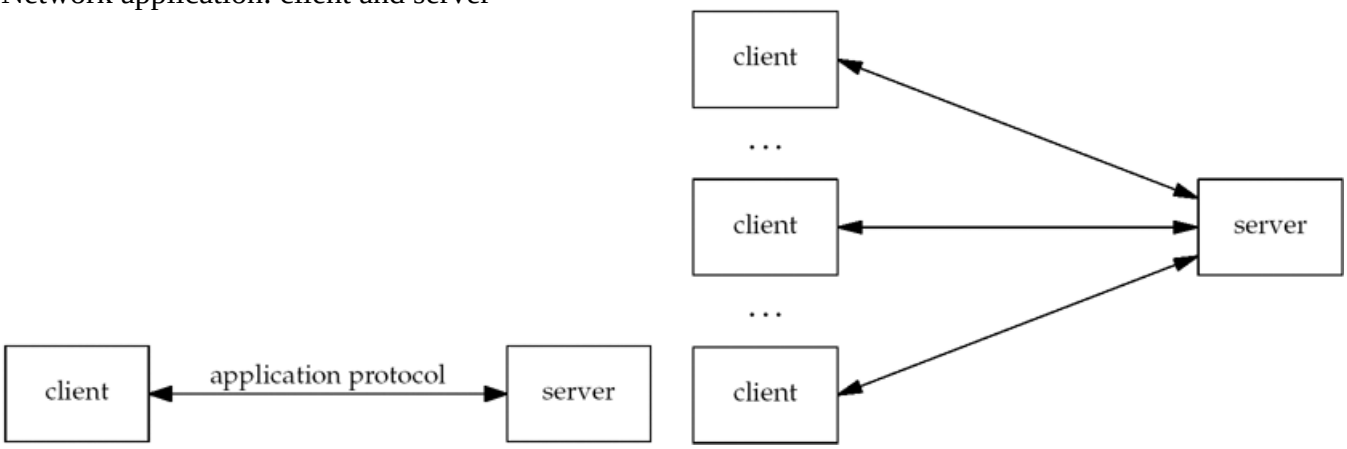
Network Programming
Unit 1

Q. What is network protocol? With a neat block diagram explain the network application for client and server.

- A network protocol is an established set of rules that determine how data is transmitted between different devices in the same network.
- Essentially, it allows connected devices to communicate with each other, regardless of any differences in their internal processes, structure or design.

Network application: client and server



- Clients normally communicate with one server at a time, although using a Web browser as an example, we might communicate with many different Web servers.
- But from the server's perspective, at any given point in time, it is not unusual for a server to be communicating with multiple clients.
- The client application and the server application may be thought of as communicating via a network protocol, but actually, multiple layers of network protocols are typically involved.

Q.List out the various approaches used to handle multiple clients at the same time.

Server can handle multiple clients by :

1. Multi threading
2. Socket Programming

Multi threading :

The simple way to handle multiple clients would be to spawn a new thread for every new client connected to the server.

Semaphore is simply a variable that is non-negative and shared between threads. This variable is used to solve the critical section problem and to achieve process synchronization in the multiprocessing environment.

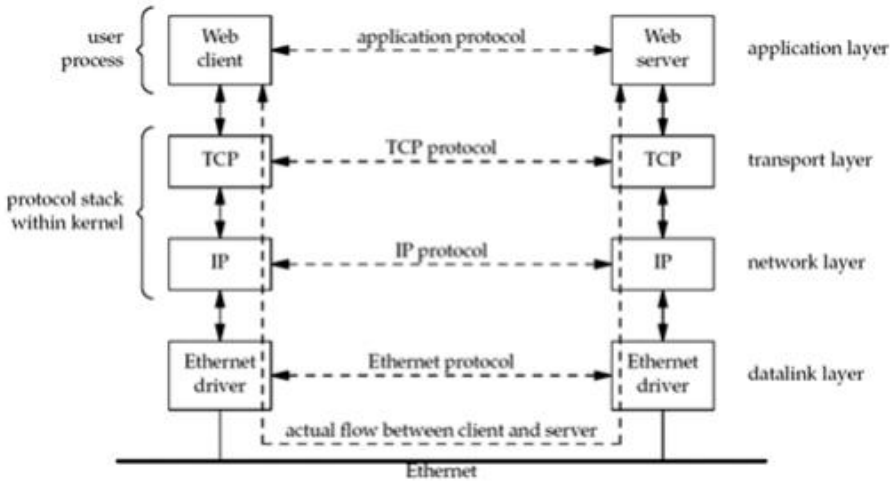
Socket Programming

Above method is strongly not recommended because of various disadvantages, namely:

- Threads are difficult to code, debug and sometimes they have unpredictable results.
- Overhead switching of context
- Not scalable for large number of clients
- Deadlocks can occur

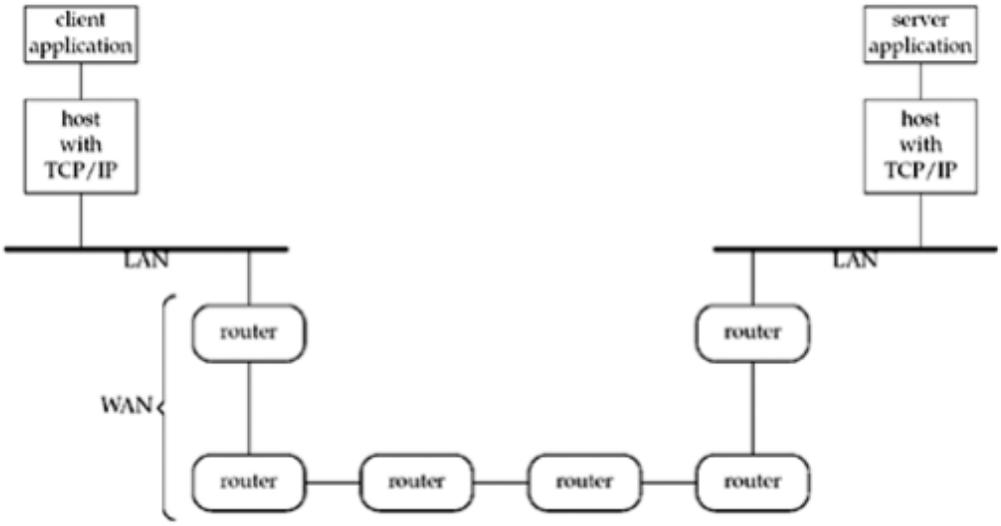
When you design a server program, plan for multiple concurrent processes. Special socket calls are available for that purpose they are called *concurrent servers*, as opposed to the more simple type of *iterative server*.

Q.With a neat block diagram, explain the client and server communication on Local Area Network using TCP.



- Even though the client and server communicate using an application protocol, the transport layers communicate using TCP.
- The actual flow of information between client and server goes down the protocol stack on one side, across the network, and up the protocol stack on the other side.
- Client & Server are typically user processes, while TCP and IP protocols are normally part of the protocol stack within the kernel.
- The four layers labeled in the diagram are the Application layer, Transport layer, Network layer, Data-link layer.
- Some clients and servers use the User Datagram Protocol (UDP) instead of TCP.

Q.With a neat block diagram, explain the client and server communication over Wide Area Network using TCP.



- The client & server on different LANs is connected to a Wide Area Network (WAN) via a router.
- Routers are the building blocks of WANs.
- The largest WAN today is the Internet.
- Many companies build their own WANs and these private WANs may or may not be connected to the Internet.

Q.List and explain the steps involved in simple daytime client.

1. **Include headers** : headers includes numerous system headers that are needed by most network programs and defines various constants that we use .
2. **Command-line arguments:** definition of the main function along with the command-line arguments.
3. **Create TCP socket :** The socket function creates an Internet (AF_INET) stream (SOCK_STREAM) socket, which is a fancy name for a TCP socket. The function returns a small integer descriptor that we can use to identify the socket in all future function calls
4. **Specify server's IP address and port :** Fill in an Internet socket address structure (sockaddr_in) with the server's IP address and port number. We set the entire structure to 0 using bzero, set the address family to AF_INET, set the port number to 13, and set the IP address to the value specified as the first command-line argument (argv[1]).
5. **Establish connection with server:** The connect function, when applied to a TCP socket, establishes a TCP connection with the server specified by the socket address structure.
6. **Read and display server's reply:** We read the server's reply and display the result using the standard I/O functions.
7. **Terminate program:** exit terminates the program. Unix always closes all open descriptors when a process terminates, so our TCP socket is now closed.

Q.Protocol Independence

- It is better to make a program *protocol-independent*.
- Protocol-independent is achieved by using the `getaddrinfo` function (which is called by `tcp_connect`).
- Another deficiency in our programs is that the user must enter the server's IP address as a dotted-decimal number. Humans work better with names instead of numbers.

