UNIX SOCKET - CLIENT SERVER MODEL

Most of the Net Applications use the Client-Server architecture, which refers to two processes or two applications that communicate with each other to exchange some information. One of the two processes acts as a client process, and another process acts as a server.

Client Process

This is the process, which typically makes a request for information. After getting the response, this process may terminate or may do some other processing.

Example, Internet Browser works as a client application, which sends a request to the Web Server to get one HTML webpage.

Server Process

This is the process which takes a request from the clients. After getting a request from the client, this process will perform the required processing, gather the requested information, and send it to the requestor client. Once done, it becomes ready to serve another client. Server processes are always alert and ready to serve incoming requests.

Example: Web Server keeps waiting for requests from Internet Browsers and as soon as it gets any request from a browser, it picks up a requested HTML page and sends it back to that Browser.

Note that the client needs to know the address of the server, but the server does not need to know the address or even the existence of the client prior to the connection being established. Once a connection is established, both sides can send and receive information.

2-tier and 3-tier architectures

There are two types of client-server architectures:

- **2-tier architecture**: In this architecture, the client directly interacts with the server. This type of architecture may have some security holes and performance problems. Internet Explorer and Web Server work on two-tier architecture. Here security problems are resolved using Secure Socket Layer *SSL*.
- **3-tier architectures**: In this architecture, one more software sits in between the client and the server. This middle software is called 'middleware'. Middleware are used to perform all the security checks and load balancing in case of heavy load. A middleware takes all requests from the client and after performing the required authentication, it passes that request to the server. Then the server does the required processing and sends the response back to the middleware and finally the middleware passes this response back to the client. If you want to implement a 3-tier architecture, then you can keep any middleware like Web Logic or WebSphere software in between your Web Server and Web Browser.

Types of Server

There are two types of servers you can have:

- **Iterative Server**: This is the simplest form of server where a server process serves one client and after completing the first request, it takes request from another client. Meanwhile, another client keeps waiting.
- **Concurrent Servers**: This type of server runs multiple concurrent processes to serve many requests at a time because one process may take longer and another client cannot wait for so long. The simplest way to write a concurrent server under Unix is to *fork* a child process to handle each client separately.

How to Make Client

The system calls for establishing a connection are somewhat different for the client and the server,

but both involve the basic construct of a socket. Both the processes establish their own sockets.

The steps involved in establishing a socket on the client side are as follows:

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

How to make a Server:

The steps involved in establishing a socket on the server side are as follows:

- Create a socket with the *socket* system call.
- Bind the socket to an address using the **bind** system call. For a server socket on the Internet, an address consists of a port number on the host machine.
- Listen for connections with the **listen** system call.
- Accept a connection with the **accept** system call. This call typically blocks the connection until a client connects with the server.
- Send and receive data using the **read** and **write** system calls.

Client and Server Interaction

Following is the diagram showing the complete Client and Server interaction: