Socket programming in C on Linux – tutorial

C By Silver Moon On Dec 24, 2011 115 Comments Like 128 G+1 50 Tweet 27

TCP/IP socket programming in C

This is a quick tutorial on **socket programming in c** language on a Linux system. "Linux" because the code snippets shown over here will work only on a Linux system and not on Windows. The windows api to socket programming is called winsock and we shall go through it in another tutorial.

Sockets are the "virtual" endpoints of any kind of network communications done between 2 hosts over in a network. 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.

The socket api on linux is similar to bsd/unix sockets from which it has evolved. Although over time the api has become slightly different at few places. And now the newer official standard is posix sockets api which is same as bsd sockets.

This tutorial assumes that you have basic knowledge of C and pointers. You will need to have **gcc compiler** installed on your **Linux system**. An IDE along with gcc would be great. I would recommend geany as you can quickly edit and run single file programs in it without much configurations. On ubuntu you can do a sudo apt-get install geany on the terminal.

All along the tutorial there are code snippets to demonstrate some concepts. You can run those code snippets in geany rightaway and test the results to better understand the concepts.

Connect with us

Other interesting stuff

Code a port scanner in C | linu

SYN Flood DOS Attack with C Code (Linux)

Code a simple socket server in

Programming udp sockets in C Linux

C program to get a domain's wl information using sockets on Li

Raw socket programming on w with Winpcap

Create a socket

This first thing to do is create a socket. The **socket** function does this.
Here is a code sample :

```
#include<stdio.h>
#include<sys/socket.h>

int main(int argc , char *argv[])
{
    int socket_desc;
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);

    if (socket_desc == -1)
    {
        printf("Could not create socket");
    }

    return 0;
}
```

Function socket() creates a socket and returns a descriptor which can be used in other functions. The above code will create a socket with following properties

Address Family - AF_INET (this is IP version 4)
Type - SOCK_STREAM (this means connection oriented TCP protocol)
Protocol - 0 [or IPPROTO_IP This is IP protocol]

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 socket 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 to create a sockaddr_in structure with proper values.

```
struct sockaddr_in server;
```

Have a look at the structure

```
// 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
};

struct in_addr {
    unsigned long s_addr;          // load with inet_pton()
};

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.

```
#include<stdio.h>
#include<sys/socket.h>
#include<arpa/inet.h> //inet_addr

int main(int argc , char *argv[])
{
    int socket_desc;
    struct sockaddr_in server;

    //Create socket
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1)
    {
        printf("Could not create socket");
    }

    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(socket_desc , (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.

Connections are present only in tcp sockets

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.

Send data over socket

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 :

```c
#include<stdio.h>
#include<string.h>    //strlen
#include<sys/socket.h>
#include<arpa/inet.h> //inet_addr

int main(int argc , char *argv[])
{
    int socket_desc;
    struct sockaddr_in server;
    char *message;

    //Create socket
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1)
    {
        printf("Could not create socket");
    }

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

    puts("Connected\n");

    //Send some data
    message = "GET / HTTP/1.1\r\n\r\n";
    if( send(socket_desc , 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.

Note

When sending data to a socket you are basically writing data to that socket. This is similar to writing data to a file. Hence you can also use the `write` function to send data to a socket. Later in this tutorial we shall use write function to send data.

Receive data on socket

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.

```c
#include<stdio.h>
#include<string.h>    //strlen
#include<sys/socket.h>
#include<arpa/inet.h> //inet_addr

int main(int argc , char *argv[])
{
    int socket_desc;
    struct sockaddr_in server;
    char *message , server_reply[2000];

    //Create socket
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1)
    {
        printf("Could not create socket");
    }

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

    puts("Connected\n");

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

    //Receive a reply from the server
    if( recv(socket_desc, server_reply , 2000 , 0) < 0)
    {
        puts("recv failed");
    }
    puts("Reply received\n");
    puts(server_reply);

    return 0;
}
```

Here is the output of the above code :

```
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=0edd21a16f0db219:FF=0:TM=1324644706:LM=1324644706:S=z6hDC9cZfGEowv_o; expires=
Date: Fri, 23 Dec 2011 12:51:46 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>
```

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!

Note

When receiving data on a socket , we are basically reading the data on the socket. This is similar to reading data from a file. So we can also use the `read` function to read data on a socket. For example :

```
read(socket_desc, server_reply , 2000);
```

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

Close socket

Function `close` is used to close the socket. Need to include the unistd.h header file for this.

```
close(socket_desc);
```

Thats it.

Summary

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

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

The other kind of socket application is called a **socket 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 hostname

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.

```
#include<stdio.h> //printf
#include<string.h> //strcpy
#include<sys/socket.h>
#include<netdb.h> //hostent
#include<arpa/inet.h>

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

    if ( (he = gethostbyname( hostname ) ) == NULL)
    {
        //gethostbyname failed
        herror("gethostbyname");
        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" , hostname , ip);
    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

In the next part we shall look into creating servers using socket. Servers are the opposite of clients, that instead of connecting out to others, they wait for incoming connections.

Socket server

OK now onto server things. Socket servers operate in the following manner

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 socket to a port

The bind function can be used to bind a socket to a particular "address and port" combination. It needs a sockaddr_in structure similar to connect function.

```
int socket_desc;
struct sockaddr_in server;

//Create socket
socket_desc = socket(AF_INET , SOCK_STREAM , 0);
if (socket_desc == -1)
{
    printf("Could not create socket");
}

//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(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0)
{
    puts("bind failed");
}
puts("bind done");
```

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 incoming connections on the socket

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(socket_desc , 3);
```

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

Accept connection

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

```
#include<stdio.h>
#include<sys/socket.h>
#include<arpa/inet.h> //inet_addr

int main(int argc , char *argv[])
{
    int socket_desc , new_socket , c;
    struct sockaddr_in server , client;
    
    //Create socket
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1)
    {
        printf("Could not create socket");
    }
    
    //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(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0)
    {
        puts("bind failed");
    }
    puts("bind done");
    
    //Listen
    listen(socket_desc , 3);
    
    //Accept and incoming connection
    puts("Waiting for incoming connections...");
    c = sizeof(struct sockaddr_in);
    new_socket = accept(socket_desc, (struct sockaddr *)&client, (socklen_t*)&c);
    if (new_socket<0)
    {
        perror("accept failed");
    }
    
    puts("Connection accepted");
    
    return 0;
}
```

Program output

Run the program. It should show

```
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
```

On the terminal you shall get

```
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Connection closed by foreign host.
```

And the server output will show

```
bind done
Waiting for incoming connections...
Connection accepted
```

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

Get the ip address of the connected client

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.

We can simply use the `write` function to write something to the socket of the incoming connection and the client should see it. Here is an example :

```
#include<stdio.h>
#include<string.h>    //strlen
#include<sys/socket.h>
#include<arpa/inet.h> //inet_addr
#include<unistd.h>    //write

int main(int argc , char *argv[])
{
    int socket_desc , new_socket , c;
    struct sockaddr_in server , client;
    char *message;
    
    //Create socket
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1)
    {
        printf("Could not create socket");
    }
    
    //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(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0)
    {
        puts("bind failed");
        return 1;
    }
    puts("bind done");
    
    //Listen
    listen(socket_desc , 3);
    
    //Accept and incoming connection
    puts("Waiting for incoming connections...");
    c = sizeof(struct sockaddr_in);
    new_socket = accept(socket_desc, (struct sockaddr *)&client, (socklen_t*)&c);
    if (new_socket<0)
    {
        perror("accept failed");
        return 1;
    }
    
    puts("Connection accepted");
    
    //Reply to the client
    message = "Hello Client , I have received your connection. But I have to go now, b
    write(new_socket , message , strlen(message));
    
    return 0;
}
```

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

```
$ telnet localhost 8888
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Hello Client , I have received your connection. But I have to go now, bye
Connection closed by foreign host.
```

So the client(telnet) received a reply from server.

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 :

```
#include<stdio.h>
#include<string.h>    //strlen
#include<sys/socket.h>
#include<arpa/inet.h> //inet_addr
#include<unistd.h>    //write

int main(int argc , char *argv[])
{
    int socket_desc , new_socket , c;
    struct sockaddr_in server , client;
    char *message;
    
    //Create socket
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1)
    {
        printf("Could not create socket");
    }
    
    //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(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0)
    {
        puts("bind failed");
        return 1;
    }
    puts("bind done");

    //Listen
    listen(socket_desc , 3);

    //Accept and incoming connection
    puts("Waiting for incoming connections...");
    c = sizeof(struct sockaddr_in);
    while( (new_socket = accept(socket_desc, (struct sockaddr *)&client, (socklen_t*)&
    {
        puts("Connection accepted");

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

    if (new_socket<0)
    {
        perror("accept failed");
        return 1;
    }

    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.

Each of the telnet terminal would show :

```
$ telnet localhost 8888
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Hello Client , I have received your connection. But I have to go now, bye
```

And the server terminal would show

```
bind done
Waiting for incoming connections...
Connection accepted
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 closed by foreign host."
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.

Handle multiple socket connections with threads

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.

On Linux threading can be done with the pthread (posix threads) library. It would be good to read some small tutorial about it if you dont know anything about it. However the usage is not very complicated.

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

```
#include<stdio.h>
#include<string.h>    //strlen
#include<stdlib.h>    //strlen
#include<sys/socket.h>
#include<arpa/inet.h> //inet_addr
#include<unistd.h>    //write

#include<pthread.h> //for threading , link with lpthread

void *connection_handler(void *);

int main(int argc , char *argv[])
{
    int socket_desc , new_socket , c , *new_sock;
    struct sockaddr_in server , client;
    char *message;
    //Create socket
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1)
    {
        printf("Could not create socket");
    }
     
    //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(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0)
    {
        puts("bind failed");
        return 1;
    }
    puts("bind done");
     
    //Listen
    listen(socket_desc , 3);
     
    //Accept and incoming connection
    puts("Waiting for incoming connections...");
    c = sizeof(struct sockaddr_in);
    while( (new_socket = accept(socket_desc, (struct sockaddr *)&client, (socklen_t*)&
    {
        puts("Connection accepted");
         
        //Reply to the client
        message = "Hello Client , I have received your connection. And now I will assi
        write(new_socket , message , strlen(message));
         
        pthread_t sniffer_thread;
        new_sock = malloc(1);
        *new_sock = new_socket;
         
        if( pthread_create( &sniffer_thread , NULL ,  connection_handler , (void*) new_
        {
            perror("could not create thread");
            return 1;
        }
         
        //Now join the thread , so that we dont terminate before the thread
        //pthread_join( sniffer_thread , NULL);
        puts("Handler assigned");
    }
     
    if (new_socket<0)
    {
        perror("accept failed");
        return 1;
    }
     
    return 0;
}
 
/*
 * This will handle connection for each client
 * */
void *connection_handler(void *socket_desc)
{
    //Get the socket descriptor
    int sock = *(int*)socket_desc;
     
    char *message;
     
    //Send some messages to the client
    message = "Greetings! I am your connection handler\n";
    write(sock , message , strlen(message));
     
    message = "Its my duty to communicate with you";
    write(sock , message , strlen(message));
     
    //Free the socket pointer
    free(socket_desc);
     
    return 0;
}
```

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 :

```
$ telnet localhost 8888
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Hello Client , I have received your connection. And now I will assign a handler for you
Hello I am your connection handler
Its my duty to communicate with you
```

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 :

```c
/*
 * This will handle connection for each client
 * */
void *connection_handler(void *socket_desc)
{
    //Get the socket descriptor
    int sock = *(int*)socket_desc;
    int read_size;
    char *message , client_message[2000];

    //Send some messages to the client
    message = "Greetings! I am your connection handler\n";
    write(sock , message , strlen(message));

    message = "Now type something and i shall repeat what you type \n";
    write(sock , message , strlen(message));

    //Receive a message from client
    while( (read_size = recv(sock , client_message , 2000 , 0)) > 0 )
    {
        //Send the message back to client
        write(sock , client_message , strlen(client_message));
    }

    if(read_size == 0)
    {
        puts("Client disconnected");
        fflush(stdout);
    }
    else if(read_size == -1)
    {
        perror("recv failed");
    }

    //Free the socket pointer
    free(socket_desc);

    return 0;
}
```

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

```
$ telnet localhost 8888
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Hello Client , I have received your connection. And now I will assign a handler for you
Greetings! I am your connection handler
Now type something and i shall repeat what you type
Hello
Hello
How are you
How are you
I am fine
I am fine
```

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

Linking the pthread library

When compiling programs that use the pthread library you need to link the library. This is done like this :

