Communication in Client Server System:

Socket:

- A **Socket** is defined as an end point of communication.
- A pair of processes communication with each other involves a pair of sockets • Sockets are combination of **IP addresses and port numbers**.
- Sockets implements specific services (Telnet, FTP, and HTTP) to specific ports (23, 21, and 80 respectively).
- When client process requests a connection, a specific port is allotted by the host computer for it. This port number can be any arbitrary integer greater than 1024. Port numbers less than 1024 are reserved for conventional applications.
  (For example, if a client on host X with IP address 192.168.5.8 wishes to connect FTP server (at port no 21) at server system having IP address 192.168.3.20; then client X may allot port 1234 for this communication. Then connection for this communication will be established through two sockets (192.168.5.8:1234) at the client X and (192.168.3.20:21) at the server side.)

Client X (192.168.5.8) FTP Server (192.168.3.20)

Socket
192.168.3.20:21

Socket
192.168.5.8:1234

- All connections should be unique. So if another process at client X wants to establish another link with the same FTP server, then it should be allotted a separate port number other than 1234 and greater than 1024.
- Sockets are considered to be low level form of communication as it exchanges unstructured stream of bytes. It is the responsibility of client or server to impose a structure to these data.

Remote Procedure Call (RPC):

- Remote procedure call or RPC supports abstract procedure call between process running on the distributed environment connected through network.
- In RPC, the client process calls some procedure or function which is not defined in the client process itself; but defined in a remote server process. So, to execute that procedure, the client frames a request to the server in form of a message containing the name of the procedure and argument values. Server executes the procedure locally and again sends back the response in form of message containing the return value generated by the procedure.
- The messages exchanged for RPC are well structured and they are addressed to a RPC peer listening to a port on a different system.

Waiting for result

Client

Call remote Return from

procedure call

Request Response

Server Call local procedure Time return result

- Client process generates the message through a special process called **stub**. Client stub process performs the following tasks:
  - ✓ Puts the argument in a message
  - ✓ Sends the message to the server
  - ✓ Waits for the reply
  - ✓ Unpacks the result
  - ✓ Returns result to the caller application
- There may be different type of machines present in the network. For successful RPC among them **Marshalling** is required. Marshalling is the process of packing parameters in the message. The reverse process of unpacking information from message is known as **Un marshalling**.
- Like in client machine, in server side also there is server stub to create, route and unpack information into messages as well as to call the requested procedure locally. These server side stub procedures are known as **Skeletons.**
- Steps of RPC:

  (1) Client application calls to remote procedure
  (2) Client stub builds the message (marshalling)
  (3) Message is sent across the network to the server
  (4) Server OS hands over the message to server stub or skeleton
  (5) Skeleton unpacks the message (un-marshalling)
  (6) Stub makes local call to the procedure
  (7) Procedure executes and returns the result to skeleton
  (8) Skeleton composes response message with the returned value(marshalling) (9)
  Message is sent across the network back to the client
  (10) Client OS passes reply message to client stub
  (11) Client stub unpacks the message (un-marshalling) and returns result to the caller application.

1

11

2

10

Stub

K = add(a,b)

Server Machine

6

Proc: add

Server Process

7

int: val(a)

Implementation of add()

int: val(b)

5

K = add(a,b)

8

Client Operating System

Proc: add

int: val(a)

Server Operating

4

int: val(b)

System Network

Skeleton Client

3

Client Machine

Client Process

9

Remote Method Invocation (RMI):

- In RPC one client process calls a remote server procedure. In case of Remote Method Invocation (RMI) a client process calls another remote object located in the server using Object Oriented Technique.
- RMI is famous concept largely implemented in Java
- Marshalling is easy as both server and client interacts via JVM.
- It requires the object sent as request or response be serializable.
- Steps of RMI:
    - Like RPC, in RMI also the client has a stub (here referred as **Proxy**) for each remote class instance to marshal arguments as well as results and un-marshal received data.
    - Proxies communicate through dispatcher.

Client

Object A Proxy for B

Request

Response

Communication
Module

Remote reference model

Skeleton and
Dispatcher for B's
class
Server

Remote object B