

#pragma comment(lib,"ws2_32.lib") //Winsock Library

int main(int argc , char *argv[])
{
    WSADATA wsa;
    SOCKET s , new_socket;
    struct sockaddr_in server , client;
    int c;
    char *message;

    printf("\nInitialising Winsock...");
    if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
    {
        printf("Failed. Error Code : %d",WSAGetLastError());
        return 1;
    }

    printf("Initialised.\n");

    //Create a socket
    if((s = socket(AF_INET , SOCK_STREAM , 0 )) == INVALID_SOCKET)
    {
        printf("Could not create socket : %d" , WSAGetLastError());
    }

    printf("Socket created.\n");

    //Prepare the sockaddr_in structure
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons( 8888 );

    //Bind
    if( bind(s ,(struct sockaddr *)&server , sizeof(server)) == SOCKET_ERROR)
    {
        printf("Bind failed with error code : %d" , WSAGetLastError());
    }

    puts("Bind done");

    //Listen to incoming connections
    listen(s , 3);

    //Accept and incoming connection
    puts("Waiting for incoming connections...");

    c = sizeof(struct sockaddr_in);
    new_socket = accept(s , (struct sockaddr *)&client, &c);
    if (new_socket == INVALID_SOCKET)
    {
        printf("accept failed with error code : %d" , WSAGetLastError());
    }

    puts("Connection accepted");

    //Reply to client
    message = "Hello Client , I have received your connection. But I have to go now, b
    send(new_socket , message , strlen(message) , 0);

    getchar();

    closesocket(s);
    WSACleanup();

    return 0;
}
```

Run the above code in 1 terminal. And connect to this server using telnet from another terminal and you should see this :

```
Hello Client , I have received your connection. But I have to go now, bye
```

So the client(telnet) received a reply from server. We had to use a getchar because otherwise the output would scroll out of the client terminal without waiting

We can see that the connection is closed immediately after that simply because the server program ends after accepting and sending reply. A server like www.google.com is always up to accept incoming connections.

It means that a server is supposed to be running all the time. Afterall its a server meant to serve. So we need to keep our server RUNNING non-stop. The simplest way to do this is to put the `accept` in a loop so that it can receive incoming connections all the time.

Live Server

So a live server will be alive for all time. Lets code this up :

```c
/*
    Live Server on port 8888
*/
#include<io.h>
#include<stdio.h>
#include<winsock2.h>

#pragma comment(lib,"ws2_32.lib") //Winsock Library

int main(int argc , char *argv[])
{
    WSADATA wsa;
    SOCKET s , new_socket;
    struct sockaddr_in server , client;
    int c;
    char *message;

    printf("\nInitialising Winsock...");
    if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
    {
        printf("Failed. Error Code : %d",WSAGetLastError());
        return 1;
    }

    printf("Initialised.\n");

    //Create a socket
    if((s = socket(AF_INET , SOCK_STREAM , 0 )) == INVALID_SOCKET)
    {
        printf("Could not create socket : %d" , WSAGetLastError());
    }

    printf("Socket created.\n");

    //Prepare the sockaddr_in structure
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons( 8888 );

    //Bind
    if( bind(s ,(struct sockaddr *)&server , sizeof(server)) == SOCKET_ERROR)
    {
        printf("Bind failed with error code : %d" , WSAGetLastError());
        exit(EXIT_FAILURE);
    }

    puts("Bind done");

    //Listen to incoming connections
    listen(s , 3);

    //Accept and incoming connection
    puts("Waiting for incoming connections...");

    c = sizeof(struct sockaddr_in);

    while( (new_socket = accept(s , (struct sockaddr *)&client, &c)) != INVALID_SOCKET
    {
        puts("Connection accepted");

        //Reply to the client
        message = "Hello Client , I have received your connection. But I have to go no
        send(new_socket , message , strlen(message) , 0);
    }

    if (new_socket == INVALID_SOCKET)
    {
        printf("accept failed with error code : %d" , WSAGetLastError());
```

```
        return 1;
    }

    closesocket(s);
    WSACleanup();

    return 0;
}
```

We havent done a lot there. Just the accept was put in a loop.

Now run the program in 1 terminal , and open 3 other terminals. From each of the 3 terminal do a telnet to the server port.

Run telnet like this

```
C:\>telnet
```

```
Welcome to Microsoft Telnet Client
Escape Character is 'CTRL+]'
Microsoft Telnet> open localhost 8888
```

```
Hello Client , I have received your connection. But I have to go now, bye
```

And the server terminal would show

```
Initialising Winsock...Initialised.
Socket created.
Bind done
Waiting for incoming connections...
Connection accepted
Connection accepted
```

So now the server is running nonstop and the telnet terminals are also connected nonstop. Now close the server program.
All telnet terminals would show "Connection to host lost."
Good so far. But still there is not effective communication between the server and the client.

The server program accepts connections in a loop and just send them a reply, after that it does nothing with them. Also it is not able to handle more than 1 connection at a time. So now its time to handle the connections , and handle multiple connections together.

Handling Connections

To handle every connection we need a separate handling code to run along with the main server accepting connections.
One way to achieve this is using threads. The main server program accepts a connection and creates a new thread to handle communication for the connection, and then the server goes back to accept more connections.

We shall now use threads to create handlers for each connection the server accepts. Lets do it pal.

