ASSIGNMENT - 1

Problem Statement: Prepare straight through cable and cross-over cable using CAT 5 cable and RJ45 connectors and write a report on it.

Theory:

1) **Ethernet:** It is a widely used LAN (Local Area Network). The original '10 BASE 5' Ethernet uses coaxial cable as a shared radius. While the newer ethernet uses ~~coaxial~~ twisted pair cables and fibre-optic cables in conjunction with switches/routers.

Different types of Ethernet:

- **Traditional Ethernet** (10 Mbps)
  10 BASE 5, 10 BASE 2, 10 BASE T, 10 BASE F
- **Fast Ethernet** (100 Mbps)
  100 BASE-TY, 100 BASE-FX, 100 BASE-T.
- **Gigabit Ethernet** (1 Gbps)
  1000 BASE-F, 1000 BASE-SX, 1000 BASE-T, 1000 BASE-LX.
- **10-Gigabit Ethernet** (10 Gbps)
  10 GBASE-LX4, ~~10 GBASE-SX, 1000 BASE-T, 1000 BASE-LX~~.
  10 GBASE-FR, 10 GBASE-SR

2) **Twisted Pair Cable:** A twisted pair cable is a cable made by interjoining two separate insulated wires and running them parallel to each other. This type of cable is widely used in different kinds of data and voice infrastructures. It is often used to help avoid certain kinds of signal interference.
Two different types of twisted pair cables are:-
   i) Unshielded Twisted Pair cables (UTP)
   ii) Shielded Twisted Pair Cables (STP)

- UTP: They have no. metallic shielding. This makes the cable small in diameter but unprotected against electrical interference.
- STP: This cable is used in various kinds of networks to prevent crosstalk. It contains an extra foil wrapping to help shield the cable signals. It is costlier when compared to UTP.

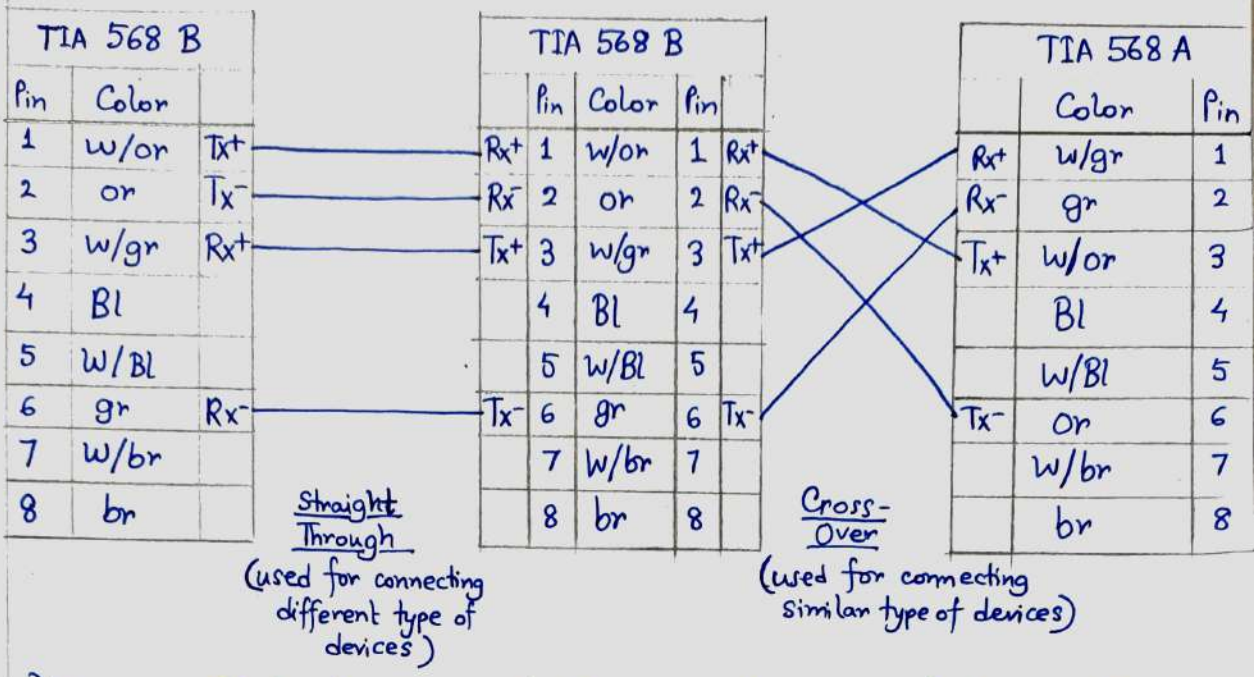
3) Category 5 Cable: Commonly referred to as CAT5, is a twisted pair cable for computer networks. It is a cable widely used for 100 Base-T and 1000 Base-T networks.

There are 4 twisted pair i.e. 8 wires with colors: Blue, Green, Orange, Brown, one pair for transmission, one for receiving and remaining two pairs are unused.

EIA/TIA 568A and EIA/TIA 568B termination standards define the pins to pair assignment in CAT 5/5e cables.

4) Straight through Cable: A straight through cable is a type of twisted pair cable that is used in LAN to connect a computer to a network hub such as a router. This type of cable is also sometimes called a patch cable, or a straight through cable. Straight through cables use one wiring standard. Either both ends use T568A or both ends ~~uses~~ uses T568B standard. It is used to connect two devices of different type.

5) ~~Unlik~~ Cross Over Cable: Unlike straight through cable, cross over cables uses two different wiring standards. The one end uses T568A and the other one uses T568B wiring standard. The internal wiring of Ethernet crossover cables reverses the transmit and receive signals. It is used to connect two devices of same type.



6) Crimping Tool: A crimping tool is a device used to join two pieces of metal by deforming one of both of them in a way that causes them to hold each other. The result of the tool's work is called crimp.

Procedure:

Steps to crimp RJ45 to the CAT5 cable:-

S-1: Strip 1 to 2 inches of the outer skin at the end of the cable wire by making a shallow cut using a crimping tool and then straighten the twisted wires using hand

S-2: Arrange the untwisted wires in a row, placing them into the position running from right to left, in which they will go into RJ45 connector

The order for T568A:-

w/gr, gr, w/or, Bl, w/BL, or, w/br, br.

S-3: Now, we need to trim the untwisted wires to a suitable length by holding the RJ45 connector next to the wires, so that the wires line up evenly with the top of the RJ45 connector.

S-4: Insert the wires into the RJ45 connector, making sure that they stay aligned and each color goes into its appropriate channel.

S-5: Use the crimping tool to crimp the RJ45 connector, (making sure that they stay aligned and each color goes into its appropriate channel) to the cable by pressing the jacket and cable into the connector so that the wedge at the bottom of the connector is pressed into the jacket.

S-6: Now repeat the above instructions to crimp an RJ45 connector to the other end of the cable.

S-7: We can ensure if our cable is working properly once both the ends are crimped properly using a cable tester. (A cable tester is an electronic device used to verify the electrical connections in a signal cable or other wired assembly).

ASSIGNMENT - 2A

Problem Statement: Design and Develop a network topology of a 100 Mbps Fast Ethernet LAN having two 24-Port Switches. Each of the switches is connected with a disjoint set of 10 nodes/PCs configured with class-C IP Address (192.168.4.41-60). Also add PDU to check ping implementing simulation using Cisco Packet Tracer.

Theory: Cisco Packet Tracer is a comprehensive networking technology teaching and learning tool that offers a unique combination of realistic simulation and visualization experience, assessment, activity authoring capabilities and multiuser collaboration and completion opportunities, Innovative features of Packet Tracer will help students and teachers collaborate, solve problems, and learn concepts in an engaging and dynamic social environment.

Benefits of Packet Tracer:-

i) Provides a realistic simulation and visualization learning environment that supplements classroom equipments, including the ability to see interval process in real-time that are normally hidden on real devices.

ii) Enables multiuser, real-time collaborations and completion for dynamic learning.

iii) Enables authorizing and localizing/localization of structured learning activities such as labs, demonstrations, quizes, exams and games.

iv) Empowers students to explore concepts, conduct experiments, and test their understanding of network.

v) Allows students and teachers to design, build, configure and troubleshoot complex networks using virtual equipment.

Procedure:

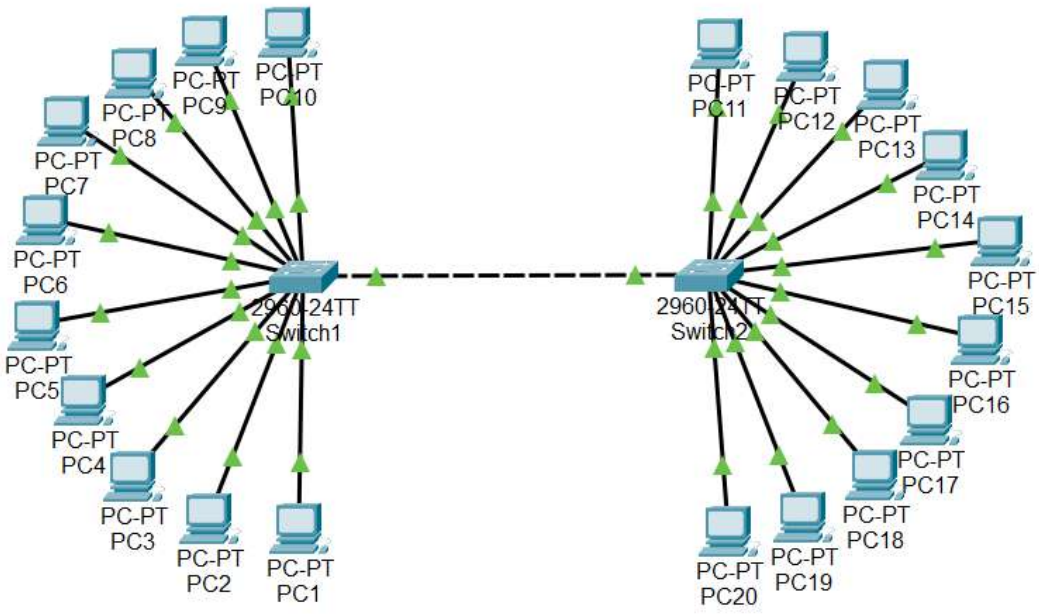
i) After opening CISCO Packet Tracer, we selected two switches which have 24 ports each and connected them with each other with a cross over copper cable.

ii) Then, we have taken 10 PCs for each switch and connected with their respective switch by straight-through copper cable.

