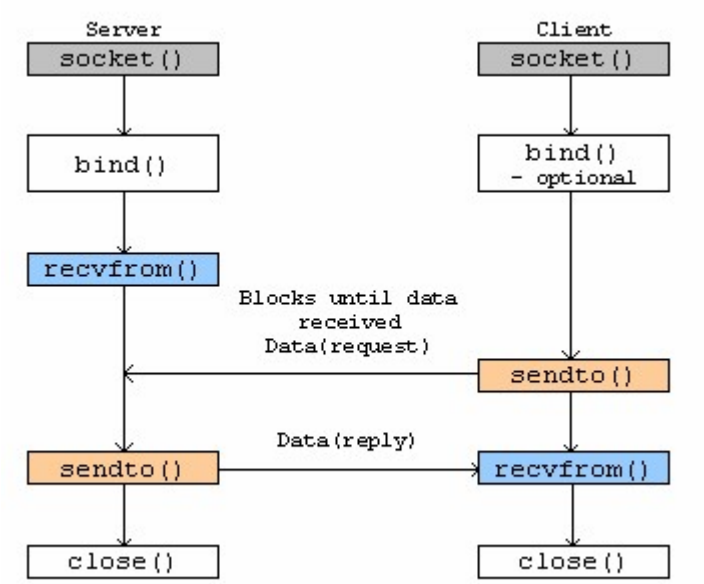
This is a simple C++ socket example with a client and echo server. This tutorial is based on simple server and client chat (linux) written by Hassan M. Yousuf.
I do not pretend bring a long explanation about sockets, you can find a lot of information in Google. I share a basic example to understand how it works.

Socket workflow



source: http://www.tenouk.com/Module39_files/image008.png

Server side

```
/*!
 * Simple socket program server.cpp
 * Version - 1.0.0
 * Based on: Simple chat program (server side).cpp -
http://github.com/hassanyf
 *
 * Copyleft (c) 2017 Rodrigo Tufino <rtufino@ups.edu.ec,
r.tufino@alumnos.upm.es>
 */

#include <iostream>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

using namespace std;
```

```
int main() {
 /* -------------- INITIALIZING VARIABLES -------------- */
 int server, client; // socket file descriptors
 int portNum = 2705; // port number
 int bufSize = 1024; // buffer size
 char buffer[bufSize]; // buffer to transmit
 bool isExit = false; // var fo continue infinitly

 /* Structure describing an Internet socket address. */
 struct sockaddr_in server_addr;
 socklen_t size;

 cout << "\n- Starting server..." << endl;

 /* ---------- ESTABLISHING SOCKET CONNECTION ----------*/

 server = socket(AF_INET, SOCK_STREAM, 0);

 /*
 * The socket() function creates a new socket.
 * It takes 3 arguments:
 * 1) AF_INET: address domain of the socket.
 * 2) SOCK_STREAM: Type of socket. a stream socket in
 * which characters are read in a continuous stream (TCP)
 * 3) Third is a protocol argument: should always be 0.
 * If the socket call fails, it returns -1.
 */

 if (server < 0) {
 cout << "Error establishing socket ..." << endl;
 exit(-1);
 }

 cout << "- Socket server has been created..." << endl;

 /*
 * The variable serv_addr is a structure of sockaddr_in.
 * sin_family contains a code for the address family.
 * It should always be set to AF_INET.
 * INADDR_ANY contains the IP address of the host. For
 * server code, this will always be the IP address of
 * the machine on which the server is running.
 * htons() converts the port number from host byte order
 * to a port number in network byte order.
 */

 server_addr.sin_family = AF_INET;
 server_addr.sin_addr.s_addr = htons(INADDR_ANY);
 server_addr.sin_port = htons(portNum);

 /*
```

```
 * This function is used to set the socket level for socket.
 * It is used to avoid blind error when reuse the socket.
 * For more info, see the url.
 *
http://stackoverflow.com/questions/5592747/bind-error-while-recreating
-socket
 */

 int yes = 1;
 if (setsockopt(server, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes))
== -1) {
 perror("setsockopt");
 exit(1);
 }
 /* ---------------- BINDING THE SOCKET --------------- */

 /*
 * The bind() system call binds a socket to an address,
 * in this case the address of the current host and port number
 * on which the server will run. It takes three arguments,
 * the socket file descriptor. The second argument is a pointer
 * to a structure of type sockaddr, this must be cast to
 * the correct type.
 */

 if ((bind(server, (struct sockaddr*) &server_addr,
sizeof(server_addr)))
 < 0) {
 cout
 << "- Error binding connection, the socket has already been
established..."
 << endl;
 exit(-1);
 }

 /* ------------------ LISTENING CALL ----------------- */

 size = sizeof(server_addr);
 cout << "- Looking for clients..." << endl;

 /*
 * The listen system call allows the process to listen
 * on the socket for connections.
 * The program will be stay idle here if there are no
 * incomming connections.
 * The first argument is the socket file descriptor,
 * and the second is the size for the number of clients
 * i.e the number of connections that the server can
 * handle while the process is handling a particular
 * connection. The maximum size permitted by most
 * systems is 5.
```

```
*/

listen(server, 1);

/* ------------------- ACCEPT CALL ------------------ */

client = accept(server, (struct sockaddr *) &server_addr, &size);

/*
* The accept() system call causes the process to block
* until a client connects to the server. Thus, it wakes
* up the process when a connection from a client has been
* successfully established. It returns a new file descriptor,
* and all communication on this connection should be done
* using the new file descriptor. The second argument is a
* reference pointer to the address of the client on the other
* end of the connection, and the third argument is the size
* of this structure.
*/

if (client < 0)
cout << "- Error on accepting..." << endl;

string echo;
while (client > 0) {
// Welcome message to client
strcpy(buffer, "\n-> Welcome to echo server...\n");
send(client, buffer, bufSize, 0);
cout << "- Connected with the client, waiting for data..." << endl;
// loop to recive messages from client
do {
cout << "\nClient: ";
echo = "";
/*
* A send operation from client is done for each word
* has written on it's terminal line. We need a special
* character to stop transmission and this loop works
* until this char ('*') arrives.
*/
do {
// wait the request from client
recv(client, buffer, bufSize, 0);
cout << buffer << " ";
// verify if client does not close the connection
if (*buffer == '#') {
// exit loop and say goodbye (It's a polite server :D)
isExit = true;
*buffer = '*';
echo = "Goodbye!";
} else if ((*buffer != '#') && (*buffer != '*')) {
// concatenate the echo string to response to the client
```

```
 echo += buffer;
 echo += " ";
 }
 } while (*buffer != '*');
 // copy the echo string to the buffer
 sprintf(buffer, "%s", echo.c_str());
 // send the message to the client
 send(client, buffer, bufSize, 0);
 } while (!isExit);

 /* ---------------- CLOSE CALL ------------- */
 cout << "\n\n=> Connection terminated with IP "
 << inet_ntoa(server_addr.sin_addr);
 close(client);
 cout << "\nGoodbye..." << endl;
 exit(1);

 }

 /* ---------------- CLOSE CALL ------------- */
 close(server);
 return 0;

}
```

