Lab No 1: **Date:**

INTRODUCTION TO COMPUTER NETWORKS

Objectives:

In this lab, student will be able to

1. Connect the computers in Local Area Network
2. Study of network devices

I. Basic Computer Network terminologies

Local Area Networks (LANs)

A network is any collection of independent computers that exchange information with each other over a shared communication medium. Local Area Networks or LANs are usually confined to a limited geographic area, such as a single building or a college campus. LANs can be small, linking as few as three computers, but can often link hundreds of computers used by thousands of people.

Ethernet

Ethernet is the standard way to connect computers on a network over a wired connection. It provides a simple interface and for connecting multiple devices, such computers, routers, and switches.

What is an IP address?

"IP address" is a shorter way of saying "Internet Protocol address." IP addresses are the numbers assigned to computer network interfaces. Although we use names to refer to the things we seek on the Internet, such as www.example.org, computers translate these names into numerical addresses so they can send data to the right location. So when you send an email, visit a web site, or participate in a video conference, your computer sends data packets to the IP address of the other end of the connection and receives packets destined for its own IP address.

When computers are connected to Internet they are uniquely identified by an address which is IP Address.

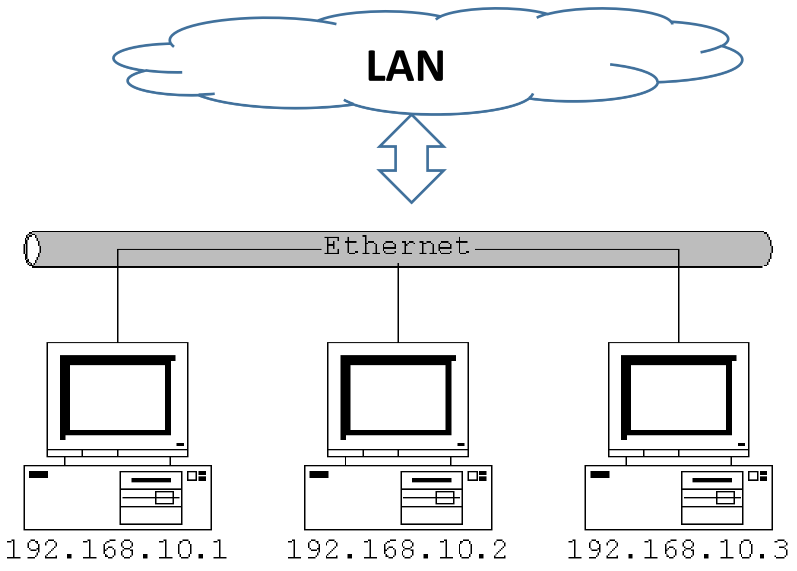


Figure 1.1: Host computers connected to internet identified through ip address

Figure 1.1 shows host computers with IP address 192.168.10.1, 192.168.10.2 and 192.168.10.3 connected to LAN using Ethernet cable.

<u>Network Devices</u>

Repeater: A **repeater** is a **network** device that is used to regenerate or replicate signals that are weakened or distorted by transmission over long distances and through areas with high levels of electromagnetic interference (EMI).

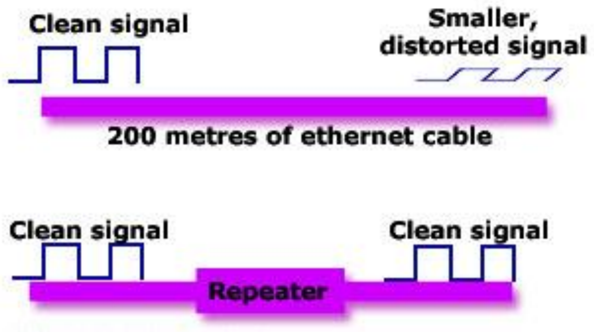


Figure 1.2: Repeater

Hub: A hub, in the context of networking, is a hardware device that relays communication data When a packet arrives at one port, it is copied to the other ports so that all segments of the LAN can see all packets.

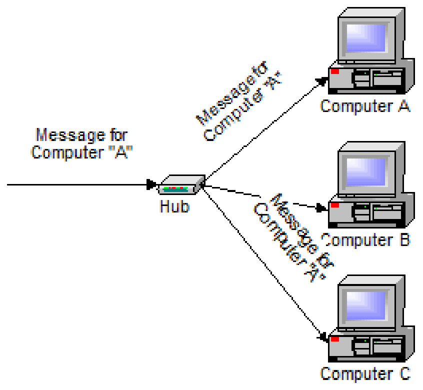


Figure 1.3: Hub

Switch:

A switch is a high-speed device that receives incoming data packets and redirects them to their destination on a local area network (LAN).

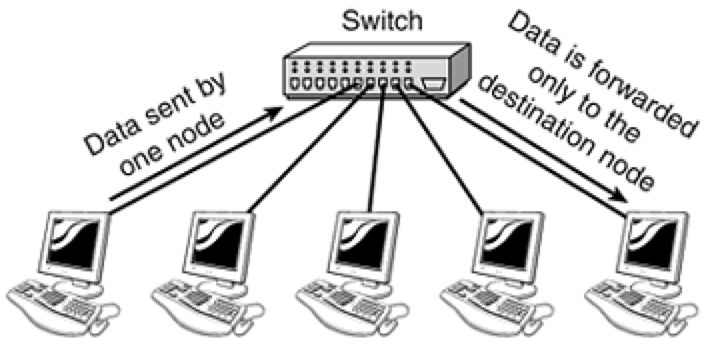


Figure 1.4: Switch

Bridge:

A **network bridge** is a computer **networking** device that creates a single aggregate **network** from multiple communication **networks** or **network** segments.

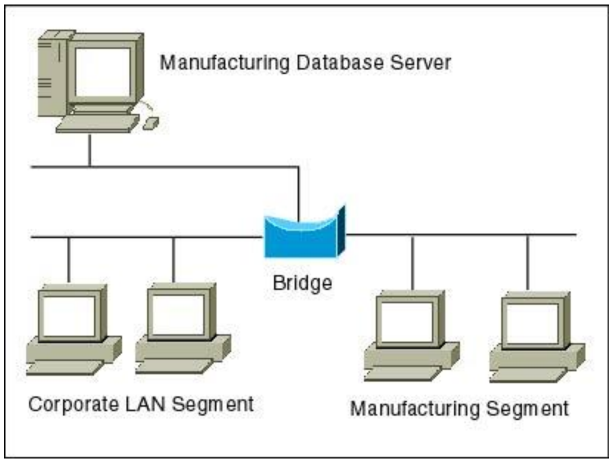


Figure 1.5: Bridge

Router:

A **router** is hardware device designed to receive, analyze and move incoming packets to another network. It may also be used to convert the packets to another network interface, drop them, and perform other actions relating to a network.

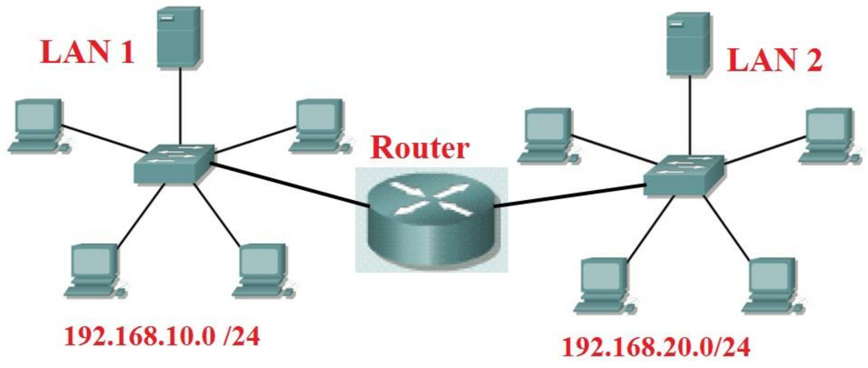


Figure 1.6: Router

Gateway:

A gateway is a network node that connects two networks using different protocols together. While a bridge is used to join two similar types of networks, a gateway is used to join two dissimilar networks. The most common gateway is a router that connects a home or enterprise network to the internet.

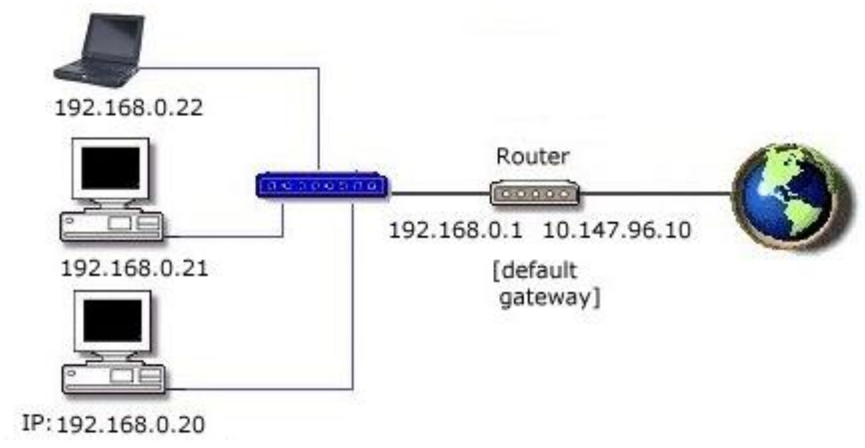


Figure 1.7: Gateway

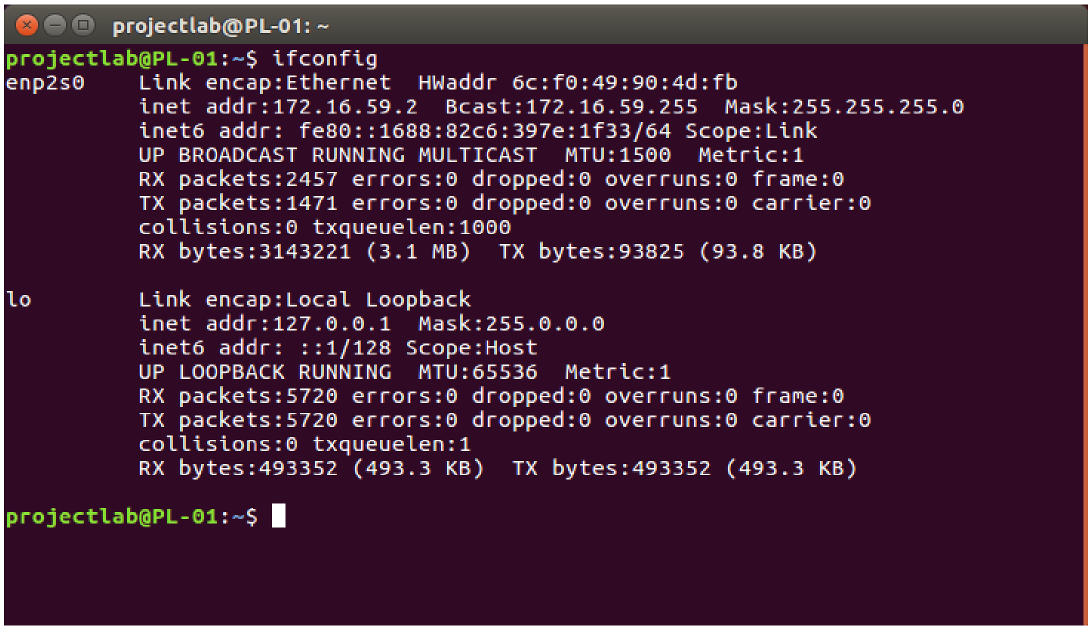
II. Unix Utilities to test Internet connection and to diagnose congestion between computers:

Ifconfig: The **ifconfig** command is used to configure a network interface. The following options are used for the reconfiguration of the IP address and network mask.
ifconfig *-a :* Shows the states of all interfaces in the system.

ifconfig *<interface name> down* : Disables the network interface, where interface name is the name of the Ethernet interface.

ifconfig *<interface name> <new IP address> up* : Assigns a new IP address to the interface and brings it up.

ifconfig *<interface name > netmask <new netmask>* : Assigns a new network mask for the interface.



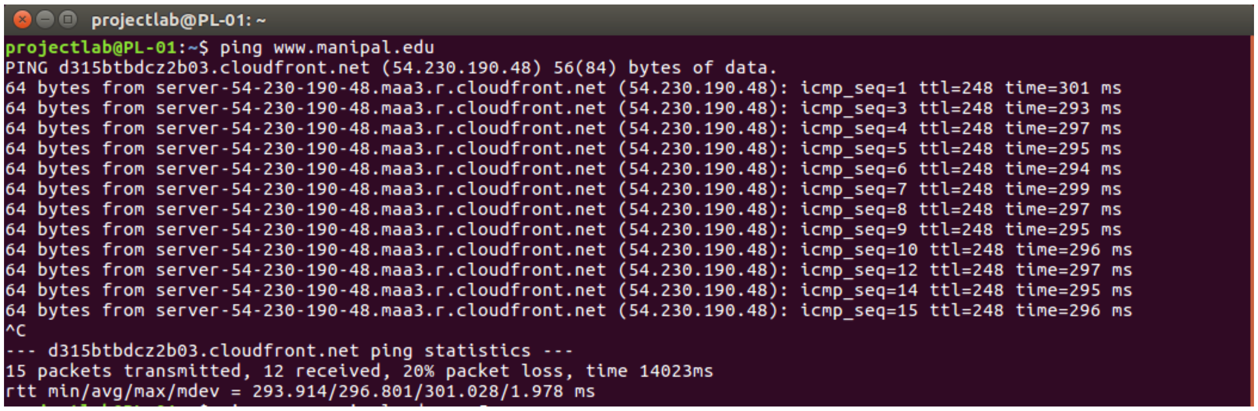
```
projectlab@PL-01:~$ ifconfig
enp2s0    Link encap:Ethernet  HWaddr 6c:f0:49:90:4d:fb
          inet addr:172.16.59.2  Bcast:172.16.59.255  Mask:255.255.255.0
          inet6 addr: fe80::1688:82c6:397e:1f33/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2457 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1471 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3143221 (3.1 MB)  TX bytes:93825 (93.8 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:5720 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5720 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:493352 (493.3 KB)  TX bytes:493352 (493.3 KB)

projectlab@PL-01:~$ 
```

Figure 1.8: ifconfig command

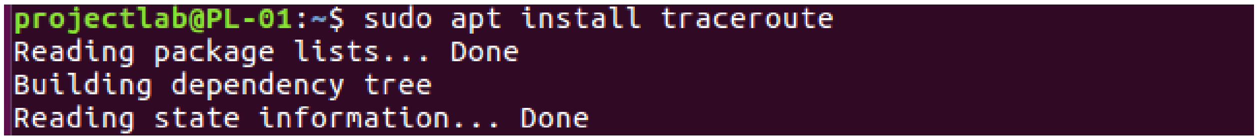
Ping: Ping (also written as PING or ping) is a utility that you use to determine whether or not a specific IP address is accessible. Ping works by sending a packet to a specified address and waiting for a reply. Ping is used primarily to troubleshoot Internet connections and there are many freeware and shareware Ping utilities available for download.



```
projectlab@PL-01:~$ ping www.manipal.edu
PING d315btbdcz2b03.cloudfront.net (54.230.190.48) 56(84) bytes of data.
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=1 ttl=248 time=301 ms
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=3 ttl=248 time=293 ms
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=4 ttl=248 time=297 ms
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=5 ttl=248 time=295 ms
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=6 ttl=248 time=294 ms
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=7 ttl=248 time=299 ms
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=8 ttl=248 time=297 ms
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=9 ttl=248 time=295 ms
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=10 ttl=248 time=296 ms
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=12 ttl=248 time=297 ms
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=14 ttl=248 time=295 ms
64 bytes from server-54-230-190-48.maa3.r.cloudfront.net (54.230.190.48): icmp_seq=15 ttl=248 time=296 ms
^C
--- d315btbdcz2b03.cloudfront.net ping statistics ---
15 packets transmitted, 12 received, 20% packet loss, time 14023ms
rtt min/avg/max/mdev = 293.914/296.801/301.028/1.978 ms
```

