Server and client example with C sockets on Linux

C By Silver Moon On Jul 30, 2012 36 Comments 讚 90 +1 15 Tweet 7

In a previous example we learnt about the basics of socket programming in C. In this example we shall build a basic ECHO client and server. The server/client shown here use TCP sockets or SOCK_STREAM.

Tcp sockets are connection oriented, means that they have a concept of independant connection on a certain port which one application can use at a time. The concept of connection makes TCP a "reliable" stream such that if errors occur, they can be detected and compensated for by resending the failed packets.

Server

Lets build a very simple web server. The steps to make a webserver are as follows :

1. Create socket
2. Bind to address and port
3. Put in listening mode
4. Accept connections and process there after.

Quick example

```
/*
    C socket server 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 , client_sock , c , read_size;
    struct sockaddr_in server , client;
    char client_message[2000];

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

    //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)
    {
        //print the error message
        perror("bind failed. Error");
        return 1;
    }
    puts("bind done");

    //Listen
    listen(socket_desc , 3);

    //Accept and incoming connection
    puts("Waiting for incoming connections...");
    c = sizeof(struct sockaddr_in);

    //accept connection from an incoming client
```

```
    client_sock = accept(socket_desc, (struct sockaddr *)&client, (socklen_t*)&c);
    if (client_sock < 0)
    {
        perror("accept failed");
        return 1;
    }
    puts("Connection accepted");

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

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

    return 0;
}
```


The above code example will start a server on localhost (127.0.0.1) port 8888
Once it receives a connection, it will read some input from the client and reply back with the same message.
To test the server run the server and then connect from another terminal using the telnet command like this

$ telnet localhost 8888

Client

Now instead of using the telnet program as a client, why not write our own client program. Quite simple again

```
/*
    C ECHO client example using sockets
*/
#include<stdio.h> //printf
#include<string.h>    //strlen
#include<sys/socket.h>    //socket
#include<arpa/inet.h> //inet_addr

int main(int argc , char *argv[])
{
    int sock;
    struct sockaddr_in server;
    char message[1000] , server_reply[2000];

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

    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_family = AF_INET;
    server.sin_port = htons( 8888 );

    //Connect to remote server
    if (connect(sock , (struct sockaddr *)&server , sizeof(server)) < 0)
    {
        perror("connect failed. Error");
        return 1;
    }

    puts("Connected\n");

    //keep communicating with server
    while(1)
    {
        printf("Enter message : ");
        scanf("%s" , message);

        //Send some data
        if( send(sock , message , strlen(message) , 0) < 0)
        {
            puts("Send failed");
            return 1;
        }

        //Receive a reply from the server
        if( recv(sock , server_reply , 2000 , 0) < 0)
        {
```

```
            puts("recv failed");
            break;
        }

        puts("Server reply :");
        puts(server_reply);
    }

    close(sock);
    return 0;
}
```

The above program will connect to localhost port 8888 and then ask for commands to send. Here is an example, how the output would look

$ gcc client.c && ./a.out
Socket created
Connected

Enter message : hi
Server reply :
hi
Enter message : how are you

Server to handle multiple connections

The server in the above example has a drawback. It can handle communication with only 1 client. Thats not very useful. One way to work around this is by using threads. A thread can be assigned for each connected client which will handle communication with the client.

Code example

```
/*
    C socket server example, handles multiple clients using threads
*/

#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

//the thread function
void *connection_handler(void *);

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

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

    //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)
    {
        //print the error message
        perror("bind failed. Error");
        return 1;
    }
    puts("bind done");

    //Listen
    listen(socket_desc , 3);

    //Accept and incoming connection
    puts("Waiting for incoming connections...");
    c = sizeof(struct sockaddr_in);


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

        pthread_t sniffer_thread;
        new_sock = malloc(1);
        *new_sock = client_sock;

```

```
62          if( pthread_create( &sniffer_thread , NULL ,  connection_handler , (void*) new_so
63          {
64              perror("could not create thread");
65              return 1;
66          }
67
68          //Now join the thread , so that we dont terminate before the thread
69          //pthread_join( sniffer_thread , NULL);
70          puts("Handler assigned");
71      }
72
73      if (client_sock < 0)
74      {
75          perror("accept failed");
76          return 1;
77      }
78
79      return 0;
80  }
81
82  /*
83   * This will handle connection for each client
84   * */
85  void *connection_handler(void *socket_desc)
86  {
87      //Get the socket descriptor
88      int sock = *(int*)socket_desc;
89      int read_size;
90      char *message , client_message[2000];
91
92      //Send some messages to the client
93      message = "Greetings! I am your connection handler\n";
94      write(sock , message , strlen(message));
95
96      message = "Now type something and i shall repeat what you type \n";
97      write(sock , message , strlen(message));
98
99      //Receive a message from client
100     while( (read_size = recv(sock , client_message , 2000 , 0)) > 0 )
101     {
102         //Send the message back to client
103         write(sock , client_message , strlen(client_message));
104     }
105
106     if(read_size == 0)
107     {
108         puts("Client disconnected");
109         fflush(stdout);
110     }
111     else if(read_size == -1)
112     {
113         perror("recv failed");
114     }
115
116     //Free the socket pointer
117     free(socket_desc);
118
119     return 0;
120 }
```

Run the above server and connect from multiple clients and it will handle all of them. There are other ways to handle multiple clients, like select, poll etc. We shall talk about them in some other article. Till then practise the above code examples and enjoy.