```
$ gcc program.c -lpthread
```

Conclusion

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.

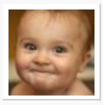
If you think that the tutorial needs some addons or improvements or any of the code snippets above dont work then feel free to make a comment below so that it gets fixed.

Last Updated On : 3rd August 2013

c sockets linux network programming socket programming unix sockets

Related Posts

- Winsock tutorial – Socket programming in C on windows
- Perl socket programming tutorial
- Python socket – network programming tutorial
- PHP Socket programming tutorial
- UDP socket programming in winsock



About **Silver Moon**

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

115 Comments BinaryTides 1 Login

Recommend 4 Share Sort by Best

Join the discussion…

Loco Pepe © • 2 years ago
Oh nice I understood everything! But I have a question: As long as I'm treating each client with a different thread, how can I make them write something then and show the message to any other client connected? (Basically a chatroom for every client) Don't know how to do this I'm new at socket programming.
24 • Reply • Share ›

frustrated student • a year ago
best tutorial i've encountered.... 1 regular visitor added
2 • Reply • Share ›

blueMix • 2 years ago
Nice and neat tutorial.
Thanks a lot.
1 • Reply • Share ›

Joopy • 2 years ago
Nice tutorial!!! Thank you
1 • Reply • Share ›

fishguy • 2 years ago
some really useful stuff here. Thanks !
1 • Reply • Share ›

Gaurav Agarwal • 2 years ago
Excellent tutorial....thank you...:)
1 • Reply • Share ›

IBRA • 3 years ago
how to implement firewall in linux....i dont wanna use iptables....??any good links or tutorials...????
1 • Reply • Share ›

> **Leo** → IBRA • a year ago
> http://techarena51.com/index.p...
> Reply • Share ›

Anjali • 4 months ago
what changes do i have to do to make it run in ipv6
Reply • Share ›

Isuru • 4 months ago
Thank you!
Reply • Share ›

Timur Kalandarov • 5 months ago
Nice article, thx!

Reply • Share ›

Sandeep Singh Rana • 6 months ago

Best tutorial for me...
but i found a problem on my machine
recv() function is not working properly for me....
-------Output Start--------
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Rana is here;Greetings : I am your message handler.
It's my duty to communicate with you.
Hello
Hello
Another message
Another message
problem
problem
essage
-------end--------
as i send a small message just after a large message it prints out the message with some concatenated text from last message sent

Reply • Share ›

Long • 7 months ago

How to run socket in linux? Help me?

Reply • Share ›

DeC • 7 months ago

Great tutorial!

but I want to point out that recv() does not null terminate the buffer, so the line

//Send the message back to client
write(sock , client_message , strlen(client_message));

in the last code should be

//Send the message back to client
write(sock , client_message , read_size);

thank you very much

Reply • Share ›

craig • 9 months ago

I had to change my message to "GET / HTTP/1.1\r\nHost: http://localhost.localdomain:8080\r\n\r\n" to get it to work.

Reply • Share ›

VMAN • 9 months ago

Does the GET
message = "GET / HTTP/1.1\r\n\r\n";
really have the correct format?
Getting a message failure when trying to send it to a server.

Reply • Share ›

Lawrence Tsang • 9 months ago

Nice tutorial for a Linux C socket new comer. Thanks.

Reply • Share ›

Roman Solomakhov • 10 months ago

Superb tutorial! Thank you very much!

Reply • Share ›

Arqam raza khan • 10 months ago

I am searching for c code in Linux. Where client send message which may be text,audio and video and client can verify that this message is audio message or text message or video message. kindly guide me.

Reply • Share ›

Amel • a year ago

i can't run the client

Reply • Share ›

Ruel • a year ago

Hello, i can't connect the server i created externaly, i have forwarded the port, and habilitated the firewall. With localhost i can connect it perfectly

Reply • Share ›

Pankaj Vavadiya • a year ago

This is very nice tutorial to learn basic of socket programming.
All written program is working.

Just mention one line that

If you want to use pthred.h then you have to compile your code with following statement
gcc -pthread sourcefilename.c

I hope you understand well

Thank you very much

Reply • Share ›

Sajeevan Agarwal • a year ago

The Tutorial Seems great but am stuck up. I was able to create the
socket but each time I am trying to connect the google server I am
failing and am unable to connect to it.

I am getting a return value of -1 for connect(socket_desc , (struct sockaddr *)&server , sizeof(server)) ;
each time.
Can
any one please help me with that, I am running a Ubuntu 14.04 on
Virtual Machine and I am having internet connection through wifi does it