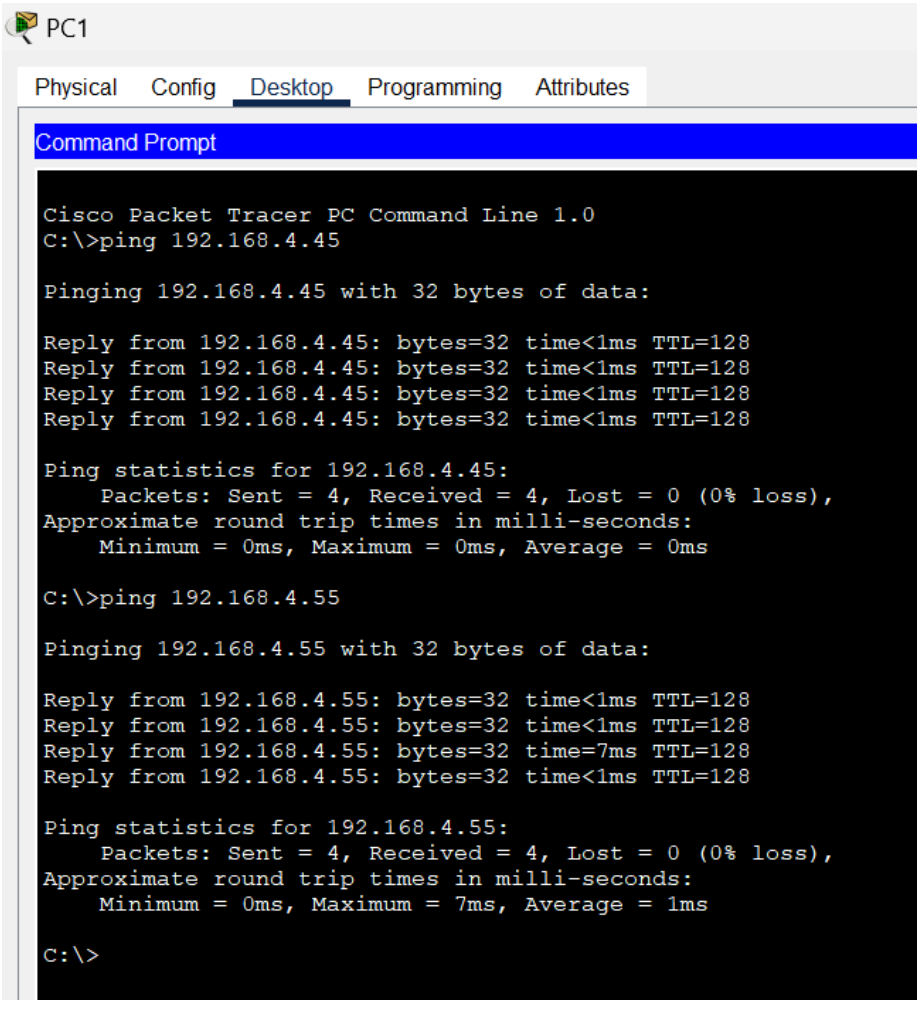
iii) For configuration, we configured each PCs ip-address with the IP address between 192.168.4.41 to 192.168.4.60.

iv) Then we go for simulation. In simulation process, we checked the network with the help of different types of protocols; many of the protocols are accepted by our constructed network that returns the correctness of our network.

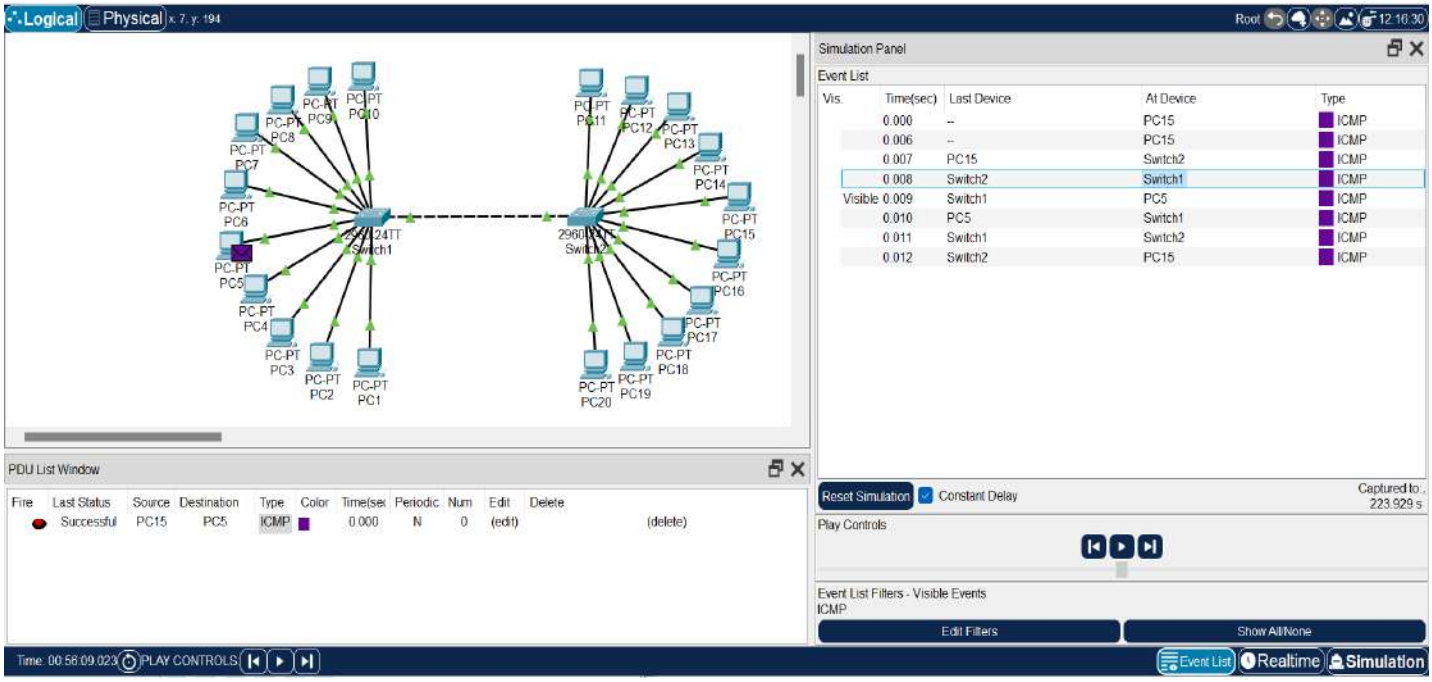
v) We then sent a PDU packet from a PC from one switch to a PC from another switch which help us to check the connection of network.

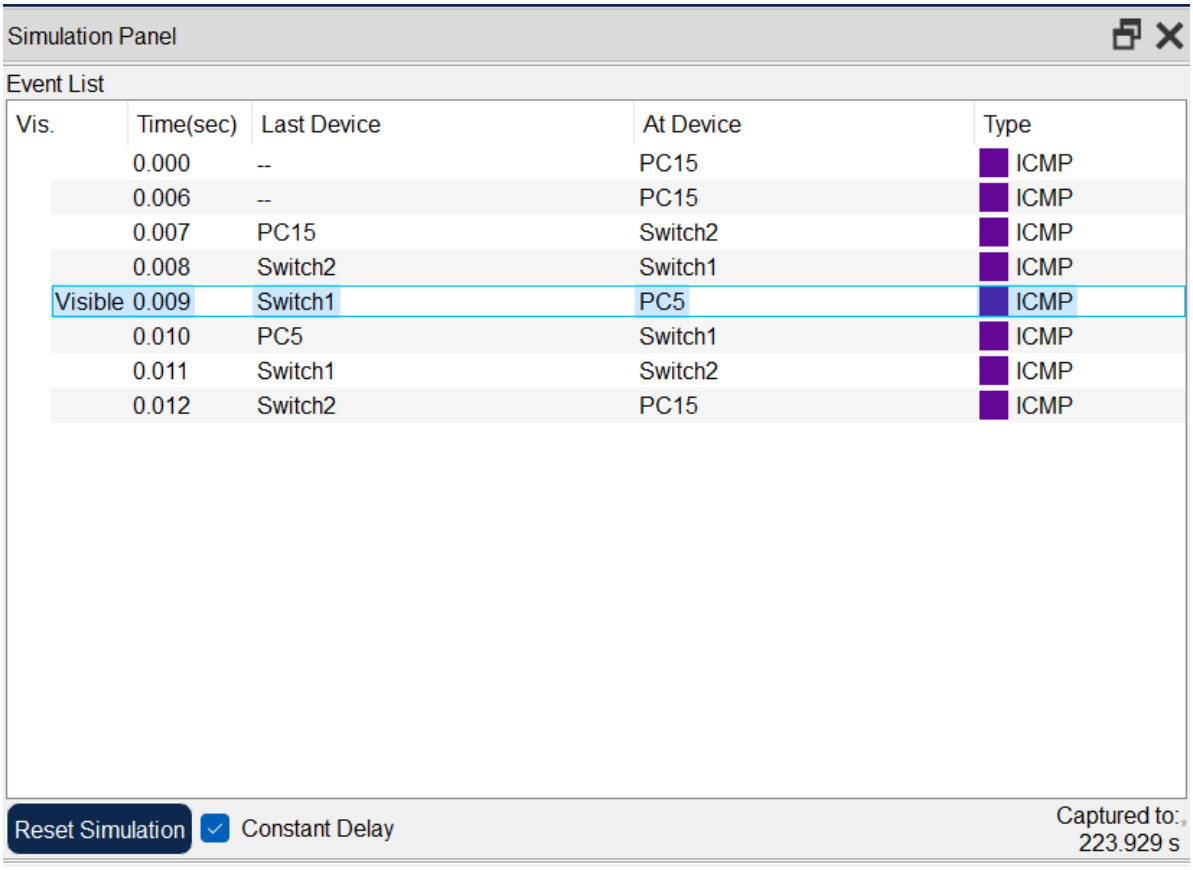


The Required Topology

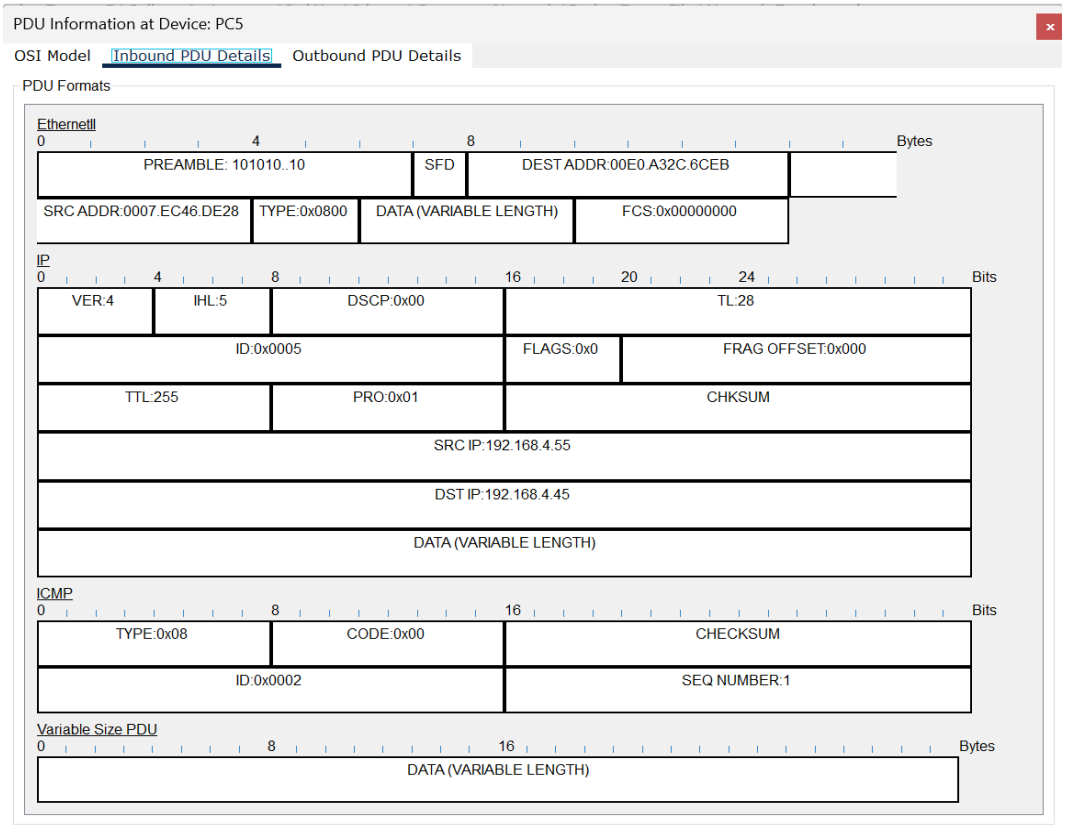


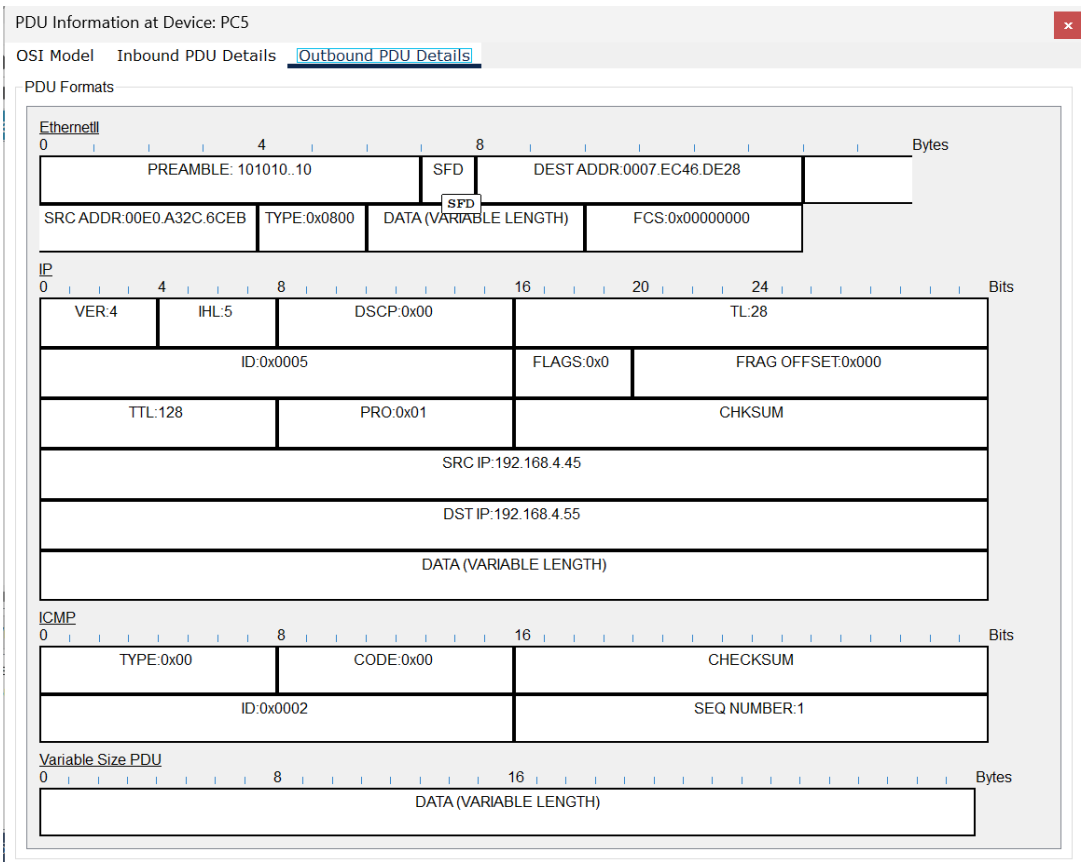
Pinging from one Machine to other Machines of the same Network





Simulation of PDU packet transfer (ICMP) from PC15 to PC5





Inbound and Outbound PDU details at PC5

ASSIGNMENT-2B

Problem Statement: Configure your PC for LAN in Windows and Linux Environment (Configuration for Ethernet Card)

Theory:

1) **IP Address:** An IP address is a unique address that identifies a device on the internet or a local network. IP stands for internet protocol and it is represented in dotted decimal format. IP Address is consists of 32 bits i.e. 4 byte (in case of IPv4)
eg. → 192.222.0.12 is an example of IPv4 address.

2) **MAC Address:** A media access control (MAC) address is a unique identifier which is assigned to a network interface Card (NIC), installed in each machine. It consists of 48 bits i.e. 6 bytes and represented by Hexa-colon format.
e.g.→ 00:0A:83:B1:C0:8C is an example of MAC address

Steps to configure IP Address:

A) **Windows System:**
1) Go to settings
2) Go to Ethernet
3) Go to change Adapter Settings
4) Click on Properties
5) Click on Internet Protocol Session 4 (TCP/IPv4)
6) Go to Properties.
7) Click on "use the following IP address" (i.e. static)
8) Set IP address as 192.168.8.51 [Here 192.168.8 is Network ID and 51 is Host ID] (as requested/required)
9) After putting the value of IP Addresses the subnet mask takes automatically its value as 255.255.255.0.
10) Set the default gate value as 192.168.8.1.

Here, we used class C type IP address.
If we connect ethernet to each single machine then there is no need to set gateway value (the prescribed one above). Otherwise it is mandatory to set.

B) Linux System:

1) Go to Setup
2) Go to network configuration
3) Go to Edit Devices
4) Go to "eth-0"
5) Set Static IP as 192.168.4.222 (as requested/required)
6) Netmask: 255.255.255.0
7) Set Default Gateway IP as 192.168.4.[redacted]1
8) Press 'OK' and 'Save' respectively
9) Go to Edit DNS Configuration.
10) Set Primary DNS as 8.8.8.8 (Google Public DNS)
11) Set Secondary DNS as 4.2.2.2
12) Press OK, save, quit and again quit respectively
13) Now service network is restarted

Steps to share a file in Windows:

1) Right Click on the file and then go to properties
2) Go to sharing
3) Go to share
4) In Network and Sharing Centre turn off the password protected sharing.
5) Press downward arrow in the dropdown list and select everyone. (You may have to change some file permissions for letting others to access it).

Steps to see the Shared File

1) Ping [IP Address]; Example → ping 192.168.8.51
   ping the IP Address of the source machine
2) In Search bar type //[Source IP]
   e.g. → //192.162.8.51

How does a router work?

Routers guide and direct network data, using packets that contain various kinds of data—such as files, communications, and simple transmissions like web interactions.

The data packets have several layers, or sections, one of which carries identifying information such as sender, data type, size, and most importantly, the destination IP (Internet protocol) address. **The router reads this layer, prioritizes the data, and chooses the best route to use for each transmission.**

A router isn't just for data transmission or Internet connections, though. Most routers allow you to connect hard drives and use them as file-sharing servers, or printers that can then be accessed by anyone on the network.

