**Homework 1**

**CMSC 621**

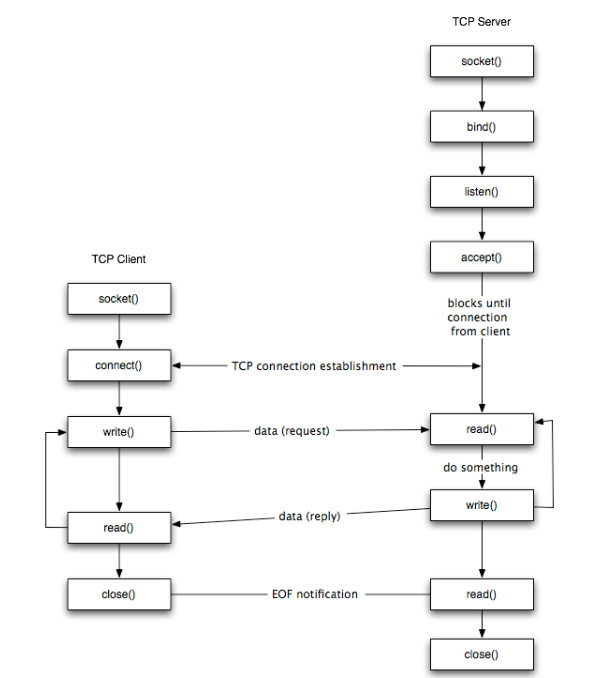
**Ajinkya Wakhale**

1.

When communicating using UNIX stream sockets, processes use the system calls socket (), accept (), bind(), listen(), and connect(). What is the purpose of each of these calls?

Answer:

Client server model is one of the most used communication paradigm in networked systems. Clients normally communicate with one server at a time. Client needs to know the address of the server but server does not need to know the address of the client prior to the connection established. Client and servers communicate by means of multiple layers’ protocols. The sequence of function calls for client and server participating is shown below.



As shown in the figure for client side,

1. Create a socket using socket method.
2. Connect to server using connect function.
3. Send and receive data by read and write.

For server side,

1. Create socket using socket function
2. Bind socket using bind function
3. Listen for connections using listen function
4. Accept connections using accept
5. Send and receive using send and receive methods.
6. socket () function- int socket ( int family, int type, int protocol).

It is called to create a socket for the specified domain and protocol. This creates an endpoint for a network connection.

Family parameter specifies protocol family, type is the constant describing type of socket (SOCK\_STREAM for stream sockets) and this function returns a non-negative number like file descriptor implying success and failure.

1. Connect() function- int connect(int sockfd, struct sockaddr \*servaddr, socklen\_t addrlen)

connect function is used by TCP client to establish connection with server. Sockfd is the the socket descriptor returned by the socket function. It returns zero if connection is established

1. Bind()- int bind(int sockfd, int sockfd, const struct sockaddr \*servaddr, socklen\_t addrlen )- bind() assigns a local protocol address to a socket. It is called to bind a path or internet address. With internet protocols , the combination of an IP4 or IP6 address along with 16 bit TCP port number. In UNIX domain, a connection is composed of one or two path names. With bind() function, processes attach itself to a specific port and IP address.
2. Listen()- int listen( int sock, int backlog)

Listen() system calls listen for connections on a socket. Listen() marks the socket referred to by sockfd as a passive socket, that is, as a socket that will be used to accept incoming connection requests using accept. For SOCK\_STREAM, the server call this function which specifies how many connection requests can be queued.

1. accept() function – int accept (int sockfd, struct sockaddr \*clientaddr, socklen\_t \*addrlen)

accept() is used to retrieve a connect request and convert that into a request. In this function, sockfd is new file descriptor that is connected to the client which called connect() and clientaddr and addrlen is used to return the protocol address of the client and it returns a new descriptor which is valid only for the particular connection.

Q.2

What are the differences between datagram sockets and stream sockets?

Answer:

Socket type define the communication properties visible to a user. The internet domain sockets provide access to TCP/IP transport protocols. Sockets exchange data only with sockets in the same domain.