Client side

```
/*!
 * Simple socket program client.cpp
 * Version - 1.0.0
 * Based on: Simple chat program (client side).cpp -
http://github.com/hassanyf
 *
 * Copyleft (c) 2017 Rodrigo Tufino <rtufino@ups.edu.ec,
r.tufino@alumnos.upm.es>
 */
#include <iostream>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <netdb.h>

using namespace std;

int main() {
 /* -------------- INITIALIZING VARIABLES -------------- */
 int client; // socket file descriptors
 int portNum = 2705; // port number (same that server)
 int bufsize = 1024; // buffer size
```

```
 char buffer[bufsize]; // buffer to transmit
 char ip[] = "127.0.0.1"; // Server IP
 bool isExit = false; // var fo continue infinitly

 /* Structure describing an Internet socket address. */
 struct sockaddr_in server_addr;

 cout << "\n- Starting client..." << endl;

 /* ---------- ESTABLISHING SOCKET CONNECTION ----------*/

 client = socket(AF_INET, SOCK_STREAM, 0);

 /*
 * The socket() function creates a new socket.
 * It takes 3 arguments:
 * 1) AF_INET: address domain of the socket.
 * 2) SOCK_STREAM: Type of socket. a stream socket in
 * which characters are read in a continuous stream (TCP)
 * 3) Third is a protocol argument: should always be 0.
 * If the socket call fails, it returns -1.
 */

 if (client < 0) {
 cout << "\n-Error establishing socket..." << endl;
 exit(-1);
 }

 cout << "\n- Socket client has been created..." << endl;

 /*
 * The variable serv_addr is a structure of sockaddr_in.
 * sin_family contains a code for the address family.
 * It should always be set to AF_INET.
 * INADDR_ANY contains the IP address of the host. For
 * server code, this will always be the IP address of
 * the machine on which the server is running.
 * htons() converts the port number from host byte order
 * to a port number in network byte order.
 */

 server_addr.sin_family = AF_INET;
 server_addr.sin_port = htons(portNum);

 /*
 * This function converts an Internet address (either IPv4 or IPv6)
 * from presentation (textual) to network (binary) format.
 * If the comunication is on the same machine, you can comment this
line.
 */
 inet_pton(AF_INET, ip, &server_addr.sin_addr);
```

```
 /* ---------- CONNECTING THE SOCKET ---------- */

 if (connect(client, (struct sockaddr *) &server_addr,
sizeof(server_addr))
 < 0)
 cout << "- Connection to the server port number: " << portNum <<
endl;

 /*
 * The connect function is called by the client to
 * establish a connection to the server. It takes
 * three arguments, the socket file descriptor, the
 * address of the host to which it wants to connect
 * (including the port number), and the size of this
 * address.
 * This function returns 0 on success and -1
 * if it fails.
 * Note that the client needs to know the port number of
 * the server but not its own port number.
 */
 cout << "- Awaiting confirmation from the server..." << endl; //line
40

 // recive the welcome message from server
 recv(client, buffer, bufsize, 0);
 cout << buffer << endl;

 cout << "- Connection confirmed, you are good to go!" << endl;
 cout << "- Enter * to end the message" << endl;
 cout << "- Enter # to end the connection\n" << endl;
 // loop to send messages to server
 do {
 cout << "Message: ";
 /*
 * The function 'cin' get an word at time and send it
 * to the server. The send operation is call until
 * the user write '*'.
 */
 do {
 // read from terminal
 cin >> buffer;
 // send to the server
 send(client, buffer, bufsize, 0);
 if (*buffer == '#') {
 // exit from the loop
 *buffer = '*';
 isExit = true;
 }
 } while (*buffer != '*');
 // wait the response from the server
 cout << "Server says: ";
```

```
 recv(client, buffer, bufsize, 0);
 // print the server message
 cout << buffer << endl;
 } while (!isExit);

 /* ---------------- CLOSE CALL ------------- */

 cout << "\nConnection terminated.\n";

 /*
 * Once the server presses # to end the connection,
 * the loop will break and it will close the server
 * socket connection and the client connection.
 */
 close(client);
 return 0;
}
```