POPL - II PROJECT
REPORT

Instructor : Saurabh Joshi
Group Members:
Aayush Arora - ee17btech11003
Pooja Mate - cs17btech11027
Aakash Daswani - ee17btech11002
Aarushi - cs17btech11046

Task:

Our task is to implement end-to-end connectivity between host server and client.

Design of the program:

How to use sockets?

Setup socket

- Where is the remote machine (IP address, hostname)
- What service gets the data (port)

Send and Receive

- Designed just like any other I/O in unix
- send -- write

- recv -- read

Close the socket

1. Setup Socket:

Both client and server need to setup the socket. It requires the domain , type and the protocol.A socket address structure is always passed by reference as an argument to any socket functions. But any socket function that takes one of these pointers as an argument must deal with socket address structures from any of the supported protocol families.

- Domain
  AF_INET -- IPv4 (AF_INET6 for IPv6)
- Type
  SOCK_STREAM -- TCP
  SOCK_DGRAM -- UDP
- Protocol
  0
- For example,
  int sockfd = socket(AF_INET, SOCK_STREAM, 0);

2. Server Binding

The bind() assigns a local protocol address to a socket. With the Internet protocols, the address is the combination of an IPv4 or

IPv6 address (32-bit or 128-bit) address along with a 16 bit TCP port number. Only server need to bind.
bind() takes in protocol-independent (struct sockaddr*).

- sockfd
  file descriptor socket() returned
- my_addr
  struct sockaddr_in for IPv4
  cast (struct sockaddr_in*) to (struct sockaddr*)

3.Connect Function:
The connect() function is used by a TCP client to establish a connection with a TCP server. The function returns 0 if the it succeeds in establishing a connection (i.e., successful TCP three-way handshake, -1 otherwise.

- connect(int sockfd, const struct sockaddr *saddr, socklen_t addrlen)

4.Listen Function:
The listen() function converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests directed to this socket.sockfd is the socket descriptor and backlog is the maximum number of connections the kernel should queue for this socket.
listen(int sockfd, int backlog);

- Sockfd

file descriptor socket() returned

- Backlog
  number of pending connections to queue

5. Accept Function:

The accept() is used to retrieve a connect request and convert that into a request. sockfd is a new file descriptor that is connected to the client that called the connect(). The cliaddr and addrlen arguments are used to return the protocol address of the client.

accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)

- sockfd

again... file descriptor socket() returned

- addr

pointer to store client address, (struct sockaddr_in *) cast to (struct sockaddr *)

- addrlen

pointer to store the returned size of addr, should be sizeof(*addr)