```

```

Run the above server and open 3 terminals like before. Now the server will create a thread for each client connecting to it.

The telnet terminals would show :

```

```

This one looks good , but the communication handler is also quite dumb. After the greeting it terminates. It should stay alive and keep communicating with the client.

One way to do this is by making the connection handler wait for some message from a client as long as the client is connected. If the client disconnects , the connection handler ends.

So the connection handler can be rewritten like this :

```

```

The above connection handler takes some input from the client and replies back with the same. Simple! Here is how the telnet output might look

```
1 |
```

So now we have a server thats communicative. Thats useful now.

Conclusion

The winsock api is quite similar to Linux sockets in terms of function name and structures. Few differences exist like :

1. Winsock needs to be initialised with the WSAStartup function. No such thing in linux.

2. Header file names are different. Winsock needs winsock2.h , whereas Linux needs socket.h , apra/inet.h , unistd.h and many others.

3. Winsock function to close a socket is `closesocket` , whereas on Linux it is `close`.
On Winsock WSACleanup must also be called to unload the winsock dll.

4. On winsock the error number is fetched by the function `WSAGetLastError()`. On Linux the errno variable from errno.h file is filled with the error number.

And there are many more differences as we go deep.

By now you must have learned the basics of socket programming in C. You can try out some experiments like writing a chat client or something similar.

If you think that the tutorial needs some addons or improvements or any of the code snippets above dont work then feel free to make a comment below so that it gets fixed.

Last Updated On : 12th December 2012

socket programming winsock winsock tutorial

Related Posts

UDP socket programming in winsock

Raw socket programming on windows with winsock

Socket programming in C on Linux – tutorial

About **Silver Moon**

Php developer, blogger and Linux enthusiast. He can be reached at binarytides@gmail.com. Or find him on Google+

51 Comments

+ ADD COMMENT

shameer January 12, 2017 at 2:37 pm

Hi,

it was a good tutorial for understanding C++ sockets for Beginners.

when i run the server and client codes,in server side the connection is accepted but in the client side it is connection error…

```
if (connect(s, (struct sockaddr *)&server, sizeof(server)) < 0)
{
puts("connect error");
return 1;
}
```

Soumyaranjan Biswal December 23, 2016 at 8:19 am

Really Awesome….
I spent one week to search winsock programming source code.Finally I got this.Thanks a lot Silver Moon.
Really U R Silver Moon.
Request :Now I want to create win32 DLL application.If you have Please forward me .

Thanks Again….

Manick October 17, 2016 at 4:28 pm

Hi You were awesome. You were great. You were fabulous. So long to understand these stuffs and finally everything works like a miracle. Thanks a ton for this great article. Great buddy. Keep up the good work.

Totzfreelance June 1, 2016 at 7:58 pm

Hi, I am a noob. I got this error, I compile your code until 'receiving data', what is the problem here Silver Moon:
C:\Users\Admin\Desktop>gcc server4.c -o server4.exe
C:\Users\Admin\AppData\Local\Temp\ccCq2ONN.o:server4.c:(.text+0x37): undefined reference to `WSAStartup@8'
C:\Users\Admin\AppData\Local\Temp\ccCq2ONN.o:server4.c:(.text+0x43): undefined reference to `WSAGetLastError@0'
C:\Users\Admin\AppData\Local\Temp\ccCq2ONN.o:server4.c:(.text+0x85): undefined reference to `socket@12'
C:\Users\Admin\AppData\Local\Temp\ccCq2ONN.o:server4.c:(.text+0x96): undefined reference to `WSAGetLastError@0'
C:\Users\Admin\AppData\Local\Temp\ccCq2ONN.o:server4.c:(.text+0xbe): undefined reference to `inet_addr@4'
C:\Users\Admin\AppData\Local\Temp\ccCq2ONN.o:server4.c:(.text+0xdc): undefined reference to `htons@4'
C:\Users\Admin\AppData\Local\Temp\ccCq2ONN.o:server4.c:(.text+0x103): undefined reference to `connect@12'
C:\Users\Admin\AppData\Local\Temp\ccCq2ONN.o:server4.c:(.text+0x15c): undefined reference to `send@16'
C:\Users\Admin\AppData\Local\Temp\ccCq2ONN.o:server4.c:(.text+0x1a7): undefined reference to `recv@16'
collect2.exe: error: ld returned 1 exit status

Jan February 18, 2017 at 11:24 pm

You need to find and add libwsock32.a library to your compiler's linkers if your using gcc based ide or gcc. you also can add ws2_32.dll if you use windows and installed visual studio.

salim said January 2, 2016 at 5:29 pm

Very good article, but i would like some clarification about something. is it possible to create a socket service and connect to a remote mysql database through it ?

am writing an application for a Point of Sale terminal and it doesn't support the mysql api directly, all web functions have to be done using a socket.

Lema June 3, 2015 at 12:22 am

HI, code example from "Handling Connections" has disappeared. Please correct.

sacha March 10, 2015 at 4:50 am

Can you use Port 22 or SSH using this?

Carlos February 13, 2015 at 1:23 pm

Hi there! First of all thanks for this great code :)
I have a little problem here, for some reason, I cannot see the Handling Connections instructions… Can this turorial be downloaded in pdf format for example? Thanks a lot!

Rajneesh Barnwal January 20, 2015 at 11:23 am

i'm very new to socket programing. i'm using turbo c compiler. where could I get the required headers for socket programing they were not their with turbo c.plz help me out

Ben January 14, 2015 at 3:04 pm

really great tutorial, right up untill all the exaple code is missing in the last section (everything after "Handling Connections" headder) Anyone else having the same problem?

Parshanth November 27, 2014 at 1:22 pm

hi dude, whlie compiling the above code through Dev-C++, i m getting a error as, source file is not compiled, will you please suggest me how to fix it…….?

Thanks and Regards,
Prashanth

daiwat November 10, 2014 at 9:56 am

connection failed error 10060…. while executing on two different machines connected via lan cable

grateful October 1, 2014 at 1:29 am

handling connection part is broken :(

Dallas July 15, 2014 at 9:46 pm

fyi–
server_reply[recv_size] = '';
should be
server_reply[recv_size-1] = '';
since C/C++ is 0-indexed

Thom Newton August 11, 2014 at 6:36 pm

No It shouldn't. If you did this you would overwrite the last character of the string.

AviHD June 28, 2014 at 12:04 pm

How about adding some snippets to the above on choosing between different IP/interfaces available on the system to listen and send on?

oskar June 20, 2014 at 8:41 pm

nice tutorial i just have one problem! i can connect to the server with the telnet client. but its not possible with the client you programmed in C why!! have to get this to work !

Dallas July 15, 2014 at 9:43 pm

You have to change the client IP to 127.0.0.1

FRED L May 3, 2014 at 7:18 am

i already configured it until "Connect to a Server", i try to understand every configuration, but i'm trying to compile and i got this error:

C:Dev-CppSockmain winsock.cpp In function ĩnt main(int, char**)':
90 C:Dev-CppSockmain winsock.cpp cannot convert m̀ain(int, char**)::sockaddr*' to c̀onst sockaddr*' for argument 2' to ĩnt connect(SOCKET, const sockaddr*, int)'

max January 31, 2014 at 3:14 pm

hi, i get err 10060 on connect() function/ client side

Jano Crema November 7, 2013 at 4:36 pm

thank you very much, Great!!!

Astrid September 17, 2013 at 5:32 am

The best tutorial!! thank… :)

sushil September 4, 2013 at 1:42 am

fantastic and the best explanation of windows socket creation…thanks a lot for such a wonderful input on sockets. However the code for the thread section for multiple client handling is missing…could you please put the code…also if i would like to have the code for sending a file such as a log file at a time interval.

Jacop August 27, 2013 at 12:26 am

handling connections part is broken i think.(if u can repair it can be good :) this topic was a good tutorial for understanding how to use sockets)

otaigbe stanley August 18, 2013 at 2:33 pm

i 'm trying to compile and test run your code i keep getting this errors

undefined reference to _imp_WSAStartup@8 and undefined reference to _imp_WSACloseup@0.
i think it is as a result of compilation error.please i need in detail how to compile winsock codes in c.

Aniello Saggese May 22, 2014 at 6:38 pm

In case of Dev-C++ this problem can be fixed by putting under the menu "Tools" on the "Compiler Options" in the frame "Add the following commands when calling the linker:" the following string: " -static-libgcc -lws2_32 " and it should work.

SuperInvitado July 8, 2013 at 2:33 am

excellent tutorial, it helped me very much, thank you Silver Moon

fahad May 29, 2013 at 2:01 pm

Greetings

on typing this line from heading "Recieving data" line 41 it gives error under "(struct sockaddr *)&server"

intelliSense: argument of type "sockaddr *" is incompatible with parameter of type "const sockaddr *"

can u plz tell how it can be resolved?

fahad May 27, 2013 at 9:57 pm

what if i have to send data from linux box to windows box can i use the client side code of linux socket programming and server side of windows socket programing?????or there are some special changes that are needed to be done??? sorry if u find my question lame. i m complete beginer

Silver Moon May 27, 2013 at 10:02 pm

yes, you are correct. It would work like that.

the code should work without any changes, just that the client needs to connect on the correct port which the server has opened.

fahad May 27, 2013 at 10:04 pm

woww u r fast :)
thnks

tarun March 7, 2013 at 8:35 pm

but all this tutorial does not tell about how can we send some text to server..when we run your code it simply say connection created and output is “Hello Client , I have received your connection. But I have to go now, bye.
but what if we want to chat with server or want to send some message?

Silver Moon March 28, 2013 at 5:42 pm

check this post on writing tcp socket server in winsock
http://www.binarytides.com/code-tcp-socket-server-winsock/

Barn February 7, 2013 at 12:11 pm

Are you sure that you can call WSAGetLastError if WSAStartup fails? From what I remember it loads the winsock dll, so if it isnt loaded, WSAGetLastError wouldn’t be available.

Silver Moon March 27, 2013 at 1:58 pm

It would be available even if WSAStartup fails. the documentation says :

“TheWSAGetLastError function is one of the only functions in the Winsock 2.2 DLL that can be called in the case of aWSAStartup failure.”

http://msdn.microsoft.com/en-us/library/windows/desktop/ms741580(v=vs.85).aspx

Mark December 20, 2012 at 8:49 pm

Great beginner friendly tutorial! Thanks.

Handling Connections section has no code content?

Silver Moon March 28, 2013 at 5:39 pm

check this post

http://www.binarytides.com/code-tcp-socket-server-winsock/

Gopher October 21, 2013 at 9:15 am

But that post does not cover threads. Please fill in the above blanks with code. The parts before “Handling Connections” was excellent.

Thomas Gray November 25, 2012 at 3:52 pm

Great tutorial !

Well explained , clear and concise.

I would greatly appreciate it if you could upload the code snippets related to handling connections using threads.

Thanks !

Silver Moon March 28, 2013 at 5:42 pm

connections can be handled with the select function. check this post
http://www.binarytides.com/code-tcp-socket-server-winsock/

Yogesh November 22, 2012 at 7:44 pm

Can you suggest good books(with examples) for winsock?

Silver Moon November 22, 2012 at 7:58 pm

I would suggest learning the winsock basics from google/internet.
there are plenty of sites and tutorials out there.
then try building a real socket application and the process of development shall teach you more.

Yogesh November 22, 2012 at 10:42 am

Hello, This is a great tutorial for beginners like me. You have kept it SSS – short, sweet, simple :).
But I am facing one problem – I am not able to connect to a remote server. I have used your program as it is, except server ip address. Socket is getting created but 'connect error' is coming. Do I need to change the port number? I tried with different port numbers but in vain. I have run this program from my company. Is company's firewall creating problem?
Please guide me to run this program
I

Silver Moon November 22, 2012 at 10:50 am

you can connect to a remote ip+port only if the port is open. So first check if the port is really open or not
This can be done by running the following command in the terminal
telnet 1.2.3.4 900
where 1.2.3.4 is the remote ip address and 900 is the port. If telnet shows the port as open, then the socket program shown in this post should also connect easily. If telnet fails, then the socket program will also fail since they both are doing the same thing.

Yogesh November 22, 2012 at 11:54 am

Hello,
Thanks for quick reply. I followed your instructions, the google port is not open.
But I want to see the successful connection by any means :). Can I connect to own computer? Or there is another way to achieve this?

Silver Moon November 22, 2012 at 11:56 am

you can do a ping http://www.google.com in your terminal, it will give you the ip address of google.com
then try connecting to port 80 of that ip. it should work.

Neel October 17, 2012 at 12:17 am

You have given great knowledge buddy. I was searching for page like this for more than 2 weeks, I extend you my heartiest appreciation.

Luis August 16, 2012 at 1:00 am

Dude this is the best noob-friendly tutorial so far!! . Thanks !

Mark Whitney July 25, 2012 at 12:58 am

Great tutorial, but where are the code examples for the threads? I just see empty boxes in the Handling Connections section.

Silver Moon March 28, 2013 at 5:41 pm

check this post

http://www.binarytides.com/code-tcp-socket-server-winsock/

it shows how to handle multiple socket connections using select function

Leave a comment

Name (required)

Mail (will not be published) (required)

Website

Comment

POST COMMENT