Modify the day time client program for IPv6.

1. **`sockaddr_in6`**
2. **`AF_INET6`**
3. **`sin6_family`**
4. **`sin6_port`**
5. **`sin6_addr`**

Q.What are wrapper functions? Develop the wrapper function for the following:

a. **Socket function**
b. **Pthread_mutex_lock**

Since terminating on an error is the common case, we can shorten our programs by defining a *wrapper function* that performs the actual function call, tests the return value, and terminates on an error. The convention we use is to capitalize the name of the function

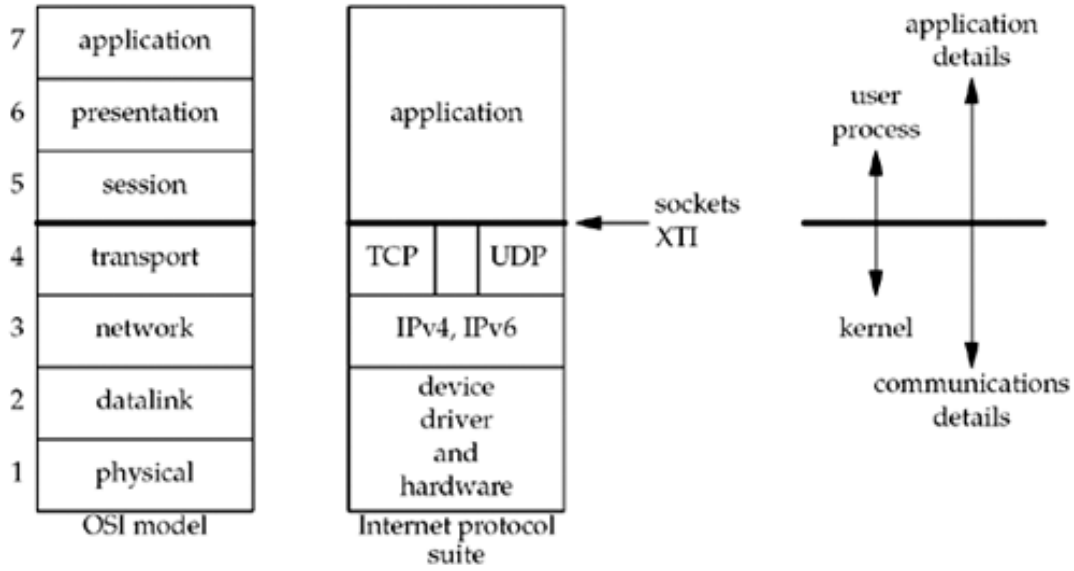
```
int Socket(int family, int type, int protocol)
{
        int n;
        if ( (n = socket(family, type, protocol)) < 0)
                err_sys("socket error");
        return (n);
}

Void Pthread_mutex_lock(pthread_mutex_t *mptr)
{
```

```
        int n;
        if ( (n = pthread_mutex_lock(mptr)) == 0)
        return;
        errno = n;
        err_sys("pthread_mutex_lock error");
}
```

Q.Write a note on Unix errno value.

- When an error occurs in a Unix function (such as one of the socket functions), the global variable errno is set to a positive value indicating the type of error
- err_sys function looks at the value of errno and prints the corresponding error message string (e.g., "Connection timed out" if errno equals ETIMEDOUT)
- The value of errno is set by a function only if an error occurs.
- Its value is undefined if the function does not return an error.
- All of the positive error values are constants with all-uppercase names beginning with "E," and are normally defined in the <sys/errno.h> header

Q.Explain with a neat block diagram the layers of OSI model and Internet protocol suite.



A common way to describe the layers in a network is to use the International Organization for Standardization (ISO) *open systems interconnection* (OSI) model for computer communications. This is a seven-layer model,

Datalink+ physical = drivers and network hardware.

The network layer is handled by the IPv4 and IPv6 protocols

We show a gap between TCP and UDP to indicate that it is possible for an application to bypass the transport layer and use IPv4 or IPv6 directly. This is called a *raw socket.*

The upper three layers of the OSI model are combined into a single layer called the application. This is the Web client (browser), Telnet client, Web server, FTP server, or whatever application we are using.

Q.Write a note on 64- bit architectures.

Datatype	ILP32 model	LP64 model
char	8	8
short	16	16
int	32	32
long	32	64
pointer	32	64

Q.List and explain the features of TCP Protocol in detail.

