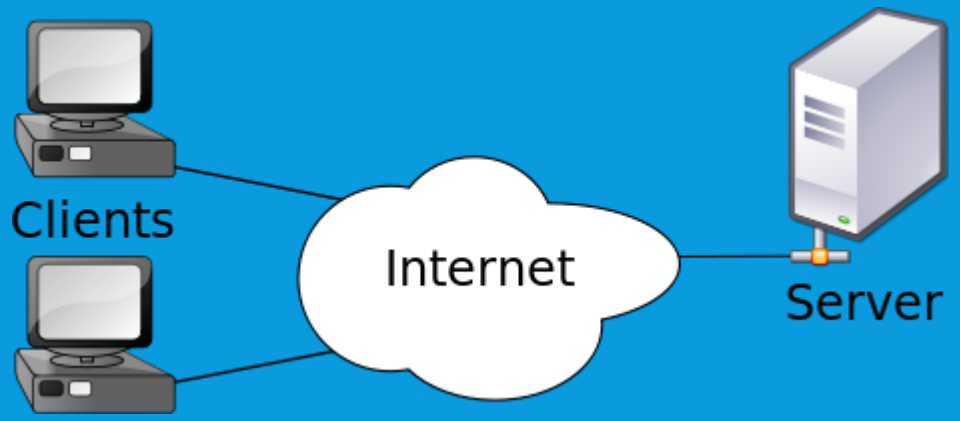
SOCKET PROGRAMMING



Bhupendra Pratap Singh

ACTS, CDAC, Pune

WHY SOCKET PROGRAMMING?

- To build any networked application in context of the Internet
- **Basic examples**
  - WWW (Firefox ,Internet Explorer) – Fetching web pages from server and display
  - File Transfer Protocol (FTP) to Transfer files
  - Some popular file sharing tools like – LimeWire (free peer-to-peer file sharing (P2P) client for Windows, OS X, Linux) & Bitcomet (Windows)

CONT….

- To understand how the data flows between the two entities over the Internet or in any networked application Socket Programming is best to provide the internal/background insights.

WHAT IS SOCKET?

- A ***socket*** is one endpoint of a two-way communication link between two programs/application processes running on the network. 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.
- A server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.
- An endpoint is a combination of an IP address and a port number
- Socket = **IP Address + Port Number**

LET US UNDERSTAND IT BETTER WITH THE TELEPHONIC ANALOGY

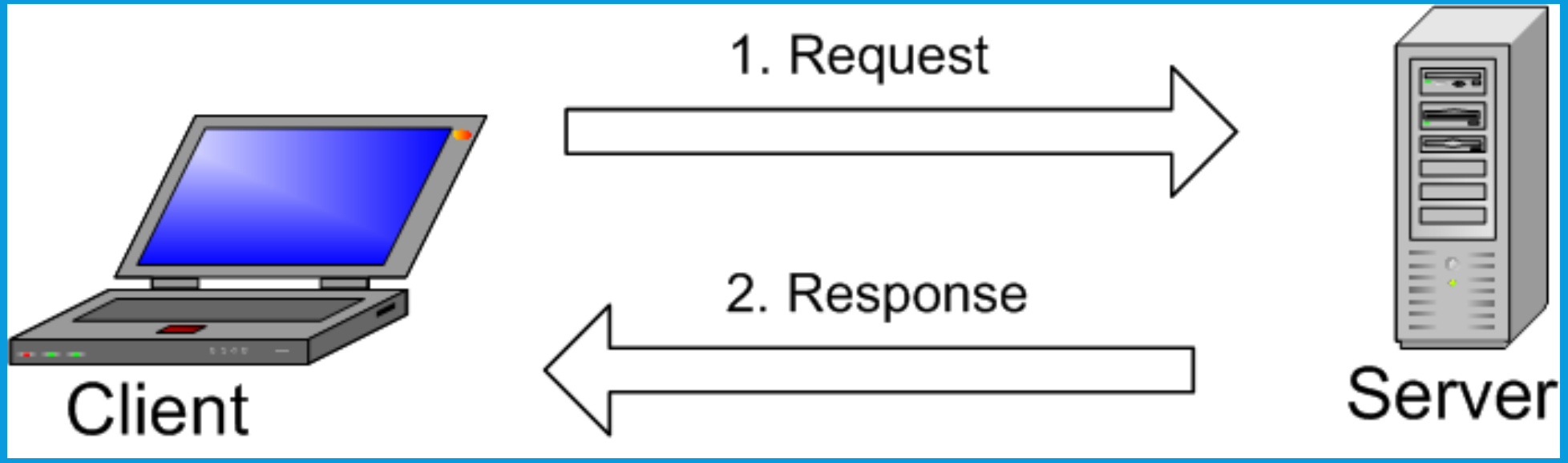
- A telephone call over a "Telephony Network " works as follows
- Both Parties have telephone installed or equipped with Mobile Phones.
- A Phone Number is assigned to individual entities.
- Caller lifts the telephone/mobile phone and dials a number.
- Telephone/Cell Phone rings and receiver picks/receive the call.
- Both parties talk and exchange data.
- Once conversation is over they hang up the phone

SOCKETS – SOME TERMINOLOGY/CALLS

- Socket() – Endpoint for communication
- Bind() – binds IP address and Port Number (Assign a unique telephone number)
- Listen() – Wait for the client(wait for a caller)
- Connect() – connect to the server (Dial a number)
- Accept() - waits for incoming connections (Receive a call)
- Send(), Recv()/ Read(), Write() – (Exchange of data) – (talk)
- Close –Close the connection (Hang-up the call)

THE CLIENT SERVER MODEL

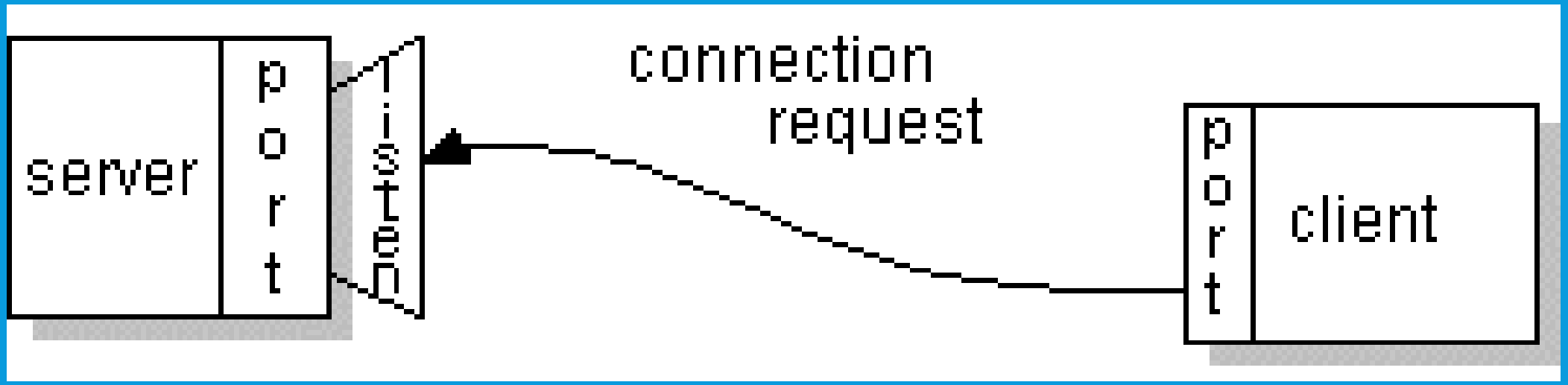
- SERVER – Provider of information
- CLIENT – Seeker of Information



ROLES OF CLIENT

- 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 connect 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.

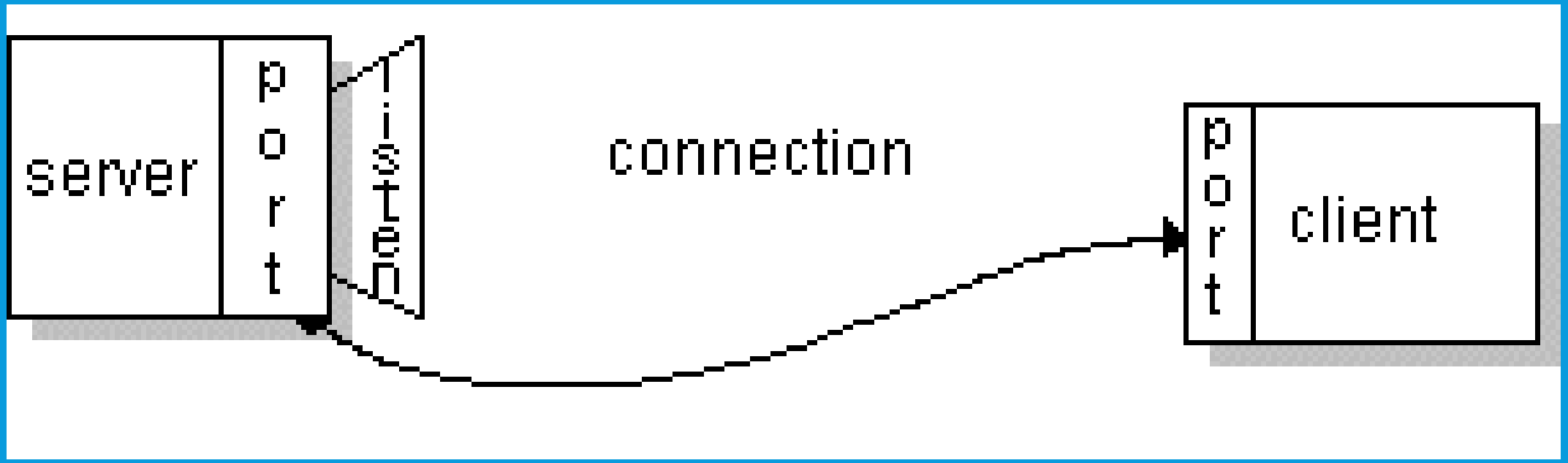
PICTORIAL REPRESENTATION



ROLES OF SERVER

- 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.

PICTORIAL REPRESENTATION



TYPES OF SERVER

- **There are Two kind of servers –**
  - Iterative server
  - Concurrent server

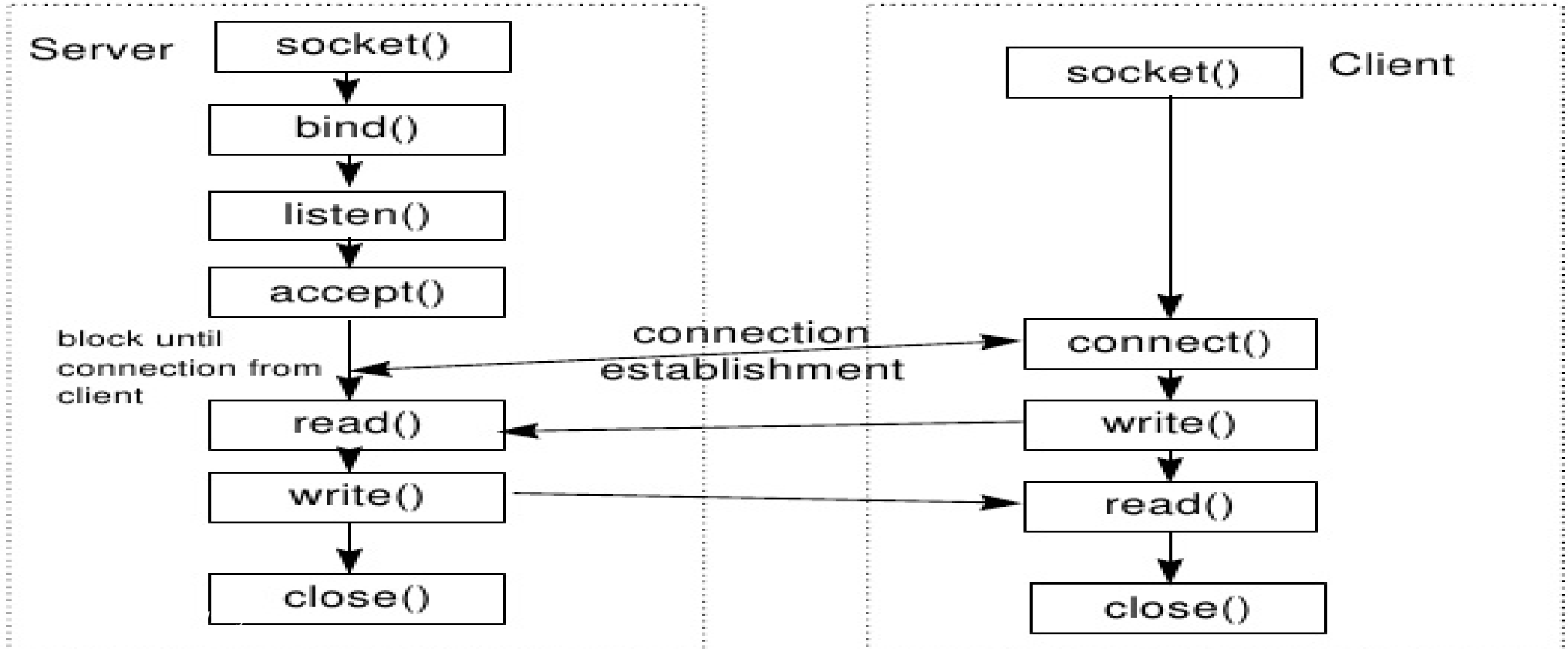
- **ITERATIVE SERVERS:**

  Server process serves one client at a time 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 can not wait for so long. There are multiple ways to write theses kind of servers, simplest method is with fork() system call.

SERVER-CLIENT FLOW (TCP ORIENTED)



UDP (CLIENT-SERVER FLOW)

DATA STRUCTURES

```
struct sockaddr
 {
   unsigned short sa_family;      // address family, AF_xxx (IPV4/IPV6)
   char           sa_data[14]; // 14 bytes of protocol address (IP addr+PORT)
};
```

CONT.....

```
struct sockaddr_in
{
        sa_family_t    sin_family; /* address family: AF_INET */
        in_port_t      sin_port;   /* port in network byte order */
        struct in_addr sin_addr;   /* internet address */
 };
//  sin_family is always set to AF_INET (Address Family_IPV4) AF_INET6 (IPV6)
//Network Byte order will always be big endian
```

CONT…

- /* Internet address. */
- struct in_addr {
- uint32_t s_addr; /* address in network byte order */
- };

SOME FUNCTIONS FOR BYTE ORDERING

- ntohs() Convert a 16-bit quantity from network byte order to host byte order (big-Endian to little-Endian).
- ntohl() Convert a 32-bit quantity from network byte order to host byte order (big-Endian to little-Endian).
- htons() Convert a 16-bit quantity from host byte order to network byte order (little-Endian to big-Endian).
- htonl() Convert a 32-bit quantity from host byte order to network byte order (little-Endian to big-Endian).

SOCKET()

- int socket(int domain, int type, int protocol);
- domain – AF_INET (IPV4 Communcation) // AF_INET6 – IPV6
- Type – SOCK_STREAM (TCP), SOCK_DGRAM (UDP)
- Protocol – 0 (integer value)
- the third argument to socket is generally an int indicating the protocol. 0 indicates that the caller does not want to specify the protocol and will leave it up to the service provider. Other than zero, another common one is IPPROTO_TCP.
- Refer man 2 socket (more details)

BIND()

- int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
- sockfd – socket descriptor
- addr – pointer to valid sockaddr structure cast as a sockaddr* pointer
- addrlen – length of sockaddr structure

LISTEN()

- int listen(int sockfd, int backlog);

- backlog – maximum length of the pending connection queue.

- Eg.

listen(sockfd,5) – this will allow maximum 5 connections in the pending queue. More than this connection will be refused.

ACCEPT()

int accept4(int sockfd, struct sockaddr *addr, socklen_t *addrlen, int flags);

The accept() system call is used with connection-based socket types (SOCK_STREAM, SOCK_DGRAM). It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. The newly created socket is not in the listening state. The original socket sockfd is unaffected by this call.

(explore flag from man 2 listen()

addrlen – pointer to the size of client structure

CONNECT()

- int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
- addrlen - (size of the addr struct)
- addr – contains all the remote server details

- (Explore send(), recv(), read(), write() from man page, to send/receive and reading/writing of data.

CLOSE() – BYE BYE .....

- int close (int sockfd)

- Eg .

```
close(sockfd);
```

The best way to predict the future is to create it.

- Peter Drucker

THANK YOU !! ------

Bhupendra Pratap Singh
CDAC, ACTS, Pune