Types of routers

Core router

Core routers are generally used by service providers (i.e. AT&T, Verizon, Vodafone) or cloud providers (i.e. Google, Amazon, Microsoft). They provide maximum bandwidth to connect additional routers or switches. Most small businesses will not need core routers. But very large enterprises that have many employees working in various buildings or locations may use core routers as part of their network architecture.

Edge router

An edge router, also called a gateway router or just "gateway" for short, is a network's outermost point of connection with external networks, including the Internet.

Edge routers are optimized for bandwidth and designed to connect to other routers to distribute data to end users. Edge routers don't usually offer Wi-Fi or the ability to manage local networks fully. They typically have only Ethernet ports—an input to connect to the Internet and several outputs to connect additional routers.

Edge router and modem are somewhat interchangeable terms, though the latter term is no longer commonly used by manufacturers or IT professionals when referencing edge routers.

Distribution router

A distribution router, or interior router, receives data from the edge router (or gateway) via a wired connection and sends it on to end users, typically via Wi-Fi, though the router usually also includes physical (Ethernet) connections for connecting users or additional routers.

Wireless router

Wireless routers, or residential gateways, combine the functions of edge routers and distribution routers. These are commonplace routers for home networks and Internet access.

Most service providers provide full-featured wireless routers as standard equipment. But even if you have the option to use an ISP's wireless router in your small business, you may want to use a business-level router to take advantage of better wireless performance, more connectivity controls, and security.

Virtual router

Virtual routers are pieces of software that allow some router functions to be virtualized in the cloud and delivered as a service. These routers are ideal for large businesses with complex network needs. They offer flexibility, easy scalability, and a lower entry cost. Another benefit of virtual routers is reduced management of local network hardware.

Router Configuration With Cisco Packet Tracer

In this network, a router and 2 PCs are used. Computers are connected with routers using a copper straight-through cable. After forming the network, to check network connectivity a simple PDU is transferred from PC0 to PC1. The network simulation status is successful. **From this network, it can be observed that the router handles data transfers between multiple devices.**

Procedure:

Step-1(Configuring Router1):

1. Select the router and Open CLI.
2. Press ENTER to start configuring Router1.

3. Type **enable** to activate the privileged mode.
4. Type **config t(**configure terminal) to access the configuration menu.
5. Configure interfaces of Router1:

• *Type **interface FastEthernet0/0** to access FastEthernet0/0 and Configure the FastEthernet0/0 interface with the IP address 192.168.10.1 and Subnet mask 255.255.255.0.*

• *Type i**nterface FastEthernet0/1** to access GigabitEthernet0/0 and Configure the FastEthernet0/1 interface with IP address 192.168.20.1 and Subnet mask 255.255.255.0.*

6. Type **no shutdown** to finish.

Router1 Command Line Interface:

Router>enable

Router#config t

Enter configuration commands, one per line. End with CNTL/Z.

Router(config)#interface FastEthernet0/0

Router(config-if)#ip address 192.168.10.1 255.255.255.0

Router(config-if)#no shutdown

Router(config-if)# %LINK-5-CHANGED: Interface GigabitEthernet0/0, changed state to up

Router(config-if)#interface FastEthernet0/1

Router(config-if)#ip address 192.168.20.1 255.255.255.0

Router(config-if)#no shutdown

Step-2(Configuring PCs):

1. Assign IP Addresses to every PC in the network.
2. Select the PC, Go to the desktop and select IP Configuration and assign an IP address, Default gateway, Subnet Mask
3. Assign the default gateway of PC0 as 192.168.10.1.
4. Assign the default gateway of PC1 as 192.168.20.1.

Step-3(Connecting PCs with Router):

1. Connect FastEthernet0 port of PC0 with FastEthernet0/0 port of Router1 using a copper straight-through cable.
2. Connect FastEthernet0 port of PC1 with FastEthernet0/1 port of Router1 using a copper straight-through cable.

Router Configuration Table:

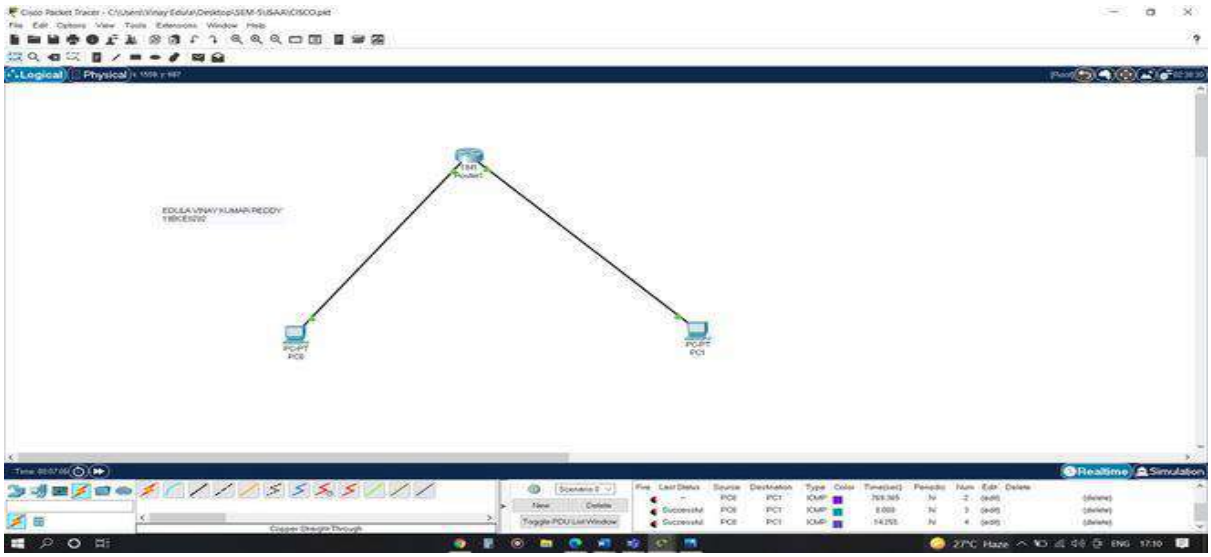
Device Name	IP address FastEthernet0/0	Subnet Mask	IP Address FastEthernet0/1	Subnet Mask
Router1	192.168.10.1	255.255.255.0	192.168.20.1	255.255.255.0

PC Configuration Table:

Device Name	IP address	Subnet Mask	Gateway
PC 0	192.168.10.2	255.255.255.0	192.168.10.1

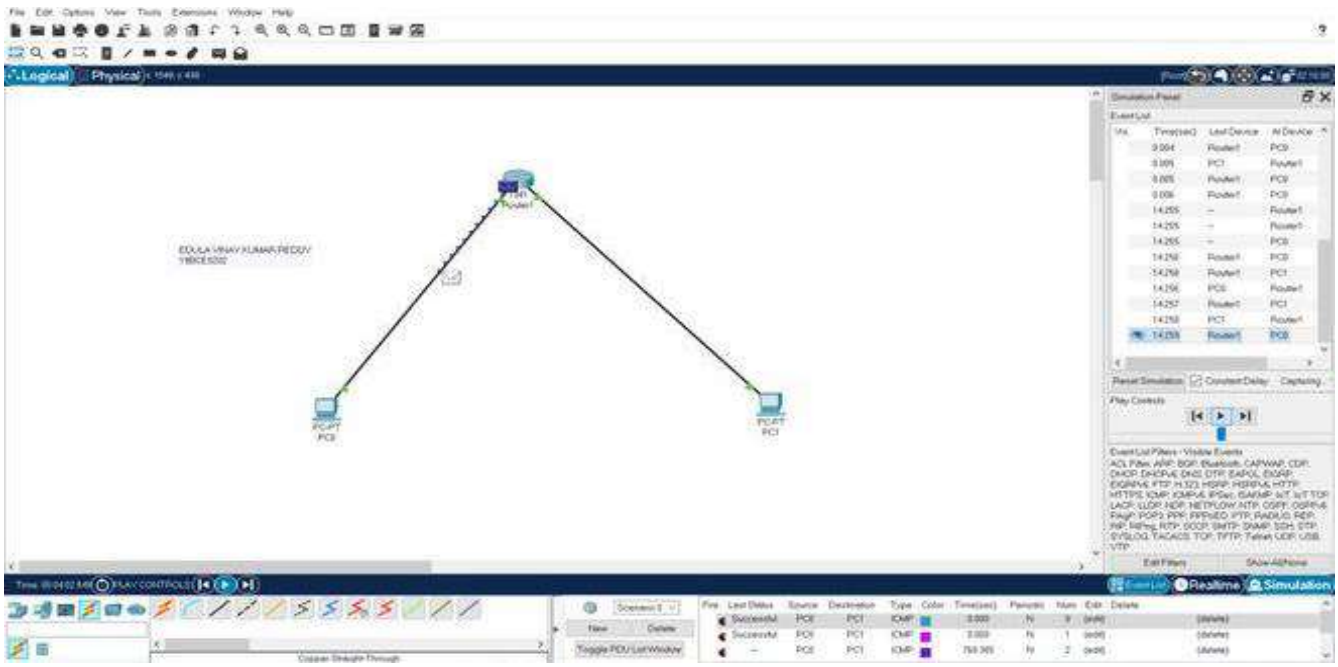
PC 1	192.168.20.2	255.255.255.0	192.168.20.1

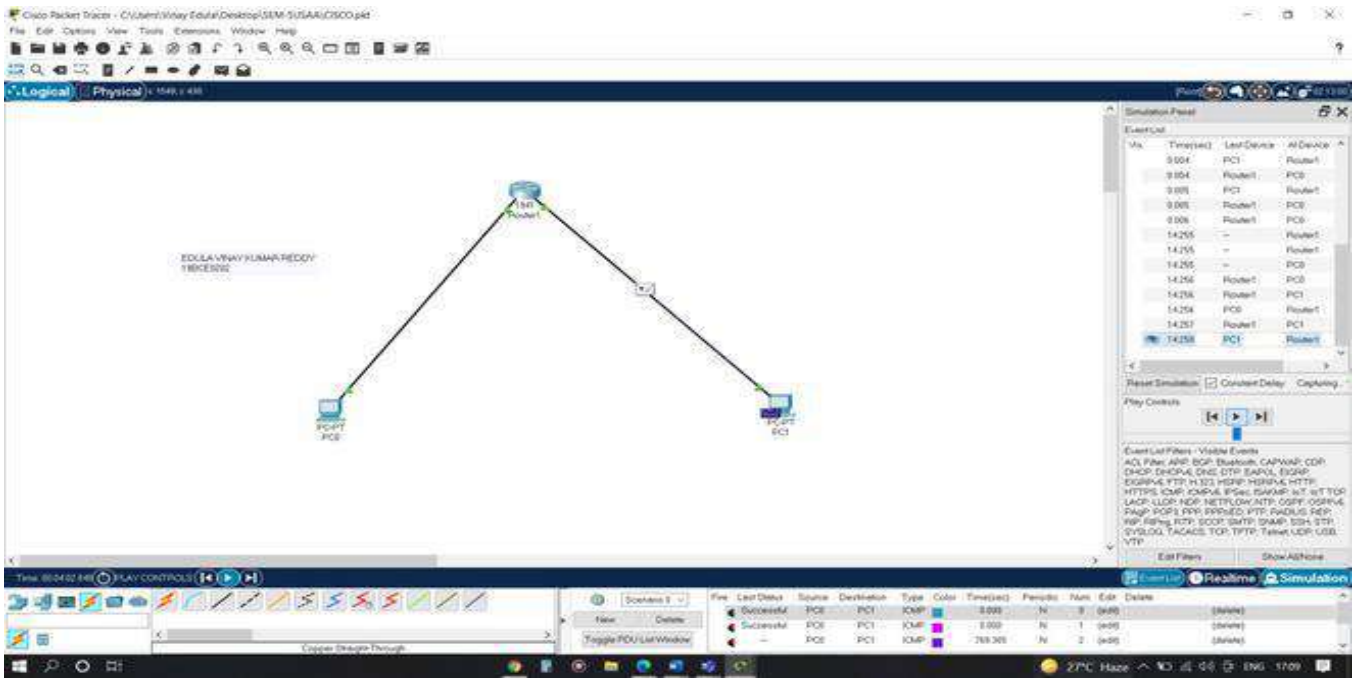
Designed Network topology:



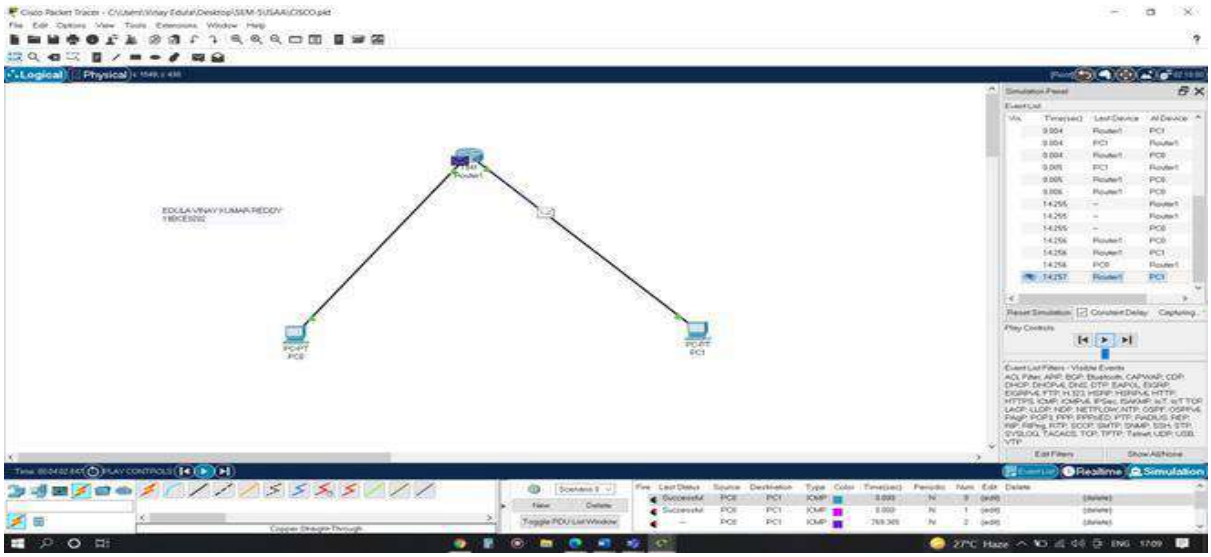
Simulation of Designed Network Topology:

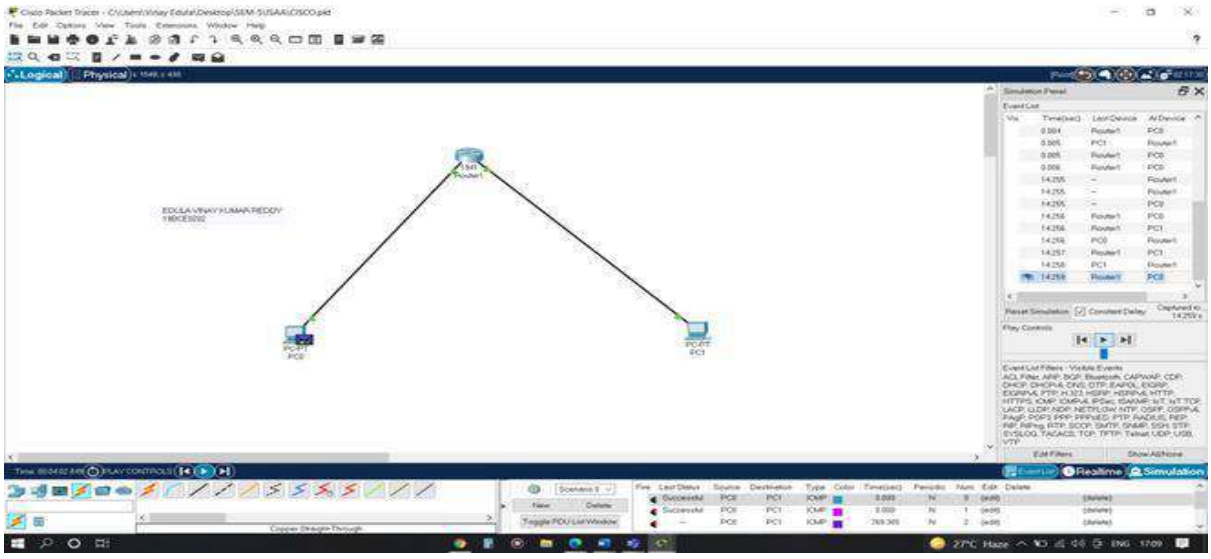
Sending a PDU From PC0 to PC1:





Acknowledgment From PC1 to PC0:





Interface Nomenclature Guide of Router and Switch

This tutorial presents Cisco device naming convention and Interface Nomenclature Guide for Router and Switch. Learn the naming convention used in fixed and modular interfaces including key difference between router and switch interface nomenclature and IOS command to view interface name in detail with example.

Cisco makes a large number of products. Among those I will include 1603, 1841, 2500, 2600 series routers and 2960 series switches in this article to explain the nomenclature. As these products are the base lines for CCNA exam.

Router Interface Nomenclature

In entry level series Cisco makes two types of routers :- fixed and modular. In fixed interface series router, all slots and ports are integrated with device. In modular interface series router, we have a choice to install the ports according to requirement.

Router supports several media types of data link layer including ethernet, fastethernet, gigabitethernet, atm, bri, pri, asynch, serial and many more. In fixed interface series router, interface numbers always start with 0 (zero) and goes their way up within specific interface type.

Fixed interface series router

In fixed interface series router, the interface nomenclature is ***type slot_#/ port_#***.

Nomenclature	Description
type	Type is media type supported by router.
slot_#	This is slot the number.
port_#	The port number is the number of the port in the specified slot.

If a fixed interface router has three Ethernet and two Serial interfaces, then they would be named Ethernet 0, Ethernet 1, Ethernet 2, Serial 0 and Serial 1.

For example, 2503 series router has two fixed serial interfaces; so they would be called serial 0 and serial 1.

Let's take one more example, 1603 Router has one Ethernet port and one BRI port; so it would be called Ethernet 0 and BRI 0.

Modular interface series router

In modular interface series router, the interface nomenclature is ***type slot***_#/***port***_#.

Modular interface series router has modular slots; we can insert any supportive interface in these slots.

For example, following 2600 series router has two slots. First slot would be slot 0; second slot would be slot 1.

If you install four serial interfaces in first slot, then they would be named 0 - 3.

This naming convention has limitation. When we insert WIC without order, configuration of one modular interface might be lost or applied to other modular interface.

For example, in above router we have two modular slots; slot 0, slot 1 and one modular interface with four serial ports. If we install this modular interface in slot 1, then the interface name would be serial 0/0, serial 0/1, serial 0/2, and serial 0/3. If you install another modular interface with two serial ports in slot 0, then new interface name would be serial 0/0, serial 0/1, serial 0/2, serial 0/3, serial 1/0 and serial 1/1. In this case old WIC interface become serial 1/0, and serial 1/1. Therefore, old interface configuration moves to new interfaces.

To overcome this limitation from Cisco 1800, Cisco 2800, and Cisco 3800 series, routers have three tired interface naming convention for WIC slots only.

New naming convention only for WIC slots is ***type slot_#/subslot_#/ port_#***.

Ports installed directly on chassis still use classic convention that is ***type slot_#/ port_#***.

For example in 1841 series router two Ethernet ports are installed on chassis, they have named Fast Ethernet 0/0 and Fast Ethernet 0/1.

Switch Interface Nomenclature

Entry level switches have only fixed interfaces. Higher end switches support modular slots. Modular interface series switch has same nomenclature like router.

Nomenclature of an interface on modular interface series switch is ***type slot_#/port_#***.

In fixed interface series switch all interfaces are installed on chassis. The Catalyst 2960 series switch supports only fixed interfaces.

Nomenclature for interface on fixed interface series switch is ***type slot_#/port_#***.

Nomenclature	Description

type	Type is media type. Media types supported by switch are Ethernet, Fast Ethernet and Gigabit Ethernet.
slot_#	This is slot number.
port_#	The port number is the number of the port in the specified slot.

For example on 2960-8TC first Ethernet interface is called ethernet 0/1 and last Ethernet is called Ethernet 0/8.

Catalyst 2960 series switches

Product	Description
WS-C2960-8TC-S	8 Ethernet 10/100 ports with one dual-purpose uplink
WS-C2960-24-S	24 Ethernet 10/100 ports
WS-C2960-24TC-S	24 Ethernet 10/100 ports and two dual-purpose uplinks
WS-C2960-48TT-S	48 Ethernet 10/100 ports and two copper uplinks
WS-C2960-48TC-S	48 Ethernet 10/100 ports and two dual-purpose uplinks

- For all fixed interfaces on a Cisco switch, the slot number is always 0.
- Port number on switch starts with 1 and goes their way up.

Key difference router and switch nomenclature

On router port number starts with 0(Zero). For example first Fast Ethernet port on router would be fastethernet 0/0.

On switch port number begins with 1. For example first Fast Ethernet port on switch would be fastethernet 0/1.

Cisco IOS commands to find out the name of interface

Interface naming convention may confuse you sometime, especially in exam. Use IOS inbuilt "**show ip interface brief**" command to turn this confusion in surety.

On 2960 switch

```
Switch>enable
Switch#show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol

FastEthernet0/1        unassigned      YES manual down                  down
FastEthernet0/2        unassigned      YES manual down                  down
FastEthernet0/3        unassigned      YES manual down                  down
FastEthernet0/4        unassigned      YES manual down                  down
FastEthernet0/5        unassigned      YES manual down                  down
FastEthernet0/6        unassigned      YES manual down                  down
FastEthernet0/7        unassigned      YES manual down                  down
FastEthernet0/8        unassigned      YES manual down                  down
FastEthernet0/9        unassigned      YES manual down                  down
FastEthernet0/10       unassigned      YES manual down                  down
 --More--
```

On 1800 series router

```
Router>show flash

System flash directory:
File  Length   Name/status
  3   33591768 c1841-advipservicesk9-mz.124-15.T1.bin
  2   28282    sigdef-category.xml
  1   227537   sigdef-default.xml
[33847587 bytes used, 30168797 available, 64016384 total]
63488K bytes of processor board System flash (Read/Write)


Router>
```

On 2600 series router

```
Router>enable
Router#show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol

FastEthernet0/0        unassigned      YES unset  administratively down down

Serial0/0              unassigned      YES unset  administratively down down

Serial0/1              unassigned      YES unset  administratively down down
Router#
```

Basic Router Troubleshooting Commands

In addition to configuring a Cisco router, it is also important to know how to troubleshoot the device. The following are some of the most commonly used Cisco router basic commands for troubleshooting −

- **show running-config** − This command is used to display the current configuration of the router.
- **show interfaces** − This command is used to display information about all of the interfaces on the router, including their status, IP address, and bandwidth usage.
- **show ip route** − This command is used to display the routing table of the router, which shows the paths that packets take to reach their destination.
- **ping IP_ADDRESS** − This command is used to test the connectivity between the router and a specific IP address.
- **traceroute IP_ADDRESS** − This command is used to trace the path that packets take to reach a specific IP address, and to identify any issues along the way.
- **debug command** − This command is used to enable debugging for a specific function, such as routing or authentication.

For example, to display the current configuration of a router, you would enter the following command −

show running-config

To display information about all of the interfaces on a router, you would enter the following command −

show interfaces

To test the connectivity between a router and an IP address of 192.168.1.100, you would enter the following command −

ping 192.168.1.100

To trace the path that packets take to reach an IP address of 8.8.8.8, you would enter the following command −

traceroute 8.8.8.8

To enable debugging for routing on a router, you would enter the following command −

debug ip routing.

ASSIGNMENT-3

Problem Statement: Write a program using pipe such that the content of an existing file to be written to the one end of a pipe and to be read from the other end of it.

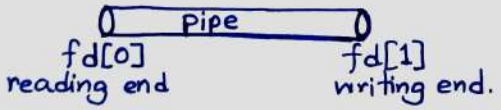
Theory:

1) **IPC ⇒** IPC is a technique to communicate between processes by the OS at rudimentary level. The concept has following applications.
   i) Sending messages as well as signals between two processes.
   ii) Communication between Operating System Commands.

2) **PIPE ⇒** PIPE is an example of Interprocess communication or IPC. pipe() is a system call and it is used for communication between two processes in half-duplex mode. It is also considered as file as both read and write operations can be performed on it. The syntax to create a pipe is as follows.

```
int pipe(int field[2]);
```

pipe takes an array of 2 integers as its only arguments. On successful execution it assigns two file descriptors. Whatever written in field[1] can be read in field[0], vice-versa.



3) **System Call ⇒** The tool used for low-level programming is System Call operating system's all its services using special type of functions, which are inbuilt to the kernel and perform basic functions. These require communication with hardware and software resources available. Users can write program by allowing system calls in UNIX or UNIX like platform using assembly level language or C-like interface.

Source Code: (pipe.c)

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<fentl.h>
#include<unistd.h>
Void main(){
    char s1[1024], s2[1024];
    int fd1, n, fd2[2], pid;
    fd1 = open("a.txt", O_RDONLY, 0764); //a.txt is an existing file.
    n = read(fd1, s1, sizeof(s1));
    pipe(fd2);
    pid = fork();
    if(pid == -1){
        printf("unsuccessful process creation\n");
        exit(1);
    }
    else if(pid==0) // in child process
    {
        close(fd2[1]);
        n = read(fd2[0], s2, sizeof(s2));
        s2[n] = '\0';
        printf("\n reading from pipe\n");
        puts(s2);
    }
    else // in parent process
    {
        close(fd2[0]);
        write(fd2[1], s1, n);
    }
}
```

Output:

```
root: $ gcc pipe.c
root: $ ./a.out
root: $
    reading from pipe
    MCKVIE_CSE_NETWORKING_LAB
root: $
```

Assignment-4

Problem Statement: Write a program using TCP socket such that a client will accept a message from the user and send it to the server. The server will display the message and the IP address and port number of the client.

Theory:

- **TCP:** The Transmission Control Protocol is a transport protocol that is used on the top of IP to ensure reliable transmission of packets. TCP includes mechanisms to solve many of the problems that arise from packet-based messaging.
- **Client:** A client is a program that runs on the local machine requesting service from the server. A client program is a finite program means that the services started by the user and terminates when the service is completed.
- **Server:** A server is a program that runs on the remote machine providing services to the clients. The client requests for a service then server does the necessary processings accordingly in response.
- **Advantages of Client-Server Networks:**

a) Centralized back-up is possible in client-server networks.
b) The networks are more secure as all the shared resources are centrally administered.
c) The use of the individual/dedicated server increases the speed of sharing resources.
d) We can increase the number of clients & server separately.

Code:

Client Side Code:

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#define CLIENT_IP "127.0.0.1"
#define CLIENT_PORT 5551
#define SERVER_IP "192.168.5.36"
#define SERVER_PORT 5550
int main(){
    struct sockaddr_in client, server;
    int sd;
    char str[512];
    bzero((char*)&client, sizeof(client));
    client.sin_family = AF_INET;
    client.sin_port = htons(CLIENT_PORT);
    client.sin_addr.s_addr = inet_addr(CLIENT_IP);
    bzero((char*)&server, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_port = htons(SERVER_PORT);
    server.sin_addr.s_addr = inet_addr(SERVER_IP);
    sd = socket(AF_INET, SOCK_STREAM, 0);
    bind(sd,(struct sockaddr*)&client, sizeof(client));
    connect(sd, (struct sockaddr*)&server, sizeof(server));
    do{
        printf("Enter a message to send to server: ");
        scanf("%s", str);
        send(sd, str, strlen(str)+1, 0);
    }while(strcmp(str,"stop"));
    close(sd);
}
```

Server Side Code:

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#define SERVER_IP "192.168.5.36"
#define SERVER_PORT 5550
int main(){
    struct sockaddr_in client, server;
    int sd, nsd, clen = sizeof(client);
    char str[512];
    bzero((char*)&server, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_port = htons(SERVER_PORT);
    server.sin_addr.s_addr = inet_addr(SERVER_IP);
    sd = socket(AF_INET, SOCK_STREAM, 0);
    bind(sd, (struct sockaddr*)&server, sizeof(server));
    listen(sd, 5);
    while(1){
        nsd = accept(sd, (struct sockaddr*)&client, &clen);
        do{
            memset(str, 0x0, 512);
            recv(nsd, str, 512, 0);
            printf("Received message from client: %s\n", str);
            printf("Client IP: %s\n", inet_ntoa(client.sin_addr));
            printf("Client Port: %u\n", ntohs(client.sin_port));
        }while(strcmp(str, "stop"));
        close(nsd);
    }
}
```

Output:

CLIENT SIDE

Enter a message to send
to the server : Hello

SERVER SIDE

Received Message from Client : Hello
Client IP : 127.0.0.1
Client Port : 5551

ASSIGNMENT-5

Problem Statement : Write a program to develop an Iterative ECHO server using TCP socket.

Theory:

Prinaple of Echo server ⇒ An echo server is an application that allows a client and a server to connect so that the client can send a message to the server and the server can receive the message and send, or echo, it back to the client.

Code:

(a) Client Side

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#define CLIENT_IP "127.0.0.1"
#define CLIENT_PORT 5561
#define SERVER_IP "192.168.5.36"
#define SERVER_PORT 5560
int main( ){
        Struct sockaddr_in client, server;
        int sd;
        char str[512], str1[512];
        bzero ((char*)&client, sizeof(client));
        client.sin_family = AF_INET;
        client.sin_port = htons (CLIENT_PORT);
        client.sin_addr.s_addr = inet_addr (CLIENT_IP);
        bzero ((char)*& server, sizeof(server));
        Server.sin_family = AF_INET;
        server.sin_port = htons (SERVER_PORT);
        server.sin_addr.s_addr = inet_addr (SERVER_IP);
```

```
    sd=socket(AF_INET, SOCK_STREAM, 0);
    bind(sd, (struct sockaddr*)&client, sizeof(client));
   connect(sd, (struct sockaddr*)&server, sizeof(server));
  do{
     printf("\nEnter a message: ");
     scanf("%s", str);
     send(sd, str, strlen(str)+1, 0);
     memset(str1, 0x0, 512);
     recv(sd, str1, 512, 0);
     printf("\nEchoed message : %s ", str1);
  }while(strcmp(str, "stop"));
  close(sd);
```

(b) Server Side

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#define SERVER_IP "192.168.5.36"
#define SERVER_PORT 5560
int main(){
    struct sockaddr_in client, server;
    int sd, nsd, clen = sizeof(client);
    char str[512];
    bzero((char*)&server, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_port = htons(SERVER_PORT);
    server.sin_addr.s_addr = inet_addr(SERVER_IP);
    sd = socket(AF_INET, SOCK_STREAM, 0);
    bind(sd, (struct sockaddr*)&server, sizeof(server));
    listen(sd, 5);
```

```
while(1){
        nsd=accept(sd,(struct sockaddr*)&client,&clen);
       do{
           memset(str,0x0,512);
           recv(nsd,str,512,0);
           send(nsd,str,strlen(str)+1,0);
       }while(strcmp(str,"stop"));
      close(nsd);
   }
}
```

Output:

CLient Side

Server Side

Enter a message : Hello
Echoed message : Hello
Enter a message : Hi
Echoed message : Hi
Enter a message : Stop
Echoed message : Stop

ASSIGNMENT-6

Problem Statement: Write a program to develop a iterative Day-Time Server using TCP Socket.

Theory:

Day-Time Protocol ⇒ It is a service in IP suite defined in 1983 in RFC867. It is intended for testing and measurement purpose in computer networks. A host may connect to server that supports the Day-Time Protocol on either TCP. The server returns an ASCII character string of the current date and time in an unspecified format.

Code:

(a) **Client Side:**

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#define CLIENT_IP "127.0.0.1"
#define CLIENT_PORT 7756
#define SERVER_IP "192.168.5.17"
#define SERVER_PORT 7755
int main(){
        struct sockaddr_in client, server;
        int sd;
        char str[512], str1[512];
        bzero((char*) &client, sizeof(client));
        client.sin_family = AF_INET;
        client.sin_port = htons(CLIENT_PORT);
        client.sin_addr.s_addr = inet_addr(CLIENT_IP);
        bzero((char*)&server, sizeof(server));
        server.sin_family = AF_INET;
        server.sin_port = htons(SERVER_PORT);
```

```
    server.sin_addr.s_addr = inet_addr(SERVER_IP);
    sd = socket(AF_INET, SOCK_STREAM, 0);
    bind(sd, (struct sockaddr*)&client, sizeof(client));
    Connect(sd, (struct sockaddr*)&server, sizeof(server));
    do{
        printf("Enter a message to send to server: ");
        scanf("%s", str1);
        send(sd, str1, strlen(str1)+1, 0);
        memset(str, 0x0, 512);
        recv(sd, str, 512, 0);
        printf("The current time is: %s\n", str);
    }while(strcmp(str1, "stop"));
    close(sd);
}
```

(b) Server Side:

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<time.h>
#define SERVER_IP "192.168.5.17"
#define SERVER_PORT 7755
int main(){
        struct sockaddr_in client, server;
        int sd, nsd, clen=sizeof(client)
        char str[512], str1[512]="Invalid Request";
        char *ptr;
        time_t ti;
        bzero((char*)&server, sizeof(server));
        server.sin_family = AF_INET;
        server.sin_port = htons(SERVER_PORT);
        server.sin_addr.s_addr = inet_addr(SERVER_IP);
```

```
sd=socket(AF_INET, SOCK_STREAM, 0);
bind (sd, (struct sockaddr*)&server, sizeof (server));
listen(sd, 5);
while(1){
    nsd=accept(sd,(struct sockaddr*)&client ,&clen);
    do{
        memset(str, 0x0, 512);
        recv(nsd, str, 512, 0);
        if(strcmp(str, "time")==0){
            ti=time (NULL);
            ptr = ctime (&ti);
            send (nsd, ptr, strlen(ptr)+1, 0);
        }
        else
            send(nsd, str1, strlen(str1)+1, 0);
    }while (strcmp(str, "stop"));
    close(nsd);
}
}
```

Output:

Client Side

Enter a message to send
to server: time

~~The echo data from~~
The current time is:
Wed Apr 24 03:08:24 2023

Server Side

ASSIGNMENT - 7

Problem Statement: Write a Program to develop an iterative echo server using UDP socket.

Theory:

User Datagram Protocol (UDP) ⇒ UDP is a communications protocol that is primarily used for establishing low-latency and loss-tolerating connections between applications on the internet. It speeds-up transmission by enabling the transfer of data before an agreement is provided by the receiving party.

Code:

(a) **Client Side:**

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa~~net~~/inet.h>
#define CLIENT_IP "127.0.0.1"
#define CLIENT_PORT 7745
#define SERVER_IP "192.168.65.17"
#define SERVER_PORT 7746
int main(){
        struct sockaddr_in client, server;
        int sd, slen=sizeof(server);
        char str[512], str1[512];
        bzero((char*)&client, sizeof(client));
        client.sin_family = AF_INET;
        client.sin_port = htons(CLIENT_PORT);
        client.sin_addr.s.addr = inet_addr(CLIENT_IP);
        bzero((char *)&server, sizeof(server));
        server.sin_family = AF_INET;
        server.sin_port = htons(SERVER_PORT);
```

```
    server.sin_addr.s_addr = inet_addr(SERVER_IP);
    sd = socket(AF_INET, SOCK_DGRAM, 0);
    bind(sd, (struct sockaddr*)&client, sizeof(client));
    do{
      printf("Enter a message to send to server: ");
      scanf("%s", str1);
      sendto(sd, str1, strlen(str1)+1, 0, (struct sockaddr*)&server,
                                          sizeof(server));
      memset(str, 0x0, 512);
      recvfrom(sd, str, 512, 0, (struct sockaddr*)&server, &slen);
      printf("The echo data from server: %s\n", str);
    }while(strcmp(str1, "stop"));
    close(sd);
}
```

(b) Server Side

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#define SERVER_IP "192.168.5.17"
#define SERVER_PORT 7746
int main(){
      struct sockaddr_in client, server;
      int sd, clen = sizeof(client);
      char str[512];
      bzero((char*)&server, sizeof(server));
      server.sin_family = AF_INET;
      server.sin_port = htons(SERVER_PORT);
      server.sin_addr.s_addr = inet_addr(SERVER_IP);
      sd = socket(AF_INET, SOCK_DGRAM, 0);
      bind(sd, (struct sockaddr*)&server, sizeof(server));
```

```
    while(1){
        do{
            memset(str,0x0,512);
            recvfrom(sd,str, 512, 0,(struct sockaddr*)&client, &clen);
            sendto (sd, str, strlen(str)+1,0,(struct sockaddr*)&client, clen);
        }while(strcmp(str,"stop"));
    }
}
```

Output:

Client Side

```
Enter a message to send to server: hi
The echo data from server : hi
Enter a message to send to server : stop
The echo data from server : stop
```

Server Side

ASSIGNMENT-8

Problem Statement: Write a Program to develop an iterative Day-Time server using UDP socket.

Theory:

User Datagram Protocol (UDP): UDP is a communications protocol that is primarily used for establishing low-latency and loss-tolerating connections between applications on the internet. It speeds-up transmission by enabling the transfer of data before an agreement is provided by the receiving party.

Code:

(a) **Client Side:**

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define CLIENT_IP "127.0.0.1"
#define CLIENT_PORT 7051
#define SERVER_IP "192.168.5.17"
#define SERVER_PORT 7050
int main(){
        struct sockaddr_in client, server;
        int sd, slen = sizeof(server);
        char str[512], str1[512];
        bzero((char*)&client, sizeof(client));
        client.sin_family = AF_INET;
        client.sin_port = htons(CLIENT_PORT);
        client.sin_addr.s_addr = inet_addr(SERVER_IP);
                                           ^ CLIENT
```

```
    bzero((char*)&server, sizeof(server));
    server.sin_family = AF_INET;
    Server.sin_port = htons(SERVER_PORT);
    server.sin_addr.s_addr = inet_addr(SERVER_IP);
    sd = socket(AF_INET, SOCK_DGRAM, 0);
    bind(sd, (struct sockaddr*)&client, sizeof(client));
    do{
       printf("Enter 'time' to show time: ");
       scanf("%s", str1);
       Sendto(sd, str1, strlen(str1+1)+1, 0, (struct sockaddr*)&server, slen);
       memset(str, 0x0, 512);
       recvfrom(sd, str, 512, 0, (struct sockaddr*)&server, &slen);
       printf("The current time is: %s\n", str);
    }while(strcmp(str1, "stop"));
    close(sd);
}
```