1. TCP is reliable protocol. That is, the receiver always sends either positive or negative acknowledgement about the data packet to the sender, so that the sender always has bright clue about whether the data packet is reached the destination or it needs to resend it.

2. TCP ensures that the data reaches intended destination in the same order it was sent.

3. TCP is connection oriented. TCP requires that connection between two remote points be established before sending actual data.

4. TCP provides error-checking and recovery mechanism.

5. TCP provides end-to-end communication.

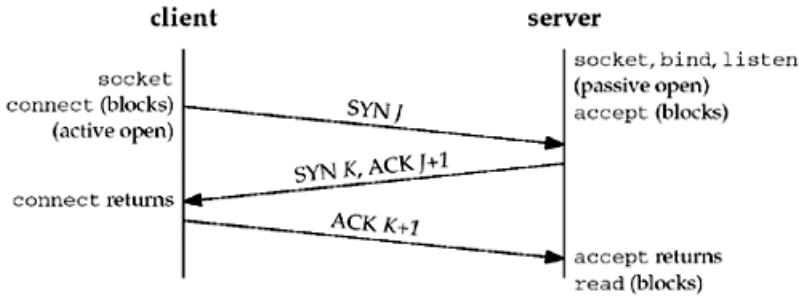
6. TCP operates in Client/Server point-to-point mode.

7. TCP provides full duplex server

	TCP	UDP	SCTP
Reliability	trustworthy	Unreliable	Trustworthy
Connection type	Connection-oriented	Connectionless	Connection-oriented
Transmission type	Byte-oriented	News-oriented	News-oriented
Transfer sequence	Strictly ordered	Disordered	Partially ordered
Overload control	Yes	No	Yes
Error tolerance	No	No	Yes

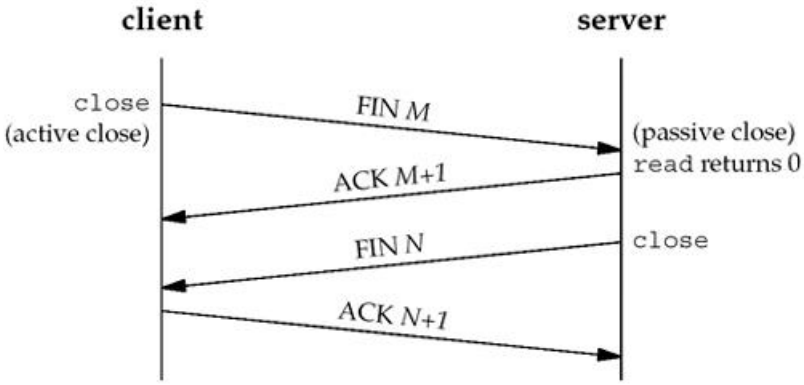
Q.Explain with a neat diagrams the following:

c. **TCP connection establishment**
d. **TCP data transfer**
e. **TCP connection termination**



- The server must be prepared to accept an incoming connection. This is normally done by calling socket, bind, and listen and is called a *passive open*.
- The client issues an *active open* by calling connect. This causes the client TCP to send a "synchronize" (SYN) segment, which tells the server the client's initial sequence number for the data that the client will send on the connection.
- The server must acknowledge (ACK) the client's SYN and the server must also send its own SYN containing the initial sequence number for the data that the server will send on the connection. The server sends its SYN and the ACK of the client's SYN in a single segment.
- The client must acknowledge the server's SYN.

The minimum number of packets required for this exchange is three; hence, this is called TCP's *three-way handshake*.