6. Send() Function:

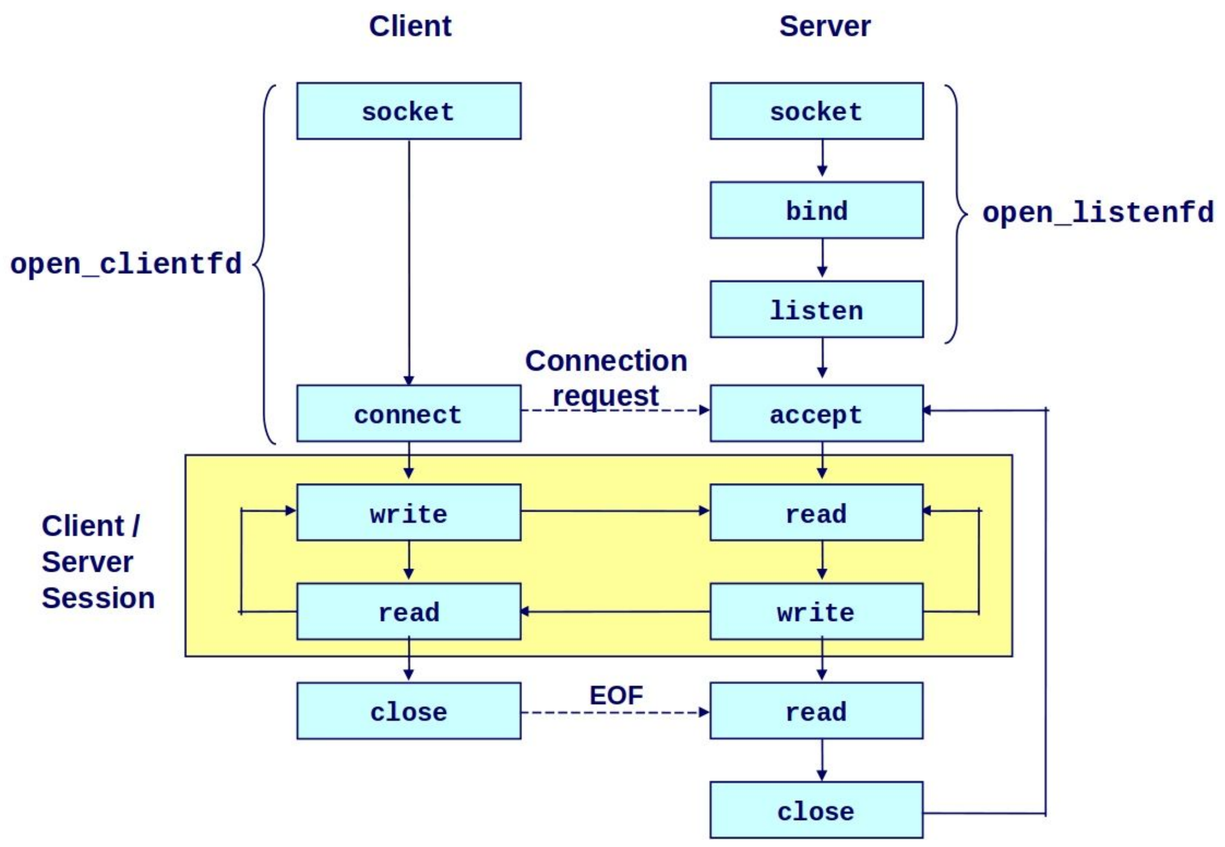
Since a socket endpoint is represented as a file descriptor, we can use read and write to communicate with a socket as long as it is connected.

7. Receive() Function:

The recv() function is similar to read(), but allows to specify some options to control how the data are received.The function returns the length of the message in bytes, 0 if no messages are available and peer had done an orderly shutdown, or -1 on error.

WORKFLOW

The client has the job of initiating contact with the server. In order for the server to be able to react to the client's initial contact, the server has to be ready. This implies two things. First, the server program can not be dormant; it must be running as a process before the client attempts to initiate contact. Second, the server program must have some sort of door (i.e., socket) that welcomes some initial contact from a client (running on an arbitrary machine).With the server process running, the client process can initiate a TCP connection to the server. This is done in the client program by creating a socket object. When the client creates its socket object, it specifies the address of the server process, namely, the IP address of the server and the port number of the process. Upon creation of the socket object, TCP in the client initiates a three-way handshake and establishes a TCP connection with the server. The three-way handshake is completely transparent to the client and server programs. During the three-way handshake, the client process knocks on the welcoming door of the server process. When the server "hears" the knocking, it creates a new door (i.e., a new socket) that is dedicated to that particular client. In our example below, the welcoming door is a ServerSocket object that we call the welcomeSocket. When a client knocks on this door, the program invokes welcomeSocket's accept() method, which creates a new door for the client. At the end of the handshaking phase, aTCP connection exists between the client's socket and the server's new socket. Henceforth, we refer to the new socket as the server's "connection socket".



A short request by a client may have to wait for longer requests to be completed? The server can be blocked on I/O while serving a request; this is inefficient!?

Solution is to have concurrent execution of functions inside both the client and server side. Multi-thread: one thread per function to be executed (dynamically created, or pre-created).When a client connects to the server, the server creates a new thread and listens to the socket. Then, if the client sends something, the server must read it concurrently via another thread.