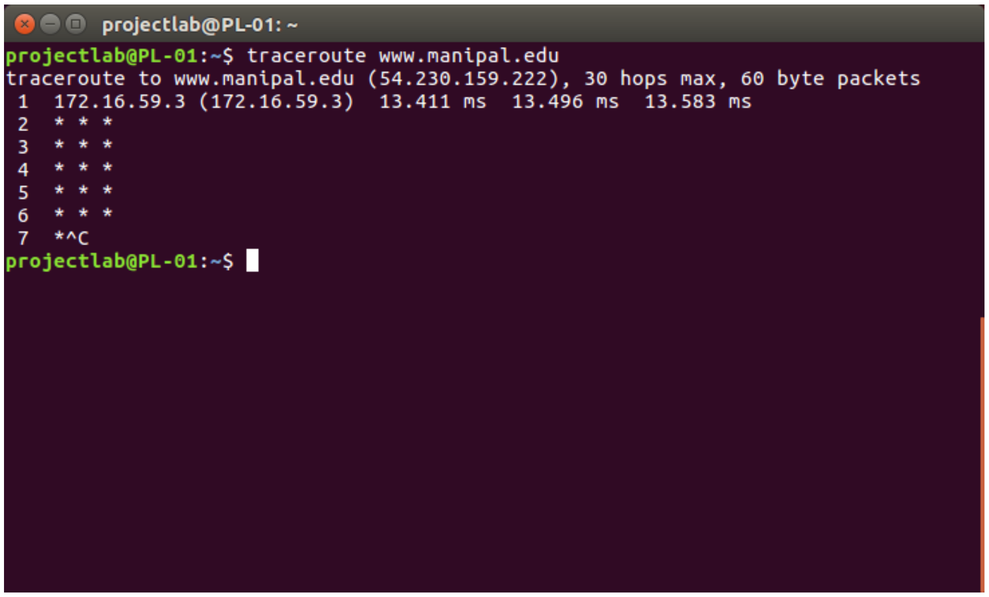
Figure 1.9 : ping command

Traceroute:Traceroute is a utility that traces a packet from your computer to an Internet host, but it will show you how many hops the packet requires to reach the host and how long each hop takes. If you're visiting a Web site and pages are appearing slowly, you can use traceroute to figure out where the longest delays are occurring. Traceroute utilities work by sending packets with low time-to-live (TTL) fields. The TTL value specifies how many hops the packet is allowed before it is returned. When a packet can't reach its destination because the TTL value is too low, the last host returns the packet and identifies itself. By sending a series of packets and incrementing the TTL value with each successive packet, traceroute finds out who all the intermediary hosts are.



```
projectlab@PL-01:~$ sudo apt install traceroute
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Figure 1.10: Install traceroute command



```
projectlab@PL-01:~$ traceroute www.manipal.edu
traceroute to www.manipal.edu (54.230.159.222), 30 hops max, 60 byte packets
 1  172.16.59.3 (172.16.59.3)  13.411 ms  13.496 ms  13.583 ms
 2  * * *
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  *^C
projectlab@PL-01:~$ 
```

Figure 1.11: traceroute command

III. SOLVED EXERCISE

Connect the computers in local area network

1. Right click on the network manager applet,

    - Goto **Edit connections →wired tab→add**

2. Put the mac address of the interface you will be configuring. The ifconfig command can show you what the mac address is:

$ ifconfig

```
eth0      Link encap:Ethernet  HWaddr 00:30:1b:b9:53:94
HWaddr 00:30:1b:b9:53:94 = mac address
```

3. Then click the ipv4 settings tab. set method to manual.
4. click add to add IP address

```
# example for computer one would be
address  | netmask       | gateway
10.0.0.1 | 255.255.255.0 |
# example for computer two would be
10.0.0.2 | 255.255.255.0 |
```

5. See if you can ping each other from computer one.

$ ping 10.0.0.2

```
ping 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=128 time=0.457 ms
```

IV. LAB EXERCISES

1. What is the IP of the machine you are using? Compare it with the IP of your neighbors. Are the IPs of your neighbors same? Why or Why not?
2. Use the ping command for the following URLs and record the success or failure statistics along with the average round trip time.
   a) google.com
   b) facebook.com
3. Based on output of **ifconfig /all** command, identify the following
   - Host name[computer name]
   - MAC address of your system[physical address]
   - Subnet mask
   - default gateway
4. In the LAN, compare your result of Q3, with your neighbor computers. What similarities do you see in the MAC address?

5. Ping the computer’s loopback IP address. Type the following command:

ping 127.0.0.1

The address 127.0.0.1 is reserved for loopback testing. If the ping is successful, then TCP/IP is properly installed and functioning on this computer

V. ADDITIONAL EXERCISES

1. In the terminal ,type **netstat** and press enter. The computer displays information about the open connections on the computer or returns to the prompt if there are none. Analyze the output.
2. With the help **man** pages, analyze the results of **ping** with options.

LAB NO: 2 **Date:**

Socket Programming in 'C' using TCP/IP

Objectives:

- To familiarize with application level programming with sockets.
- To establish communication between the peers.
- To send data through sockets using system calls.

Prerequisites:

- Knowledge of the C programing language.
- Knowledge of network devices.

I. Introduction

Computer networks has hosts, routers and communication channels. Host computer run applications, these applications may send and receive data from other host computers through network. Routers in the network forward information which is in the form of packets. Packets contain some information such as IP address of sender, receiver and information to be sent. Protocol is a standard used to define a method of exchanging data over a computer network such as local area network, Internet, Intranet, etc. Each protocol has its own method of how data is formatted when sent and what to do with it once received, how that data is compressed or how to check for errors in data. One of the most common and known protocols is HTTP (HyperText Transfer Protocol), which is a protocol used to transmit data over the world wide web (Internet).TCP/IP provides end-to-end connectivity specifying how data should be formatted, addressed, transmitted, routed, and received at the destination. Each host will be running multiple applications, PORT NUMBER is used to identify the application running in the host.

Client Server Architecture

Client/server architecture is a computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client. This type of architecture has one or more client computers connected to a central server over a network or Internet connection. This system shares computing resources. Here, a machine (referred as client) makes a request to connect to another machine (called as server) for providing some service. The services running on the server run on known ports (application identifiers) and the client needs to know the address of the server machine and this port in order to connect to the server. On the other hand, the server does not need to know about the address or the port of the client at the time of connection initiation. The first packet which the client sends as a request to the server contains these information about the client which are further used by the server to send any information. Client (Active Open) acts as the active device which makes the first move to establish the connection whereas the server (Passive Open) passively waits for such requests from some client. Client/server architecture works when the client computer sends a resource or process request to the server over the network connection, which is then processed and delivered to the client. A server computer can manage several clients simultaneously, whereas one client can be connected to several servers at a time, each providing a different set of services. In its simplest form, the Internet is also based on client/server architecture where the Web server serves many simultaneous users with Web page and or website data.These client and server communication happens through sockets.

Sockets

Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors. In Unix, every I/O action is done by writing or reading a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else. To a programmer, a socket looks and behaves much like a low-level file descriptor. This is because commands such as read() and write() work with sockets in the same way they do with files and pipes.

Types of Sockets

There are four types of sockets available to the users. The first two are most commonly used and the last one is rarely used.

- Stream Sockets: Delivery in a networked environment is guaranteed. If you send through the stream socket three items "A, B, C", they will arrive in the same order - "A, B, C". These sockets use TCP (Transmission Control Protocol) for data transmission. If delivery is impossible, the sender receives an error indicator. Data records do not have any boundaries.
- Datagram Sockets: Delivery in a networked environment is not guaranteed. They're connectionless because you don't need to have an open connection as in Stream Sockets - you build a packet with the destination information and send it out. They use UDP (User Datagram Protocol).
- Raw Sockets: These provide users access to the underlying communication protocols, which support socket abstractions. Raw sockets are not intended for the general user; they have been provided mainly for those interested in developing new communication protocols, or for gaining access to some of the more cryptic facilities of an existing protocol.

Types of Servers

There are two types of servers you can have:

- Iterative Server: This is the simplest form of server where a server process serves one client and after completing the first request, it takes request from another client. Meanwhile, another client keeps waiting.
- Concurrent Servers: This type of server runs multiple concurrent processes to serve many requests at a time because one process may take longer and another client cannot wait for so long. The simplest way to write a concurrent server under Unix is to fork a child process to handle each client separately.

How to Make a Client

The system calls for establishing a connection are somewhat different for the client and the server, but both involve the basic construct of a socket. Both the processes establish their own sockets. The steps involved in establishing a socket on the client side are as follows:

- Create a socket with the socket() system call.
- Connect the socket to the address of the server using the connect() system call.

- Send and receive data. There are a number of ways to do this, but the simplest way is to use the read() and write() system calls.

How to Make a Server

The steps involved in establishing a socket on the server side are as follows:

- Create a socket with the socket() system call.
- Bind the socket to an address using the bind() system call. For a server socket on the Internet, an address consists of a port number on the host machine.
- Listen for connections with the listen() system call.
- Accept a connection with the accept() system call. This call typically blocks the connection until a client connects with the server.
- Send and receive data using the read() and write() system calls.

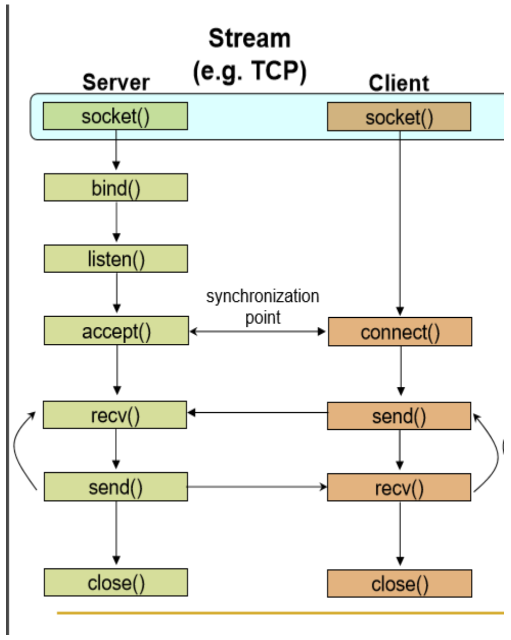


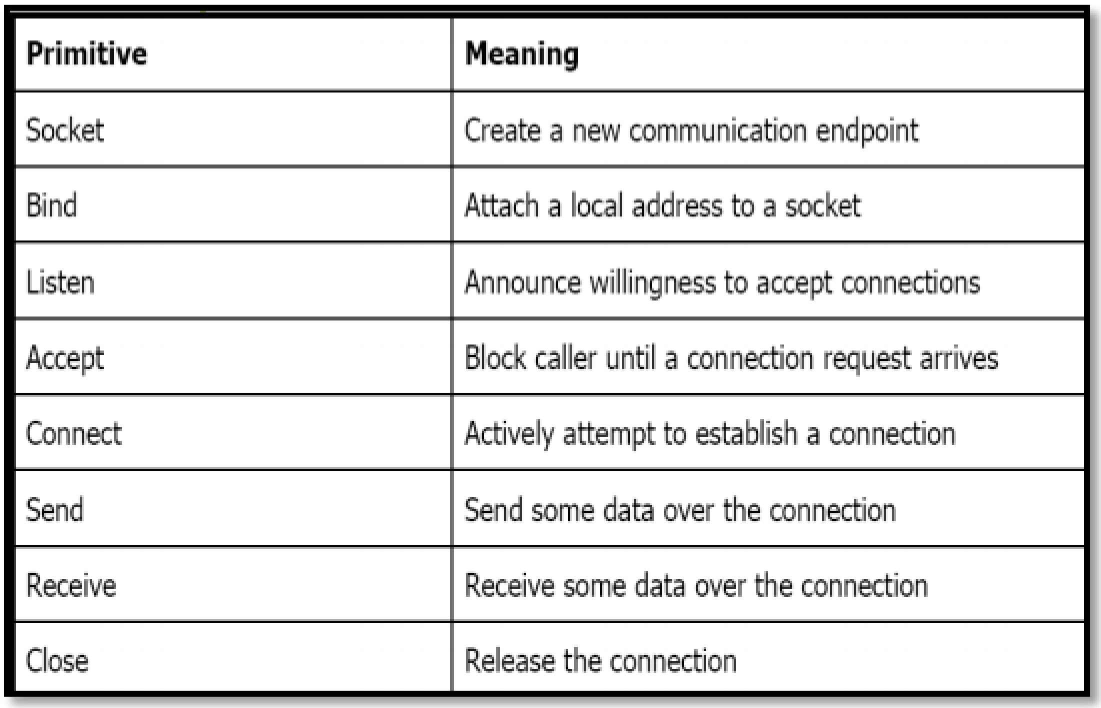
Figure 2.1 TCP client server interactions

Basic data structures used in Socket programming

Various structures are used in Unix Socket Programming to hold information about the address and port, and other information. Most socket functions require a pointer to a socket address structure as an argument. Structures defined in this chapter are related to Internet Protocol Family.

- **Socket Descriptor**
  - A simple file descriptor in Unix. Data type is integer.
- **Socket Address**
  - This construct holds the information for socket address.

Table 2.1 System calls used in socket programming



Primitive	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

syntax

```
struct sockaddrs {
unsigned short sa_family; // address family, AF_xxx or //PF_xxx
char sa_data[14];               // 14 bytes of protocol address
 };
```

- AF stands for Address Family and PF stands for Protocol Family.

Table 2.2 Address Family

Name	Purpose

AF_UNIX, AF_LOCAL	Local communication
AF_INET	IPv4 Internet protocols
AF_INET6	IPv6 Internet protocols
AF_IPX	IPX - Novell protocols

- **struct sockaddr_in**
  - This construct holds the information about the address family, port number, Internet address,and the size of the struct sockaddr.
- struct sockaddr_in {

  short int sin_family; // Address family unsigned short int sin_port; // Port number

  struct in_addr sin_addr; // Internet address

  };
- The IP address structure, in_addr, is defined as follows

  struct in_addr {

  unsigned long int s_addr;

  };

Some of The System Calls Used For Conversion

Some systems (like x8086) are Little Endian i-e. least signficant byte is stored in the higher address, whereas in Big endian systems most significant byte is stored in the higher address. Consider a situation where a Little Endian system wants to communicate with a Big Endian one, if there is no standard for data representation then the data sent by one machine is misinterpreted by the other. So standard has been defined for the data representation in the network (called Network Byte Order) which is the Big Endian.

The system calls that help us to convert a short/long from Host Byte order to Network Byte

Order and vice versa are:

- htons() -- "Host to Network Short"

- htonl() -- "Host to Network Long"
- ntohs() -- "Network to Host Short"
- ntohl() -- "Network to Host Long"

To ensure correct byte ordering of the 16-bit port number, your server and client need to apply these functions to the port address. For example

