Tidy Tutorials

Programming tutorials, examples and code snippets. C++, Java, Perl, PHP and javascript. Enterprise examples using javaEE, JNDI and Beans. Core tutorials using sockets, threads, database, IPC, XML etc. Web 2 widgets using javascrpt. Simple easy to understand step by step tutorials,

Linux C++ Socket Example with Client Server and Mulit-Threading

Aim

The aim of this Linux C++ example is to create simple client/server socket communication and multi-threading program on Linux. This shows the basics such as binding, listening and accepting sockets for Servers and connecting sockets for clients. Once the server accepts a socket, a thread is spawned to handle it. The threads are C++ POSIX threads. This is the simplest Linux C++ multi-threaded socket example you can find. Please leave any comments or questions at the end of this tutorial and I will endeavour to answer them. For a forking server look at Linux C++ Forking Server and Client Example.

Assumptions

This article assumes that you have a Linux based C++ compiler installed and configured. You can use cygwin if you do not have a Linux environment.

Versions used in this example

Sofware/Component	Image
Fedora Core 5	N/A

Links to these files can be found here


You can download the zipped example here.

Steps required for a server socket

1. Initialize the socket using socket().
2. Set any options such as blocking etc using setsockopt().
3. Bind to the local address and port using bind(). For a server use INADDR_ANY as the address.
4. listen() for connections.
5. Go into a loop and accept connections usin accept(). Spawn threads (pthread_create) to handle these connections, so you can accept more connections.

   Read and write to the socket. within the thread.
6. close()
7. When you are done, close the server socket using close()

Steps required for a client socket

1. Initialize the socket using socket().
2. Set any options such as blocking etc using setsockopt().
3. Connect to the remote host using connet(). Supply address and port here.

   Read and write to the socket. close()

Write and compile the Server

1. The server listens on port 1101 for a connection. When it receives a connection it creates a thread via pthread_create to handle it. This thread then reads from the socket, appends " SERVER ECHO" to it and sends it back to the client. As the server doesn't require an IP address, we assign INADDR_ANY to the sockaddr_in struct. Save this code as LinSever.cpp.

```
#include <fcntl.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <stdio.h>
#include <netinet/in.h>
#include <resolv.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <pthread.h>

void* SocketHandler(void*);

int main(int argv, char** argc){

    int host_port= 1101;

    struct sockaddr_in my_addr;

    int hsock;
    int * p_int ;
    int err;

    socklen_t addr_size = 0;
    int* csock;
    sockaddr_in sadr;
    pthread_t thread_id=0;


    hsock = socket(AF_INET, SOCK_STREAM, 0);
    if(hsock == -1){
        printf("Error initializing socket %d\n", errno);
        goto FINISH;
```

Code Diaries Sitemap

```
    }

    p_int = (int*)malloc(sizeof(int));
    *p_int = 1;

    if( (setsockopt(hsock, SOL_SOCKET, SO_REUSEADDR, (char*)p_int, sizeof(int)) == -1 )||
        (setsockopt(hsock, SOL_SOCKET, SO_KEEPALIVE, (char*)p_int, sizeof(int)) == -1 ) ){
        printf("Error setting options %d\n", errno);
        free(p_int);
        goto FINISH;
    }
    free(p_int);

    my_addr.sin_family = AF_INET ;
    my_addr.sin_port = htons(host_port);

    memset(&(my_addr.sin_zero), 0, 8);
    my_addr.sin_addr.s_addr = INADDR_ANY ;

    if( bind( hsock, (sockaddr*)&my_addr, sizeof(my_addr)) == -1 ){
        fprintf(stderr,"Error binding to socket, make sure nothing else is listening on this port %d\n",errno)
        goto FINISH;
    }
    if(listen( hsock, 10) == -1 ){
        fprintf(stderr, "Error listening %d\n",errno);
        goto FINISH;
    }

    //Now lets do the server stuff

    addr_size = sizeof(sockaddr_in);

    while(true){
        printf("waiting for a connection\n");
        csock = (int*)malloc(sizeof(int));
        if((*csock = accept( hsock, (sockaddr*)&sadr, &addr_size))!= -1){
            printf("---------------------\nReceived connection from %s\n",inet_ntoa(sadr.sin_addr));
            pthread_create(&thread_id,0,&SocketHandler, (void*)csock );
            pthread_detach(thread_id);
        }
        else{
            fprintf(stderr, "Error accepting %d\n", errno);
        }
    }

FINISH:
;
}

void* SocketHandler(void* lp){
    int *csock = (int*)lp;

    char buffer[1024];
    int buffer_len = 1024;
    int bytecount;

    memset(buffer, 0, buffer_len);
    if((bytecount = recv(*csock, buffer, buffer_len, 0))== -1){
        fprintf(stderr, "Error receiving data %d\n", errno);
        goto FINISH;
    }
    printf("Received bytes %d\nReceived string \"%s\"\n", bytecount, buffer);
    strcat(buffer, " SERVER ECHO");

    if((bytecount = send(*csock, buffer, strlen(buffer), 0))== -1){
         fprintf(stderr, "Error sending data %d\n", errno);
         goto FINISH;
     }

     printf("Sent bytes %d\n", bytecount);


FINISH:
     free(csock);
     return 0;
}
```

