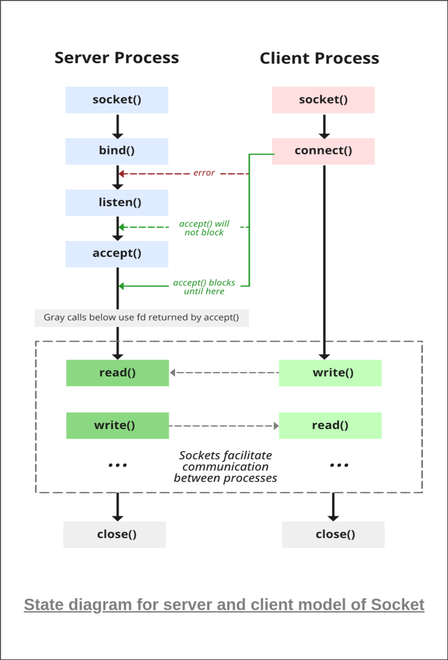
**Theoretical Background on Socket Programming**

**What is Socket Programming?**

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server.

**State Diagram for Server and Client Model**



**Stages for Server**

The server is created using the following steps:

**1. Socket Creation**

int sockfd = socket(domain, type, protocol)

* **sockfd:** socket descriptor, an integer (like a file handle)
* **domain:** integer, specifies communication domain. We use AF\_ LOCAL as defined in the POSIX standard for communication between processes on the same host. For communicating between processes on different hosts connected by IPV4, we use AF\_INET and AF\_I NET 6 for processes connected by IPV6.
* **type:** communication type  
  SOCK\_STREAM: TCP(reliable, connection-oriented)  
  SOCK\_DGRAM: UDP(unreliable, connectionless)
* **protocol:**Protocol value for Internet Protocol(IP), which is 0. This is the same number that appears on the protocol field in the IP header of a packet.(man protocols for more details)

### ****2. Bind****

int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

After the creation of the socket, the bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to the localhost, hence we use INADDR\_ANY to specify the IP address.

### ****3. Listen****

int listen(int sockfd, int backlog);

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

### ****5. Accept****

int new\_socket= accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);

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. At this point, the connection is established between client and server, and they are ready to transfer data.

## ****Stages for Client****

**1. Socket connection:** Exactly the same as that of server’s socket creation

**2. Connect:**The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server’s address and port is specified in addr.

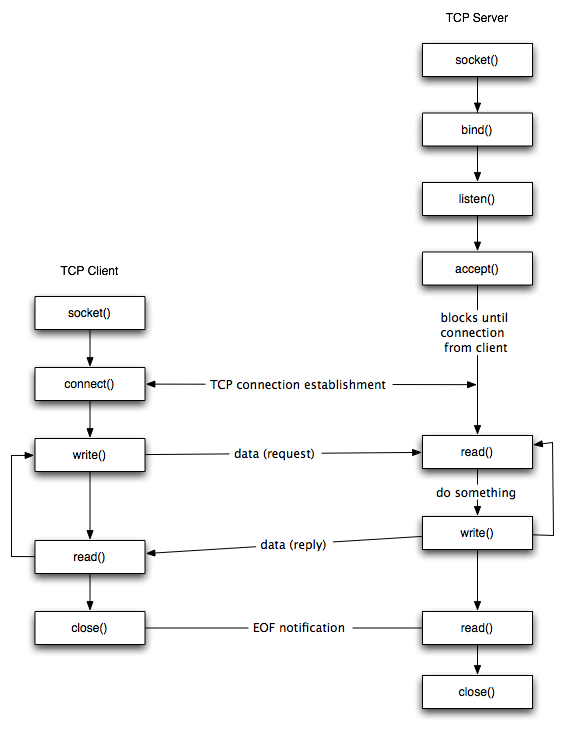
int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

**Theoretical Background on TCP,UDP**

### TCP

### TCP is suited for applications that require high reliability, and transmission time is relatively less critical. It is used by other protocols like HTTP, HTTPs, FTP, SMTP, Telnet. TCP rearranges data packets in the order specified. There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent. TCP does Flow Control and requires three packets to set up a socket connection before any user data can be sent. TCP handles reliability and congestion control. It also does error checking and error recovery. Erroneous packets are retransmitted from the source to the destination.

The sequence of function calls for the client and a server participating in a TCP connection is presented in Figure .



**For Sending and Receiving data in TCP**

1. SEND: int send(int sockfd, const void \*msg, int len, int flags);

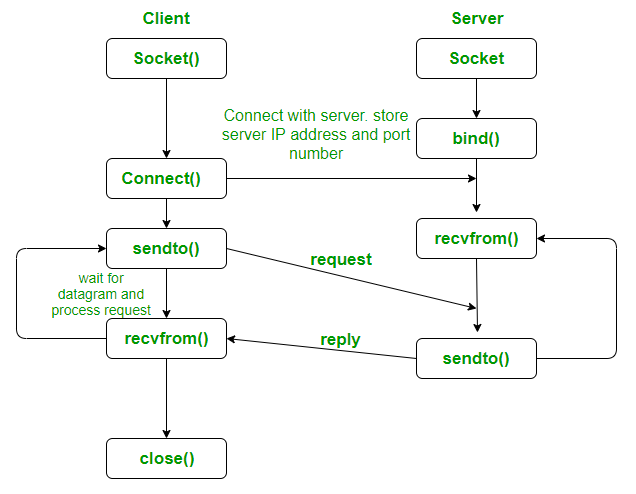
* msg: message you want to send
* len: length of the message
* flags := 0
* returned: the number of bytes actually sent

1. RECEIVE: int recv(int sockfd, void \*buf, int len, unsigned int flags);

* buf: buffer to receive the message
* len: length of the buffer (“don’t give me more!”)
* flags := 0
* returned: the number of bytes received

**UDP**

UDP is a connection less protocol. There is no connection is established between client and server. In UDP, the client does not form a connection with the server like in TCP and instead, It just sends a datagram. Similarly, the server need not to accept a connection and just waits for datagrams to arrive. We can call a function called **connect()** in UDP but it does not result anything like it does in TCP. There is no 3 way handshake. It just checks for any immediate errors and store the peer’s IP address and port number. connect() is storing peers address so no need to pass **server address** and **server address length** arguments in **sendto()**.



**For Sending and Receiving data in UDP**

1. SEND (DGRAM-style): int sendto(int sockfd, const void \*msg, int len, int flags, const struct sockaddr \*to, int tolen);

* msg: message you want to send
* len: length of the message
* flags := 0
* to: socket address of the remote process
* tolen: = sizeof(struct sockaddr)
* returned: the number of bytes actually sent

2. RECEIVE (DGRAM-style): int recvfrom(int sockfd, void \*buf, int len, unsigned int flags,struct sockaddr \*from, int \*fromlen);

* buf: buffer to receive the message
* len: length of the buffer (“don’t give me more!”)
* from: socket address of the process that sent the data
* fromlen:= sizeof(struct sockaddr)
* flags := 0
* returned: the number of bytes received