INTRODUCTION

The given assignment asks us to implement a basic server and client using sockets.

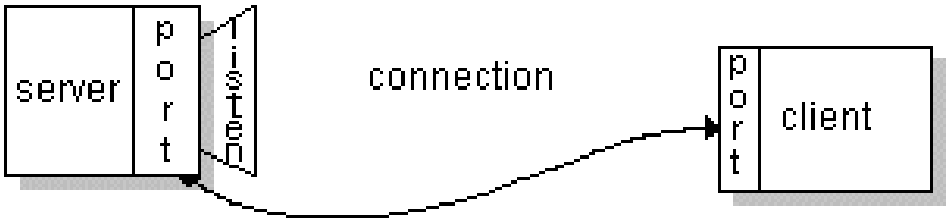
A network socket is an internal endpoint which allows communication between two different processes on the same or different machines.

A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.

The server program just waits, listening to the socket for a client to make a connection request.

On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.

If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.

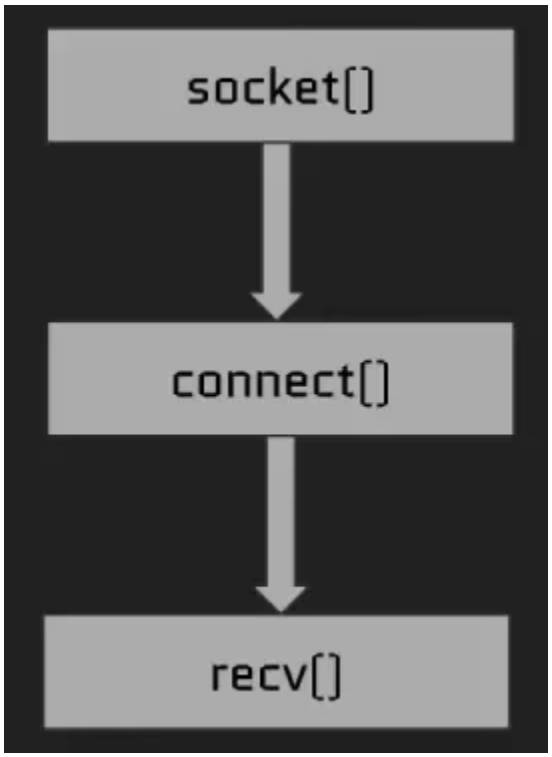
The client and server can now communicate by writing to or reading from their sockets.

We implement TCP based server client communication as taught in class.

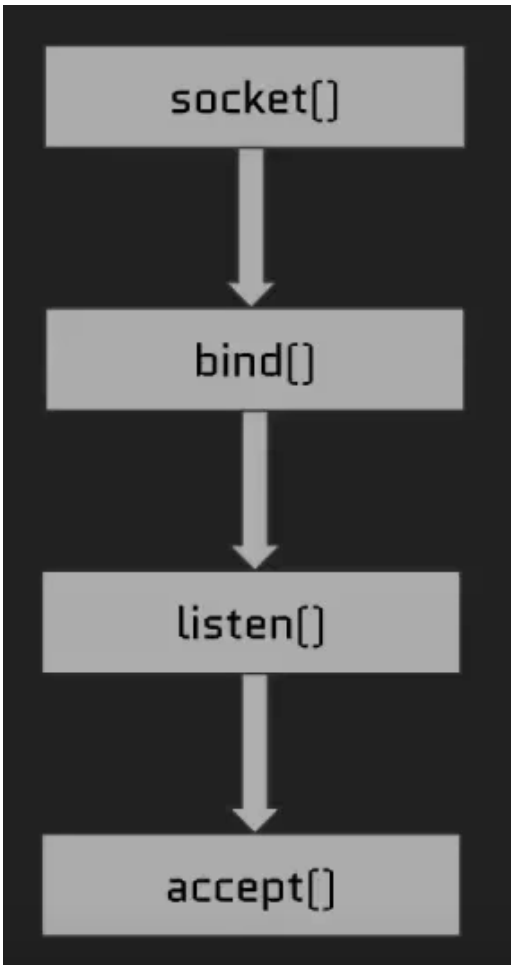
As a matter of fact, common networking protocols like HTTP and FTP rely on sockets underneath to make connections.

SOCKET PROGRAMMING IN LINUX

Linux provides us with a nice API to deal with socket programming, which we use in this assignment.



Client Socket Workflow



Server Socket Workflow

The API provides with functions to perform each of the above mentioned steps. Refer to code for detailed comments on implementation.

DEMO

1. *Copy the server.c to a folder(server directory).*
2. *Create a few text files with contents in this (server)directory.*
3. *Copy the client.c to a different folder(client 1 directory).*
4. *Copy client.c again to another folder(client 2 directory).*
5. *From the server directory, run the command: gcc server.c -o server –lpthread*
6. *From client 1 directory run: gcc client.c -o client1*
7. *From client 2 directory run: gcc client.c -o client2*
8. *Now run client1, client2, client3 from the respective directories in different terminals.*
9. *The server is handling multiple clients*

10. *To exit client, type exit. (Note: CTRL + C quits the client and server as well because both are foreground processes and they have the same process group)*