Hide line numbers

2. Open a prompt to the working directory and compile the code using your C++ compiler

```
..workspace\SocketExample>g++ -o server LinServer.cpp -lpthread
```

Write and compile the Client

1. The client reads a line from the console and sends this to the server. It then reads from the server and displays it on the console. Save this code as LinClient.cpp.

```
#include <fcntl.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <stdio.h>
#include <netinet/in.h>
#include <resolv.h>
#include <sys/socket.h>
```

```
#include <arpa/inet.h>
#include <unistd.h>

int main(int argv, char** argc){

    int host_port= 1101;
    char* host_name="127.0.0.1";

    struct sockaddr_in my_addr;

    char buffer[1024];
    int bytecount;
    int buffer_len=0;

    int hsock;
    int * p_int;
    int err;

    hsock = socket(AF_INET, SOCK_STREAM, 0);
    if(hsock == -1){
        printf("Error initializing socket %d\n",errno);
        goto FINISH;
    }

    p_int = (int*)malloc(sizeof(int));
    *p_int = 1;

    if( (setsockopt(hsock, SOL_SOCKET, SO_REUSEADDR, (char*)p_int, sizeof(int)) == -1 )||
        (setsockopt(hsock, SOL_SOCKET, SO_KEEPALIVE, (char*)p_int, sizeof(int)) == -1 ) ){
        printf("Error setting options %d\n",errno);
        free(p_int);
        goto FINISH;
    }
    free(p_int);

    my_addr.sin_family = AF_INET ;
    my_addr.sin_port = htons(host_port);

    memset(&(my_addr.sin_zero), 0, 8);
    my_addr.sin_addr.s_addr = inet_addr(host_name);

    if( connect( hsock, (struct sockaddr*)&my_addr, sizeof(my_addr)) == -1 ){
        if((err = errno) != EINPROGRESS){
            fprintf(stderr, "Error connecting socket %d\n", errno);
            goto FINISH;
        }
    }

    //Now lets do the client related stuff

    buffer_len = 1024;

    memset(buffer, '\0', buffer_len);

    printf("Enter some text to send to the server (press enter)\n");
    fgets(buffer, 1024, stdin);
    buffer[strlen(buffer)-1]='\0';

    if( (bytecount=send(hsock, buffer, strlen(buffer),0))== -1){
        fprintf(stderr, "Error sending data %d\n", errno);
        goto FINISH;
    }
    printf("Sent bytes %d\n", bytecount);

    if((bytecount = recv(hsock, buffer, buffer_len, 0))== -1){
        fprintf(stderr, "Error receiving data %d\n", errno);
        goto FINISH;
    }
    printf("Recieved bytes %d\nReceived string \"%s\"\n", bytecount, buffer);

    close(hsock);

FINISH:
;
}
```

Hide line numbers

2. Open a prompt to the working directory and compile the code using your C++ compiler

```
..workspace\SocketExample>g++ -o client LinClient.cpp
```

Run the example

1. Open a prompt to the working directory and run the server.

```
..workspace\SocketExample>server
```

2. Open another prompt to the working directory and run the Client.

```
..workspace\SocketExample>client
```

3. Now type something into the client prompt and press ENTER.
4. You will see the exchange between client and server. You can open many client prompts at the same time and test.

24 comments:

jr said...