Last Updated On : 27th November 2012

socket programming

Related Posts

Programming udp sockets in C on Linux

Socket programming in C on Linux – tutorial

C program to get a domain's whois information using sockets on Linux

Programming udp sockets in python

Code a simple socket server in Python

About **Silver Moon**

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

Loco Pepe © • 2 years ago
Hi Silver, thanks for every tutorial you wrote here, they have been very useful :). I do have a question, though: I need to make a server write a client's message into every other client connected to simulate a chatroom. So if client 1 writes "hello", client 2 and 3 see in their own screen "Client 1 wrote: hello". Any ideas to do this? Thanks for everything!.
11 • Reply • Share ›

Aja • a year ago
Can someone explain the pthread_join? Why is it commented out? I assume we need it, but where should it go in the code?
7 • Reply • Share ›

oladunk123 • 2 years ago
Thanks for writing tutorials, but as far as the multi client versions goes, both this version and the revised version posted below are prone to errors - for different reasons. The original version containing a buffer overflow/possible access violation and the revised version containing a race condition.
7 • Reply • Share ›

Monica • 3 years ago
I want to include one functionality which is not working here.
If we disconnect the server, all the client processes connecte dto it, will automatically be disconnected.
How to add that functionality in the above code?
3 • Reply • Share ›

3bood • 2 years ago
Hey every one; i tried to compiler the multithraded server code but i got this message:

/tmp/ccTHnrln.o: In function `main':
thread.c:(.text+0x142): undefined reference to `pthread_create'
collect2: ld returned 1 exit status
2 • Reply • Share ›

Brennan ↗ 3bood • 2 years ago
add -lpthread flag when you compile
4 • Reply • Share ›

Nayank ↗ Brennan • 9 months ago
Hey you got a solution? I am also stuck with this problem.
• Reply • Share ›

shubham ↗ Brennan • a year ago
how it can be implemented...? please
• Reply • Share ›

ChrisX • a year ago
http://stackoverflow.com/quest...

This code itself has some serious issues in it. Do NOT use it, on the long term if you try to build a server on this it will hang after a time.
1 • Reply • Share ›

leonardo • 3 years ago
thank you very much for your explanation!, it help me a lot!!
1 • Reply • Share ›

Phani Krishna • a month ago
Thanks for an amazing tutorial
• Reply • Share ›

lgrcyanny • a year ago
Great tutorial, thanks, for multi connection, epoll will be a nice choice

• Reply • Share ›

keen123 • a year ago

Nice tutorial. I do have a question:
In this program we're using INADDR_ANY as the server address, which means listen on all the available IP address.
How to re-write a this program if I want to achieve this:
1. Server should listen on two specific sockets lets say for example 192.168.1.1:8888 & 192.168.1.2:8888?

Can you please demonstrate the same, it would be really helpful for me?

• Reply • Share ›

Murad Ali • 2 years ago

Kindly any one tell me how we can make this code so that client can communicate with client in multi clients

• Reply • Share ›

Javier • 2 years ago

Silver,
Excellent code. Just what I needed. Thank you.

Greetings from Argentina,
Javier

• Reply • Share ›

Irfan Doank • 2 years ago

great code...i have use this code to receive data from gps thanks...but when i try receive data from vt310 gps meitrack, i just receive $$, can you help me?sorry for my bad english...

• Reply • Share ›

Teem • 2 years ago

how to kick client if send wrong message?

• Reply • Share ›

Marc • 2 years ago

Very nice well explained piece of work. Got it to build and run on a PC with eclipse and Cygwin compiler in a few minutes :-) , think I needed to add one stupid "include " for the definition of "close".

• Reply • Share ›

Hadri Rahman • 2 years ago

I got the multithreaded code to work, but when I send a message using the client, I get this:

$ ./echoclient.out
Socket created

Connected

Enter message: Hello
Server reply:

Hi There! I'm the connection handler.

Type the message and the server shall repeat it.

Enter message: Hello
Server reply:
Hellohere! I'm the connection handler.

Type the message and the server shall repeat it.

---

see more

• Reply • Share ›

Loco Pepe © ↗ Hadri Rahman • 2 years ago

Same thing happened to me. No idea why :/

1 • Reply • Share ›

serg ↗ Loco Pepe © • 2 years ago

The problem is in reusind "client_message" buffer. Replace in server code:

```
while( (read_size = recv(client_sock , client_message , 2000 , 0)) > 0 )
{
//Send the message back to client
write(client_sock , client_message , strlen(client_message));
}
```

to:

```
while( (read_size = recv(client_sock , client_message , 2000 , 0)) > 0 )
{
//Send the message back to client
write(client_sock , client_message , read_size);
}
```

This will send exactly received number of bytes, not whole string.