**Stream sockets**- It allows processes to communicate using TCP/IP. It provides a bidirectional, reliable and unduplicated flow of data with no record boundaries. Once the connection is established, data can be read from and written to these sockets as byte stream. Socket type is SOCK\_STREAM.

**Datagram sockets**- They communicate through UDP. It supports bidirectional flow of messages. A process on a datagram socket may receive messages in different order from the sending sequence and may receive duplicate messages. Record boundaries in the data are preserved. Socket type is SOCK\_DGRAM

Q. 3

Solution-

Steps for the mediator process would be as follows:

1. create Socket
2. bind the socket
3. listen with the backlog of 2 as it can queue maximum two requests.
4. Then it accepts the connection from both processes.

**Mediator function:**

#define wait\_time 20;

Char data1[20], data2[20];

int process [2]; int nproc = 0;

int device\_Status;

struct sockaddr\_un mediator; /\* Struct to store socket name and type\*/

d =socket(STREAM); /\*new socket \*/

bind(d, &mediator, sizeof(mediator)); /\*bind the socket for unique identity \*/

listen(d,2); /\*listen with backlog of 2 \*/

while(nproc<2){ /\*Accepts incoming connections from client \*/

process[i] = accept (d, &mediator, sizeof(mediator));

nproc ++;

}

For i=0 to 2;

{

Int process\_select = select(process [0], process[1]);

If( process[i]== process\_select || process\_select == ‘BOTH’){

Recv(&process[i],&c,1);

Timer(wait\_time);

If(c==’r’) { /\*if data read is for requesting permission \*/

Write(“Granted”) // sending message that permission is granted

Data 1 = Recv(process[i],&c,1) // Checking the status

If(c!=’g’ || data1==-1){ // process is taking time

Timer(wait\_time); //wait for the process to send back the acknowledgment

Data2 =recv(process[i], &c,1); //checking whether acknowledgment is sent by process

If(c!=’g’|| data2 ==-1){ // Still no data

Close(process[i]); //Close the connection for this process

}

}

}

}

Q.4

Pipes and sockets are two kinds of communications abstractions available in UNIX. An alternative abstraction that might be considered is the “named pipe”. To use one, you would assign a pathname to a pipe when it was created. (Think of the named pipe as another kind of file, in addition to regular files, directories, symbolic links, etc.). Then, you could obtain a read or write descriptor for the pipe by simply opening the named pipe. Once two processes had the read and write descriptors, the behavior would be exactly as for a conventional pipe. Discuss the advantages and disadvantages of a named pipe relative both to a conventional pipe and to the socket mechanism.

Answer:

**Named pipe and Conventional pipe.**

1. Named pipe is a file based communication method. It is persistent on file system and hence process need not to run at the same time to pass messages while in conventional pipe exists as long as the program holding its file descriptor runs.
2. Processes using name pipe need not to be related. Only file name act as a contract between two processes. Whereas processes using conventional pipe needs to have parent child relationship. Two unrelated processes cannot communicate with conventional pipe.
3. Named pipe can have ownership and permission controls associated with pipe same as file whereas it is not needed in conventional pipe. The file name of the pipe serves as an address or contact between the processes for communication. If only one process writes to a named pipe and other process reads from the named pipe, then named pipe behaves exactly as an unnamed pipe between the two-related process.
4. Named pipe is very useful to send one-line requests.
5. One of the disadvantage is that multiple processes cannot use a single named pipe to send or receive multi-line messages, unless you define more complex protocol to control message interleaving.

**Named pipe and Sockets**

1. Named pipe facilitates communication on the local system while sockets support communication on local system as well as external systems using TCP/IP protocols.
2. Named pipe are FIFO structures whereas sockets provide bi-directional communication.
3. On the same socket, a process can attend multiple clients. Operating system supplies new socket descriptor for each client. Data always ends up with the correct process whereas in PIPE this functionality is not supported.
4. In Named pipe, data is a single continuous stream whereas sockets support packets as well as sequenced packets.