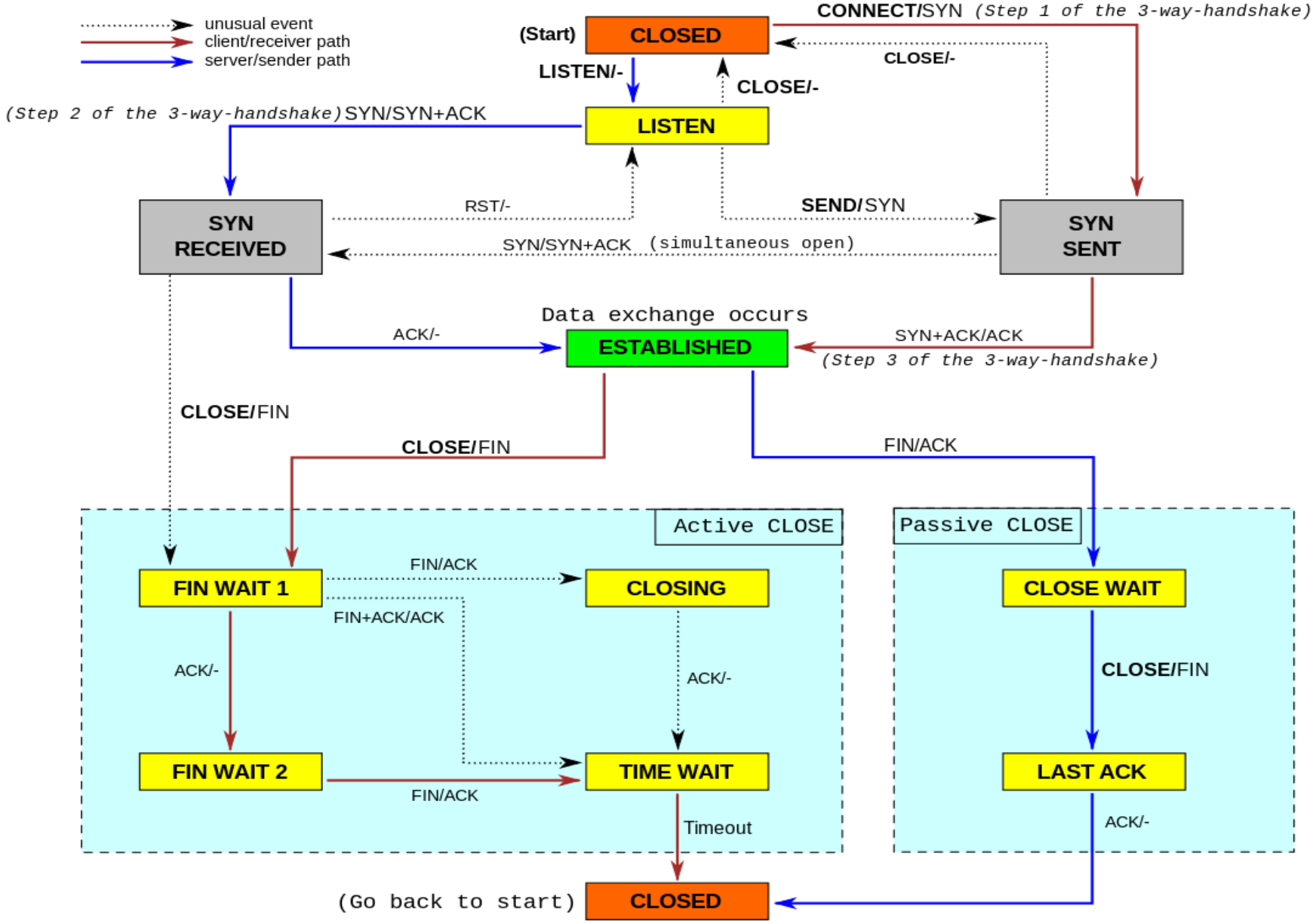


1. One application calls close first, and we say that this end performs the *active close*. This end's TCP sends a FIN segment, which means it is finished sending data.

2. The other end that receives the FIN performs the *passive close*. The received FIN is acknowledged by TCP. The receipt of the FIN is also passed to the application as an end-of-file , since the receipt of the FIN means the application will not receive any additional data on the connection.

3. Sometime later, the application that received the end-of-file will close its socket. This causes its TCP to send a FIN.

4. The TCP on the system that receives this final FIN (the end that did the active close) acknowledges the FIN.


unusual event
client/receiver path
server/sender path
(Start)
CLOSED
CONNECT/SYN (Step 1 of the 3-way-handshake)
CLOSE/-
LISTEN/-
CLOSE/-
(Step 2 of the 3-way-handshake) SYN/SYN+ACK
LISTEN
SYN RECEIVED
RST/-
SEND/SYN
SYN/SYN+ACK (simultaneous open)
SYN SENT
Data exchange occurs
ACK/-
ESTABLISHED
SYN+ACK/ACK
(Step 3 of the 3-way-handshake)
CLOSE/FIN
CLOSE/FIN
FIN/ACK
Active CLOSE
Passive CLOSE
FIN WAIT 1
FIN/ACK
CLOSING
FIN+ACK/ACK
CLOSE WAIT
ACK/-
ACK/-
CLOSE/FIN
FIN WAIT 2
FIN/ACK
TIME WAIT
LAST ACK
Timeout
ACK/-
(Go back to start)
CLOSED