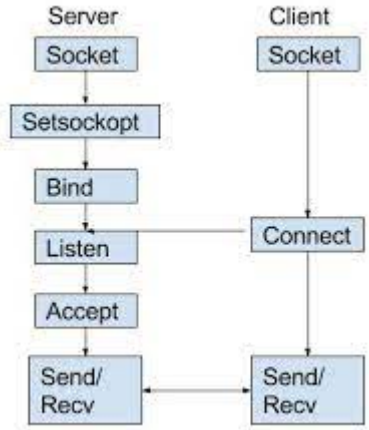
What is socket programming?

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 other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.



- **Socket creation:**

  `int sockfd = socket(domain, type, protocol)`

  sockfd: socket descriptor, an integer (like a file-handle)

  domain: integer, communication domain e.g., AF_INET (IPv4 protocol) , AF_INET6 (IPv6 protocol)

  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 which appears on protocol field in the IP header of a packet.(man protocols for more details)

Setsockopt:

```
int setsockopt(int sockfd, int level, int optname,
```

- ```
  const void *optval, socklen_t optlen);
  ```

  This helps in manipulating options for the socket referred by the file
  ```

descriptor sockfd. This is completely optional, but it helps in reuse of address and port. Prevents error such as: “address already in use”.

Bind:

```
int bind(int sockfd, const struct sockaddr *addr,
                         socklen_t addrlen);
```

After creation of the socket, 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.

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.

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, connection is established between client and server, and they are ready to transfer data.

Some terminologies

By default, a socket that has been created using socket() is **active**. An active socket can be used in a connect() call to establish a connection to a passive socket. This is referred to as performing an active open.

A **passive socke**t (also called a listening socket) is one that has been marked to allow incoming connections by calling listen(). Accepting an incoming connection is referred to as performing a passive open

TCP is a connection-oriented protocol. Connection-orientation means that the communicating devices should establish a connection before transmitting data and should close the connection after transmitting the data.

UDP is the Datagram oriented protocol. This is because there is no overhead for opening a connection, maintaining a connection, and terminating a connection. UDP is efficient for broadcast and multicast type of network transmission.

Encapsulation is an important principle of a layered networking protocol. The key idea of encapsulation is that the information (e.g., application data, a TCP segment, or an IP datagram) passed from a higher layer to a lower layer is treated as opaque data by the lower layer.

Stream Sockets (SOCK_STREAM)

– Connection oriented

– Rely on TCP to provide reliable two-way connected communication

Datagram Sockets (SOCK_DGRAM)

– Rely on UDP

– Connection is unreliable

Socket Structure

struct sockaddr: Holds socket address information for many types of sockets

```
struct sockaddr {
unsigned short sa_family; //address family AF_xxx
unsigned short sa_data[14]; //14 bytes of protocol addr
}
```

struct sockaddr_in: A parallel structure that makes it easy to reference elements of the socket address

sin_port and sin_addr must be in Network Byte order

Dealing with IP Addresses

```
struct in_addr {
unsigned long s_addr; // that's a 32bit long, or 4 bytes
};
```

To convert binary IP to string:

```
 inet_noa()
printf("%s,inet_ntoa(my_addr.sin_addr));
```

int connect(int sockfd, struct sockaddr *serv_addr, int addrlen)
– sockfd is the socket descriptor returned by socket()
– serv_addr is pointer to struct sockaddr that contains information on destination IP address and port
– addrlen is set to sizeof(struct sockaddr)
– returns -1 on error

int accept(int sockfd, void *addr, int *addrlen);
– sockfd is the listening socket descriptor
– information about incoming connection is stored in addr which is a pointer to a local struct sockaddr_in
– addrlen is set to sizeof(struct sockaddr_in)
– accept returns a new socket file descriptor to use

for this accepted connection and -1 on error

int send(int sockfd, const void *msg, int len, int
flags);
– sockfd is the socket descriptor you want to send data to
(returned by socket() or got from accept())
– msg is a pointer to the data you want to send
– len is the length of that data in bytes
– set flags to 0 for now
– sent() returns the number of bytes actually sent (may
be less than the number you told it to send) or -1 on
Error

int recv(int sockfd, void *buf, int len, int flags);
– sockfd is the socket descriptor to read from
– buf is the buffer to read the information into
– len is the maximum length of the buffer
– set flags to 0 for now
– recv() returns the number of bytes actually read into
the buffer or -1 on error
– If recv() returns 0, the remote side has closed
connection on you

.

References

1. https://www.geeksforgeeks.org/socket-programming-cc/#:~:text=What%20is%20socket%20programming%3F,reaches%20out%20to%20the%20server.
2. The linux programming interface (book by Michel kerrisk)
3. https://www.geeksforgeeks.org/fork-system-call/
4. https://www.geeksforgeeks.org/creating-multiple-process-using-fork/
5. http://home.iitk.ac.in/~chebrolu/ee673-f06/sockets.pdf
6.