```
server_address.sin_addr.s_addr= htonl(INADDR_ANY);
server_address.sin_port = htons(9734);
```

IP address is a 32bit integer-not convenient for humans. So, the address is written in dotted decimal representation.

- **inet_addr()** converts the Internet host address from the standard numbers-and-dots notation into binary data. It returns nonzero if the address is valid, zero if not.
- **inet_aton()** is also used for same purpose.

System calls used

1. Socket creation in C using socket():

Syntax:

int sockid = socket(family, type, protocol);

where

- *sockid* is socket descriptor, an integer (like a file-handle)
- *family* is the communication domain, like PF_INET for IPv4 protocols and Internet addresses or PF_UNIX for Local communication and File addresses.
- *Type* defines communication type such as SOCK_STREAM or SOCK_DGRAM.
- *protocol* specifies protocol used. It take values like IPPROTO_TCP or IPPROTO_UDP but usually set to 0 (i.e., use default protocol).

If the return value *sockid* is negative values, it means there is problem in socket creation.

NOTE: socket call does not specify where data will be coming from, nor where it will be going to – it just creates the interface!

2. Assign address to socket using bind():

Bind() associates and reserves a port for use by the socket.

Syntax:

int status = bind(sockid, &addrport, size);

- *Sockid* is a integer describing socket descriptor
- *addrport* is struct sockaddr which contains the (IP) address and port of the machine „ for TCP/IP server, internet address is usually set to INADDR_ANY, i.e., chooses any incoming interface
- *size* specifies the size (in bytes) of the addrport structure
- *Status* will be assigned -1 returns on failure.

3. Listening to connection requests using listen():

This system call instructs TCP protocol implementation to listen for connections

Syntax:

int status = listen(sockid, queueLimit);

- *Sockid* is socket descriptor which is created using socket()
- Queuelemit is an integer which specifies number of active participants that can "wait" for a connection
- *Status* will be assigned -1 returns on failure.

Note : The listening socket (sockid) is never used for sending and receiving is used by the server only as a way to get new sockets.

4. Establish Connection using connect()

The client establishes a connection with the server by calling connect().

Syntax:

int status = connect(sockid, &foreignAddr, addrlen);

- *sockid* is socket descriptor to be used in connection
- *foreignAddr is* struct sockaddr which contains address of the passive participant
- *addrlen is* sizeof(*foreignAddr*)
- *Status* will be assigned -1 returns on failure

Note : connect() is blocking where as listen() is non blocking.

5. Accept incoming Connection using accept()

The server gets a socket for an incoming client connection by calling accept()

Syntax:

int newsockid = accept(sockid, &clientAddr, &addrLen);

- *newsockid* is an_integer, the new socket is created in server which is client specific and this new socket is used for data-transfer between server and client.
- Sockid is the socket created using socket system call, which is used only to listen to incoming requests from client.
- clientAddr is in the form of struct sockaddr, address of the active participant.
- addrLen is size of clientAddrult parameter

Note: accept() is blocking, it waits for connection before returning and dequeues the next connection on the queue for socket (sockid).

6. Exchanging data with stream socket

Application running in server and client(s) can transfer data using send() and receive() system call.

Syntax:

int count = send(sockid, msg, msgLen, flags);

- *Sockid* is the new socket descriptor created by accept in server side and socket in client side, depending on where it is used.
- *Msg* is an array holding message to be transmitted
- msgLen holds length of message (in bytes) to transmit
- flags are integer, special options, usually set 0
- Return value *count* has number of bytes transmitted and is set to -1 on error

  Syntax:

  int count = recv(sockid, recvBuf, bufLen, flags);
- recvBuf stores received message
- bufLen holds number if bytes

7. closing the socket using close()

When finished using a socket, the socket should be closed.

Syntax:

int status= close(sockid);

- sockid: the file descriptor (socket being closed)
- status: 0 if successful, -1 if error

Closing a socket closes a connection (for stream socket) and frees up the port used by the socket.

II. SOLVED EXERCISE:

Write an iterative TCP client server program where client sends a message to server and server echoes back the message to client. Client should display the original message and echoed message.

Algorithm: TCPEcho

SERVER:

```
STEP 1: Start
STEP 2: Declare the variables for the socket
STEP 3: Specify the family, protocol, IP address and port number
STEP 4: Create a socket using socket() function
STEP 5: Bind the IP address and Port number
STEP 6: Listen and accept the client's request for the connection
STEP 7: Read the client's message
STEP 8: Display the client's message
STEP 9: Send the client's message
STEP 10: Close the socket
```

CLIENT:

```
STEP 1: Start
STEP 2: Declare the variables for the socket
STEP 3:  Specify the family, protocol, IP address and port number
```

STEP 4: Create a socket using socket() function

STEP 5: Call the connect() function

STEP 6: Send the input message to the server

STEP 7: Receive the message from server

STEP 8: Display the message

STEP 9: Close the socket

STEP 10: Stop

Note: As socket is also a file descriptor, we can use read and write system calls to receive and send data.

Program:

Server code:

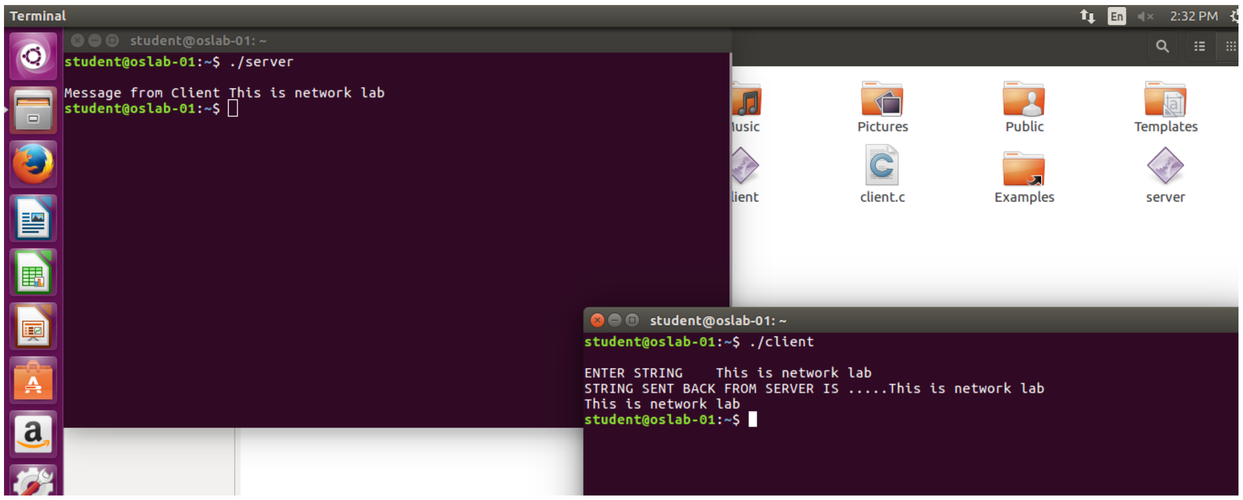
```
#include<stdio.h>
#include<string.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#define PORTNO 10200
int main()
{
        int sockfd,newsockfd,portno,clilen,n=1;
        char buf[256];
        struct sockaddr_in seraddr,cliaddr;
        int i,value;
        sockfd = socket(AF_INET,SOCK_STREAM,0);
        seraddr.sin_family = AF_INET;
        seraddr.sin_addr.s_addr = INADDR_ANY;
        seraddr.sin_port = htons(PORTNO);
        bind(sockfd,(struct sockaddr *)&seraddr,sizeof(seraddr));
        listen(sockfd,1);
        clilen = sizeof(clilen);
        newsockfd=accept(sockfd,(struct sockaddr *)&cliaddr,&clilen);
        n = read(newsockfd,buf,sizeof(buf));
        printf(" \nMessage from Client %s \n",buf);
        n = write(newsockfd,buf,sizeof(buf));
       return 0;
}
```

Client Code:

```
#include<sys/types.h>
#include<sys/socket.h>
#include<stdio.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<stdlib.h>
#include<string.h>
int main()
{
      int len,result,sockfd,n=1;
      struct sockaddr_in address;
      char ch[256],buf[256];
      sockfd = socket(AF_INET, SOCK_STREAM, 0);
      address.sin_family=AF_INET;
      address.sin_addr.s_addr=inet_addr("127.0.0.1");
      address.sin_port=htons(10200);
      len = sizeof(address);
      result=connect(sockfd,(struct sockaddr *)&address,len);
      if(result==-1)
```

```
        {
                perror("\nCLIENT ERROR");
                exit(1);
        }
        printf("\nENTER STRING\t");
        gets(ch);
       ch[strlen(ch)]='\0';
       write(sockfd,ch,strlen(ch));
       printf("STRING SENT BACK FROM SERVER IS .....");
      while(n){
      n=read(sockfd,buf,sizeof(buf));
      puts(buf);
}
  return 0;
}
```

Sample Input and Output



```
student@oslab-01:~$ ./server

Message from Client This is network lab
student@oslab-01:~$
```

```
student@oslab-01:~$ ./client

ENTER STRING    This is network lab
STRING SENT BACK FROM SERVER IS .....This is network lab
This is network lab
student@oslab-01:~$
```

Steps to execute the program

1. Open two terminal windows and open a text file from each terminal with .c extension using command: $gedit *filename.c*

2. Type the client and server program in separate text files and save it before exiting the text window.

3. First compile and run the server using commands mentioned below

   a. $gcc *filename –o executablefileName* //renaming the a.out file

   b. $./ executablefileName

4. Compile and run the client using the same instructions as listed in 3a & 3b.

Note: The ephemeral port number has to be changed every time the program is executed.

III. LAB EXERCISES:

Write an iterative TCP client server 'C' program

1. DayTime Server: Where client sends request to time server to send current time. Server responds to the request and sends the current time of server to client. [Hint: read man pages of asctime() and localtime()]
2. ChatServer: Where client and server should be able to transfer messages until one of them types "QUIT".
3. StringProcessingServer: To perform string operations in the server and display the result at the client. The client accepts a sentence from the user and sends it to the server. The client then accepts a word to be replaced in the sentence and sends it to the server. The server will replace the word and sends the modified sentence back to the client which is displayed on the client screen. Then both the processes terminate.

IV. ADDITIONAL EXERCISES:

Write an iterative TCP client server 'C' program

1. To illustrate encryption and decryption of messages using TCP. The client accepts message to be encrypted through standard input device. The client will encrypt the string by adding 4(random value) to ASCII value of each alphabet. The encrypted message is sent to the server. The server then decrypts the message and displays both encrypted and decrypted form of the string. Program terminates after one session.

2. Where the client accepts a sentence from the user and sends it to the server. The server will check for duplicate words in the string. Server will find number of occurrence of duplicate words present and remove the duplicate words by retaining single occurrence of the word and send the resultant sentence to the client. The client displays the received data on the client screen. The process repeats until user enter the string “Stop”. Then both the processes terminate.