have any impact on the connection.

Reply • Share ›

Indian ➦ Sajeevan Agarwal • 8 months ago

open cmd window and type
nslookup www.google.com
where u will get the ip address for google.com
74.125.236.83

Reply • Share ›

Dean ➦ Sajeevan Agarwal • 10 months ago

I had the same trouble. It's the particular IP address used here (74.125.235.20).

Try using one of Google's other IP addresses, like 74.125.224.72.

Reply • Share ›

Life Diluted • a year ago

Hello there,
Why don't you consider making some video tutorials on YouTube. It will be great.
Thanks for this tutorial.

Reply • Share ›

Rohitash Jain • a year ago

exceelent tutorials

but while running the last code i am getting in this line

new_sock = malloc(1);

invalid conversion from ‘void*’ to ‘int*’

please anybody help me here

Reply • Share ›

Loc Nguyen ➦ Rohitash Jain • a year ago

Iam also got the same problem, i change it to new_sock = (int*) malloc(sizeof(int*));
i works but only 1 time, when i disconnect and reconnect it a gain, it does not feedback our data send

Reply • Share ›

Hans ➦ Rohitash Jain • a year ago

Just cast it:

new_sock = (int*)malloc(1);

and actually, shouldn't it have enough space for an int? So I would write

new_sock = (int*) malloc(sizeof(int));

Reply • Share ›

Shres • a year ago

Thanks for article. Any advise on upgrading this code to use browser as a client and may be using html5.
Reply • Share ›

lalit • a year ago
Great tutorial....every program is running....concepts are explained throughly....
Reply • Share ›

vineeth • a year ago
I am successfully able to create the socket. But when i am trying to connect to the google server as mentioned in the code i am getting the error as Connect errror.
Why this error ?
Reply • Share ›

Dean → vineeth • 10 months ago
Try using one of Google's other IP addresses. The one in the tutorial didn't work for me either.
Reply • Share ›

Од Алдар • a year ago
What does "&" mean in "bind(socket_desc,(struct sockaddr *)&server , sizeof(server))" ?
Reply • Share ›

frustrated student → Од Алдар • a year ago
here is the meaning
bind("the socket to be bound", the address, size of the address)
look at the components of struct sockaddr_in
1 • Reply • Share ›

baagii • 2 years ago
how to FTP server ?
Reply • Share ›

koko • 2 years ago
Hello this is really great thanks alot, just a simple question, i am able to connect to the server using couple computers on my room, telnet from windows works great too except it doesn't wait me to press enter it repeats the char as soon as i type it, my question is but what if i wanted to connect from a non local ip address? could you please give me some hints?
Reply • Share ›

sailee jain • 2 years ago
nice tutorial :) Have you written any such tutorial for writing raw sockets in C ?
Reply • Share ›

Silver Moon Mod → sailee jain • 2 years ago
yes, you can check the following tutorial

http://www.binarytides.com/raw...
1 • Reply • Share ›

Babita Sandooja → Silver Moon • a year ago
Thanks sir. Such an awesome tutorial. But I need the same for raw sockets. The link you mentioned is just sending data out. How about handling multiple connections using raw socket? I actually want to implement a chat room using raw sockets in C. Any help would be highly appreciated. Just give me idea or tell me the steps to do like the function calls, etc. Thanks :)
1 • Reply • Share ›

nanducg • 2 years ago
how udp working..
Reply • Share ›

ari • 2 years ago
A great tutorial.
thank u very much.
Reply • Share ›

Bala reegaa • 3 years ago
thanks for ur xcellent tutorial....keep on going
Reply • Share ›

unclepi • 3 years ago
That's great! Thank you
Reply • Share ›

Pancho • 3 years ago
Excellent tutorials!

Reply • Share ›

sailaja • 3 years ago
Thanks, I was able to understand this easily
Reply • Share ›

sujin • 3 years ago
Thanks Mr
Reply • Share ›

saikrishna • 3 years ago
how to connect with other systems in the room using sockets...there we need to specify just the ipaddress of the system....please clarify
Reply • Share ›

Silver Moon Mod saikrishna • 3 years ago
first you need to run a socket server on the other machines that open a certain port N.
then on your computer you have to run the socket client, and give it the ip address of the other machine and the port number N to connect.
Reply • Share ›

Chris Apostolopoulos • 3 years ago
Hello,thank you for this tutorial , it was very useful! Just one question:

I'm trying to write a program that connects to itself , create a client and server and then sends a string and prints to from the server. I searched through the net and saw that instead of using inet_addr(site IP) you can use htol(INADDR_ANY) , but this creates an error when connecting the socket. Any tips?
Reply • Share ›

Load more comments

Subscribe   Add Disqus to your site   Privacy   DISQUS