**Introduction**

Chat server is an application which is basically responsible for listening to incoming calls of clients. Client running in any PC can connect to the server if IP address of the server is known. It can respond to the messages of connected clients. It broadcasts the message from clients to all the clients connected to the server

We have tried to implement a similar concept by designing a standalone application that is made up the combination of two-application, server application (which runs on server side) and client application (which runs on client side). This application is using for chatting in LAN**.**

For making this application we are using C programming language in Linux networking, and threading also.

**Objective**

Most operating systems provide precompiled programs that communicate across a network. Common examples into the TCP/IP world are web clients (browsers) and web servers, and the FTP and TELNET clients and servers. Sometimes when we are using this utilities of the internet we don't think about all the process involved.

We have tried to design a chat client-server based application, a mechanism that makes communication possible over the Network.

We examine the functions for communication through sockets. A socket is an endpoint used by a process for bi-directional communication with a socket associated with another process.

**How it works?**

Most network applications can be divided into two pieces: a client and a server.

Creating a socket

#include <sys/types.h>

#include <sys/socket.h>

When you create a socket there are three main parameters that you have to specify:

* the domain
* the type
* the protocol

The Domain parameter specifies a communications domain within which communication will take place. Finally the protocol type, since we used a Stream Socket type we must use a protocol that provide a connection-oriented protocol, like IP, so we decide to use IP in our protocol Type, and we saw in /etc/protocols the number of ip, 0. So our function now is:

* s = socket(AF\_INET , SOCK\_STREAM , 0)
* where’s’ is the file descriptor returned by the socket function.

**The Mini-chat Server structure**

**Binding a socket to a port and waiting for the connections**

Like all services in a Network TCP/IP based, the sockets are always associated with a port, like Telnet is associated to Port 23, FTP to 21... In our Server we have to do the same thing, bind some port to be prepared to listening for connections (that is the basic difference between Client and Server), Listing 2. Bind is used to specify for a socket the protocol port number where it will be waiting for messages.

The IP used:-

address.sin\_addr.s\_addr = INADDR\_ANY /\*use a specific IP of host\*/

The port used:-

address.sin\_port = htons(15000); /\* use a specific port number \*/

And finally bind our port to the socket.

bind(create\_socket , (struct sockaddr \*)&address,sizeof(address));

Now another important phase, prepare a socket to accept messages from clients, the listen function is used on the server in the case of connection oriented communication and also the maximum number of pending connections (resource 3).

listen (create\_socket, MAXNUMBER)

Maybe the biggest difference is that client needs a Connect () function. The connect operation is used on the client side to identify and, possibly, start the connection to the server. The connect syntax is

Connect (create\_socket,(struct sockaddr \*)&address,sizeof(address)) ;

The Send () function is used to send the buffer to the server

send(new\_socket,buffer,bufsize,0);

and the Recv() function is used to receive the buffer from the server, look that it is used both in server and client.

recv(new\_socket,buffer,bufsize,0);

**Conclusion**

Since the software of the TCP/IP protocol is inside the operating system, the exactly interface between an application and the TCP/IP protocols depends of the details of the operating system(resource 4).In our case we examine the UNIX BSD socket interface because Linux follow this.

The Mini-chat developed here is nothing more than a explain model of a client/server application using sockets in Linux and should be used like a introduction of how easy is to develop applications using sockets. After understand this you can easily start to think about IPC (Inter Process Communication), fork, threads (resource 5) and much more. The basic steps to make it work are:

1. Run the server
2. Run the client with the address of the server.

**Resources**

1. Operating Systems , Harvey M. Deitel , 1990
2. Socket Linux Man Page
3. Network Functions in C - Tutorial
4. Internetworking with TCP/IP Vol1 - Doulgas Commer
5. Unix Network Programming , Vol2 , Richard Stevens
6. Unix Network Programming, Vol1, Richard Stevens