Thanks for this example...It help me alot

October 14, 2010 at 7:06 PM

Anonymous said...

The socket server leaks memory, in Ubuntu 10.04. Eash time I use the example client, then use ps v I see an increase in memory use (size) by 8 bytes.

I was looking on the web for examples because my own server also leaks memory and since my program has to run continuously that will be fatal.

October 31, 2010 at 9:51 PM

righteous said...

Yes, quite rightly so. I'm not calling pthread_detach or pthread_join and I should be doing either one of them. I will fix the example as soon as I can.

November 1, 2010 at 4:12 AM

Anonymous said...

Hey , i would like to clarify regarding the server coding. You stated that to close the server port , we are to make use of close()? I tried using csock.close() but it dosnt seem to work for me. Am i getting it wrong? I used hsock.close() but im facing the same problem

February 9, 2011 at 4:29 AM

righteous said...

These are linux sockets, csock and hsock are integers and not classes.

To close use close(hsock).

February 9, 2011 at 9:56 PM

Anonymous said...

in c is better to initialize and declare using:
char host_name[]={"127.0.0.1"};
than
char* host_name="127.0.0.1";

listen the compiler warning :)

also you can avoid dangerous goto using return 1;

August 19, 2011 at 8:59 AM

righteous said...

good oberservations. Goto is not dangerous.

Remember this is just an example. Not production code :)

August 20, 2011 at 5:36 AM

dddddd said...

thanx for the example.....

September 26, 2011 at 3:33 AM

hyd said...

very good example of multiple client server program

September 26, 2011 at 3:36 AM

Anonymous said...

it seems weird for me whenever i have multiple clients. it seems to kick the previous client out...

what's the problem?

October 2, 2011 at 12:28 AM

Anonymous said...

How would I broadcast a message to all clients from the server?

October 9, 2011 at 8:14 AM

Anonymous said...

Thanks!, this example really help me ;)

November 11, 2011 at 8:03 PM

November 11, 2011 at 8:03 PM

Anonymous said...

should be close(*csock); or free(*csock); and not free(csock);

January 11, 2012 at 7:57 PM

rahul soni said...

if there are 5 clients connected, how do I send a data to say 4th client ?

June 22, 2012 at 12:48 AM

righteous said...

That's completely upto your implementation. Because it's multithreaded you can keep track of your clients through a global struct.

June 27, 2012 at 9:30 PM

Anonymous said...

Thanks for this example it has help immensely, now I have a much better understanding. Also thanks to the supporting commentary as well.

July 9, 2012 at 8:36 AM

Anonymous said...

Thank you very much. Saved me lots of time.

August 25, 2012 at 1:03 PM

Anonymous said...

This is just what I needed to get started. Thanks!!!

February 15, 2013 at 8:41 AM

Anonymous said...

Thank you for the example.

February 18, 2013 at 1:06 AM

Anonymous said...

Could you please tell me how to implement this for making game that can be played over a network?

Please reply asap!

October 29, 2013 at 11:28 AM

Anonymous said...

how to add time in this c++? which mean at the client it display time...

December 15, 2013 at 10:25 AM

shanmukavel_profile said...

Hello Sir,
I am new for android. i trying for data transfer between computer and mobile phone via USB cable.
Communication can be done two way:
1) The first way is Socket,Protocol so like
2) The Second way is Using USB functions
ie, can be take example for in android USB HOST and USB Accessory Methods and in JAVA can also have JAVA HOST methods.
Please sir,If you will have this code means send me sir,otherwise give some ideas and techniques of code..
thank you sir…

December 22, 2013 at 8:55 PM

Anonymous said...

Problems:
1) The program is a C program not a C++ program (well, it's both).
2) It makes no usage of C++ features.
3) The code is confusing to read unless you know what's going on.
4) The program creates a new thread per client instead of using select/pselect to handle multiple clients.

June 26, 2014 at 3:27 AM

Naviya Nair said...

Very interesting and good Explanation
ASP NET Training
ASP NET Training
ASP NET Online Training

C-Sharp Training
Dot Net Training in Chennai
Online .Net Training

MVC Training
WCF Training
Web-API Training
LINQ Training
Entity Framework
Training
Dot Net Interview Questions

December 19, 2015 at 1:24 AM

Post a Comment

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)