2 • Reply • Share ›

Pranav ➔ serg • 8 months ago

sorry to say but this suggestion did not help in solving the problem...
the same thing continues
:(

• Reply • Share ›

Loca lino ➔ Hadri Rahman • 3 months ago

thanks!

• Reply • Share ›

Alexander Scoutov ➔ Hadri Rahman • a year ago

I just put
memset(client_message,'\0',sizeof(client_message));

into the end of the block
while((read_size = recv(client_sock , client_message , 2000 , 0)) > 0)

• Reply • Share ›

Hadri Rahman • 2 years ago

Hey, I tried running the multithreaded server code, but when I compiled it I received this error message:

echoserver.c:88:1: error: expected declaration or statement at end of input

Where line 88 is the closing curly bracket just after :
free(socket_desc);
return 0;
}

I'm not sure what went wrong, any suggestions?

• Reply • Share ›

jimmy • 2 years ago

Hi Silver,

first of all thanks a lot for sharing this. I really appreciate it, so thanks again.

Here come my questions, just in order to clarify:

1. You declare new_sock as a pointer and dynamically allocate e new byte each time a new client connects. Why didn't you just declared it as a normal int, assigning the pointer to client_sock as you already did?

2. Sniffer_thread is only declared once and no dynamic allocation happens after a client connects, so I assume more than one thread (or thread_handler) can refer to the same pthread_t variable without any influence on the previously created/started threads. Am I right?

3. When passing arguments to the thread handler function (I see you pass the new socket variable as a pointer...), these become thread variables, thus are not shared by the different threads, am I right?

4. This would also be valid if passing a client address to the thread handler, thus it would not affect the client address of any other connection handled by the other threads, right?
Again, thanks a lot for the code snippet.

• Reply • Share ›

Silver Moon Mod ➔ jimmy • 2 years ago

well, the thread handling shown in the code, may not be fully correct, I just wrote it as an experiment and it worked.

1. I created new_sock as a pointer to allocate memory separately for each thread, otherwise it would get overwritten across threads.

2. The sniffer_thread (should have named it something better) variable is created everytime in the while loop, so a new one gets created for every client. Its address is the first parameter to pthread_create function.

3. Not exactly, the new_sock is created by doing a malloc(1); in the main thread, so a new one is created for every thread.

4. You can pass any custom structure variable to the thread function and store anything in it. Like create a struct, fill it in the main thread with socket pointer, address variables, and then pass to thread. Just make

sure that in the main thread it does not get over-written (by using malloc for example)
• Reply • Share ›

jimmy ➚ Silver Moon • 2 years ago
Hi Silver,

thanks once again for the prompt answer. You are right for what concerns point 4: as long as the thread copies the passed struct in an own local struct, so it gets not over-written by the next thread operation, everything should be fine.

Still, for what concerns points 1 to 3, I'm not quite sure the dynamic allocation was the best choice and also the constant declaration of a new pthread_t var in the while loop sounds a little weird to me. I compared a couple of other threaded server sources and many of them declare the new_sock variable as a global one, passing it as an argument and not caring anymore about it (as it gets copied in a local thread variable) and only declare the pthread_t var once (in the main function variables). It then gets handled by the OS (?) in order to create all the needed and independent threads...as far as I got. Am I missing something or did I get it right?
Thanks again for the clarifications.
• Reply • Share ›

Silver Moon Mod ➚ jimmy • 2 years ago
I gave the code a little thought. Your arguments are probably right.

Since we are accept and create a thread for only 1 incoming connection at a time, it is okay to use the same new_sock and pthread_t variable. The thread would create its local copy of new_sock and work fine.

I guess its all right to declare pthread_t variable only once and reuse it.

According to http://man7.org/linux/man-page...
###

Before returning, a successful call to pthread_create() stores the ID of the new thread in the buffer pointed to by thread; this identifier is used to refer to the thread in subsequent calls to other pthreads functions.

###

Therefore you can use the same variable again and again.

I wrote another version of the server following your ideas and it works fine.

https://gist.github.com/silv3r...
• Reply • Share ›

tibalt ➚ Silver Moon • a year ago
see line 62 in https://gist.github.com/silv3r...
I have met such issue that the "client_sock" is modified before the "connection_handler" handle the connection. That means one client's request will be ignored forever!
• Reply • Share ›

Forvaine • 2 years ago
Perfect example, thanks a lot!
Will be checking out the select version as well. :)
• Reply • Share ›

Sravan Reddy • 2 years ago
Thanks for the example! Just what I am looking for.
I am wondering whether a similar threaded C-server will be sufficient when the client is a Python Web-handler issuing large number of simultaneous requests to the C-program.
• Reply • Share ›

Silver Moon Mod ➚ Sravan Reddy • 2 years ago
the server and client can be in different languages, it does not matter.

However, how many requests the server can handle will depend on how it is coded. Instead of threads, select function can be used to handle multiple connections.
Check the following article for details on how to use the select function.

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

Hari Shankar • 2 years ago
very well laid out and crisp ! Thankyou!
• Reply • Share ›

Shai • 2 years ago

Thank you,

Great examples. it work for me

• Reply • Share ›

shruthi • 2 years ago

iam getting connection refused error

• Reply • Share ›