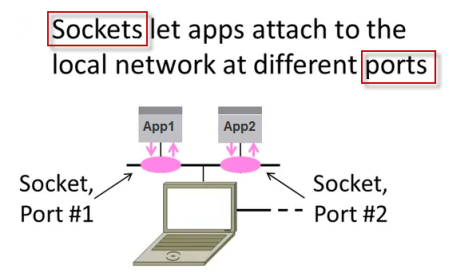
Socket



Server/Client Applications

The basic mechanisms of client-server setup are:

1. A client app send a request to a server app.
2. The server app returns a reply.
3. Some of the basic data communications between client and server are:
   1. File transfer - sends name and gets a file.
   2. Web page - sends url and gets a page.
   3. Echo - sends a message and gets it back.

Server Socket

1. create a **socket** - Get the file descriptor!
2. **bind** to an address -What port am I on?
3. **listen** on a port, and wait for a connection to be established.
4. **accept** the connection from a client.
5. **send/recv** - the same way we read and write for a file.
6. **shutdown** to end read/write.
7. **close** to releases data.

Client Socket

1. create a **socket**.
2. **bind\*** - this is probably be unnecessary because you're the client, not the server.
3. **connect** to a server.
4. **send/recv** - repeat until we have or receive data
5. **shutdown** to end read/write.
6. **close** to releases data.

Client/Server

The **client-server model** distinguishes between applications as well as devices. **Network clients make requests to a server by sending messages**, and **servers respond to their clients by acting on each request and returning results**.

For example, let's talk about **telnet**.   
When we connect to a remote host on port 23 with telnet (the client), a program on that host (called **telnetd**, the server) springs to life. It handles the incoming telnet connection, sets us up with a login prompt, etc.

One server generally supports numerous clients, and multiple servers can be networked together in a pool to handle the increased processing load as the number of clients grows.

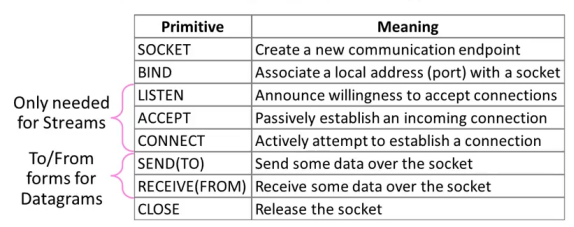
Some of the most popular applications on the Internet follow the client-server model including email, FTP and Web services. Each of these clients features a user interface and a client application that allows the user to connect to servers. In the case of email and FTP, users enter a computer name (or an IP address) into the interface to set up connections to the server.

The steps to establish a socket on the **server** side are:

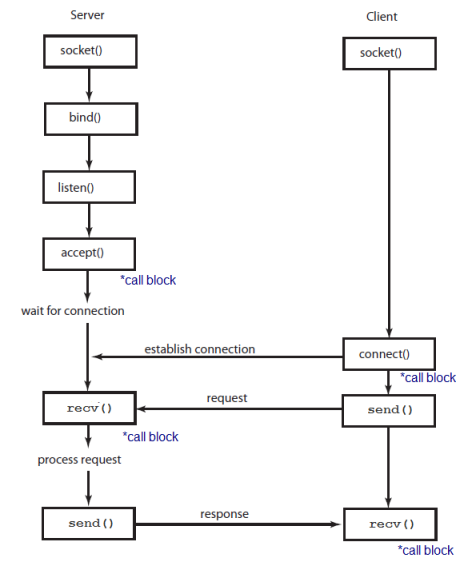
1. Create a socket with the **socket()** system call.
2. The server process gives the socket a name. In linux file system, local sockets are given a filename, under /tmp or /usr/tmp directory. For network sockets, the filename will be a service identifier, port number, to which the clients can make connection. This identifier allows to route incoming connections (which has that the port number) to connect server process. A socket is named using **bind()** system call.
3. The server process then waits for a client to connect to the named socket, which is basically listening for connections with the **listen()** system call. If there are more than one client are trying to make connections, the **listen()** system call make a queue.  
   The machine receiving the connection (the server) must bind its socket object to a known port number. A port is a 16-bit number in the range 0-65535 that's managed by the operating system and used by clients to uniquely identify servers. Ports 0-1023 are reserved by the system and used by common network protocols.
4. Accept a connection with the **accept()** system call. At **accept()**, a new socket is created that is distinct from the named socket. This new socket is used solely for communication with this particular client.  
   For TCP servers, the socket object used to receive connections is not the same socket used to perform subsequent communication with the client. In particular, the **accept()**system call returns a new socket object that's actually used for the connection. This allows a server to manage connections from a large number of clients simultaneously.
5. Send and receive data.
6. The named socket remains for further connections from other clients. A typical web server can take advantage of multiple connections. In other words, it can serve pages to many clients at once. But for a simple server, further clients wait on the listen queue until the server is ready again.

The steps to establish a socket on the **client** side are:

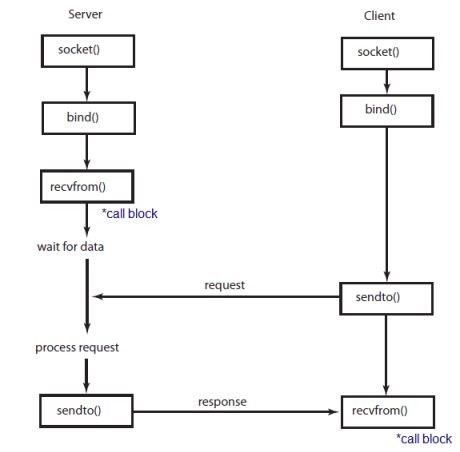
1. Create a socket with the **socket()** system call.
2. Connect the socket to the address of the server using the **connect()** system call.
3. Send and receive data. There are a number of ways to do this, but the simplest is to use the **read()** and **write()** system calls.



**TCP communication**



**UDP communication** - clients and servers don't establish a **connection** with each other



\*call block, go to [Blocking socket vs non-blocking socket](https://www.bogotobogo.com/cplusplus/sockets_server_client.php#block_vs_non_blocking).