(b) Server Side

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<time.h>
#define SERVER_IP "192.168.65.17"
#define SERVER_PORT "7050"
int main(){
        struct sockaddr_in client, server;
        int sd, clen = sizeof(client);
        char str[512], str1[512] = "Invalid request";
        char* ptr;
        time_t ti;
```

```
bzero((char*)&server, sizeof(server));
server.sin_family = AF_INET;
server.sin_port = htons(SERVER_PORT);
server.sin_addr.s_addr = inet_addr(SERVER_IP);
sd=socket(AF_INET, SOCK_DGRAM, 0);
bind(sd, (struct sockaddr*)&server, sizeof(server));
while(1){
    do{
        memset(str, 0x0, 512);
        recvfrom(sd, str, 512, 0, (struct sockaddr*)&client, &clen);
        if(strcmp(str, "time")==0){
            ti=time(NULL);
            ptr = ctime(&ti);
            sendto(sd, ptr, strlen(ptr)+1, 0, (struct sockaddr*)&client,
                                                clen);
        }
        else{
            sendto(sd, str1, strlen(str1)+1, 0, (struct sockaddr*)
                                &client, clen);
        }
    }while(strcmp(str,"stop"));
}
}
```

Output:

Client Side

Enter 'time' to show time: time
The current time is:
Wed Apr 24 04:02:10 2023

Server Side

ASSIGNMENT-9

Problem Statement: Write a program to develop a concurrent Echo server using TCP Socket. Use atleast two client programs to implement it.

Theory:

Concurrent Server ⇒ A concurrent server is such a server which handles multiple clients at the same time.

Implementation:

Server Side:

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet.h>
#include<arpa/inet.h>
#define SERVER_IP "192.168.5.17"
#define SERVER_PORT 5560
int main(){
    struct sockaddr_in client, server;
    int sd, nsd, clen=sizeof(client), pid;
    char str[512];
    bzero((char*)&server, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_port = htons(SERVER_PORT);
    server.sin_addr.s_addr = inet_addr(SERVER_IP);
    sd = socket(AF_INET, SOCK_STREAM, 0);
    bind(sd, (struct sockaddr*)&server, sizeof(server));
    listen(sd,5);
    while(1){
        nsd=accept(sd, (struct sockaddr*)&client, &clen);
        pid=fork();
        if(pid==0){
            do{
                memset(str, 0*0, 512);
```

```
            recv(nsd, str, 512, 0);
            send(nsd, str, strlen(str)+1, 0);
        }while(strcmp(str,"stop"));
    }
    else if (pid>0)
    {
        close(nsd);
    }
  }
}
```

Client Side:

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define CLIENT_IP "127.0.0.1"   // We have to change this for every client.
#define CLIENT_PORT "5561"      // We have to change this for every client. It must be unique.
#define SERVER_IP "192.168.5.17"
#define SERVER_PORT "5560"
int main(){
    struct sockaddr_in client, server;
    int sd;
    char str[512], str1[512];
    bzero((char*)&client, sizeof(client));
    client.sin_family = AF_INET.
    client.sin_addr.s_addr = inet_addr(CLIENT_IP);
    client.sin_port = htons(CLIENT_IP);
```

```
    bzero((char*)&server, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_port = htons(SERVER_PORT);
    server.sin_addr.s_addr = inet_addr(SERVER_IP);
    sd = socket(AF_INET, SOCK_STREAM, 0);
    bind(sd, (struct sockaddr*)&client, sizeof(client));
    connect(sd, (struct sockaddr*)&server, sizeof(server));
    do{
        printf("\nEnter a message to send to server: ");
        scanf("%s", str);
        send(sd, str, strlen(str)+1, 0);
        memset(str1, 0x0, 512);
        recv(sd, str1, 512, 0);
        printf("\nEchoed message received from server: %s", str1);
    }while(strcmp(str, "stop"));
    close(sd);
}
```

Output :

Client Side (Client 1)

```
gcc client.c -o client
./client
Enter a message to send to server : hello
Echoed message received from server : hello
Enter a message to send to server : debrup
Echoed message received from server : debrup
Enter a message to send to server : client 1
Echoed message received from server : client 1
```

Client Side (Client 2)

```
gcc client2.c -o client2
./client2
Enter a message to send to server : hello
Echoed message received from server : hello
Enter a message to send to server : client 2
Echoed message received from server : client 2
```

Server side

```
gcc server.c -o server
./server
```

ASSIGNMENT - 10

Problem Statement: Write a program to implement sliding window protocol.

Theory:

Sliding Window Protocol ⇒ A sliding window protocol is a feature of packet-based data transmission protocol. Sliding window protocols are used where reliable in-order delivery of packets is required, such as in data-link layer (OSI Model) as well as in the Transmission Control Protocol (TCP).

We implement this protocol using client-server concept. Here from the client side we pass a input stream and the window size, on the basis of window size the input stream will be truncated and sent to server-side as the packet. And server side will acknowledge each of successful delivery of packet.

Implementation

Client-Side:

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#define CLIENT_IP "127.0.0.1"
#define CLIENT_PORT 5580
#define SERVER_IP "192.168.5.17"
#define SERVER_PORT 5581
int main(){
        struct sockaddr_in client,server;
        int sd,w,i,j,count=0;
        char str[512], buff[512],ack[512];
        bzero((char)*&client,sizeof(client));
        client.sin_family= AF_INET;
        client.sin_port = htons(CLIENT_PORT);
        client.sin_addr.s_addr=inet_addr(CLIENT_IP);
        bzero((char)* &server,sizeof(server));
```

```
server.sin_family = AF_INET;
server.sin_port = htons(SERVER_PORT);
server.sin_addr.s_addr = inet_addr(SERVER_IP);
sd = socket(AF_INET, SOCK_STREAM, 0);
bind = (sd, (struct sockaddr*)&client, sizeof(client));
connect(sd, (struct sockaddr*)&server, sizeof(server));
do{
  printf("\nEnter a message: ");
  scanf("%s", &str);
  printf("\nEnter a window size: ");
  scanf("%d", &w);
  j=0;
  count=0;
  for(i=0; i<strlen(str); i++) {
     if (j<w){
       buff[j] = str[i];
       j++;
     }
     if(j==w || i==strlen(str)-1){
       buff[j] = '\0';
       Send(sd, buff, strlen(buff)+1, 0);
       memset(ack, 0x0, 512);
       recv(sd, ack, 512, 0);
       printf("%s %d\n", ack, count++);
       j=0;
     }
  }
}while(strcmp(str, "stop"));
close(sd);
}
```

Server Side:

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#define SERVER_IP "192.168.5.17"
#define SERVER_PORT 5581
int main(){
    struct sockaddr_in client,server;
    int sd,nsd,clen=sizeof(client);
    char str[512],ack[]="ACKNOWLEDGEMENT RECEIVED";
    bzero((char*)&server,sizeof(server));
    server.sin_family=AF_INET;
    server.sin_port=htons(SERVER_PORT);
    server.sin_addr.s_addr=inet_addr(SERVER_IP);
    sd=socket(AF_INET,SOCK_STREAM,0);
    bind(sd,(struct sockaddr*)&server,sizeof(server));
    listen(sd,5);
    while(1){
        nsd=accept(sd,(struct sockaddr*)&client,&clen);
        do{
            memset(str,0x0,512);
            recv(nsd,str,512,0);
            printf("Data Received: ");
            puts(str);
            send(nsd,ack,strlen(ack)+1,0);
        }while(strcmp(str,"stop"));
        close(nsd);
    }
}
```

Output:

Client-Side:

```
gcc client.c -o client
./client
Enter a message: mckvie
Enter window size: 4
ACKNOWLEDGEMENT RECEIVED 0
ACKNOWLEDGEMENT RECEIVED 1

Enter a message: debrup
Enter window size: 2
ACKNOWLEDGEMENT RECEIVED 0
ACKNOWLEDGEMENT RECEIVED 1
ACKNOWLEDGEMENT RECEIVED 2
```

Server-Side:

```
gcc server.c -o server
./server
Data Received: mckv
Data Received : ie
Data Received : de
Data Received : br
Data Received : up
```

ASSIGNMENT – EXTRA 1

Problem Statement: Write a program to develop a concurrent Echo Server using UDP socket. Use atleast two client program to implement it.

Theory:

Concurrent Server → A concurrent server is such a server which handles multiple clients at the same time.

Code:

Client Side (1 & 2):

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#define CLIENT_IP "127.0.0.3"   // This is changed for other clients
#define CLIENT_PORT 5961        // This is changed for other clients
#define SERVER_IP "192.168.5.17"
#define SERVER_PORT 5568
int main(){
    struct sockaddr_in client,server;
    int sd, serv=sizeof(server);
    char str[512],str1[512];
    bzero((char*)&client, sizeof(client));
    client.sin_family = AF_INET;
    client.sin_port = htons(CLIENT_PORT);
    client.sin_addr.s_addr = inet_addr(CLIENT_IP);
    bzero((char*)&server, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_port = htons(SERVER_PORT);
    server.sin_addr.s_addr = inet_addr(SERVER_IP);
    bzero sd = socket(AF_INET, SOCK_DGRAM, 0);
      bind(sd,(struct sockaddr*)&client, sizeof(client));
```

```
do{
    printf("\nEnter a message to send to server: ");
    scanf("%s",str);
    sendto(sd, str, strlen(str)+1, 0, (struct sockaddr *)&server,
            sizeof(server));
    memset(str1, 0x0, 512);
    recvfrom(sd, str1, 512, 0,(struct sockaddr*)&server,&serr);
    printf("\nEchoed message received from server: %s", str1);
}while(strcmp(str,"stop"));
close(sd);
}
```

Server Side:-

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inent.h>
#define SERVER_IP "192.168.5.17"
#define SERVER_PORT 5568
int main(){
        struct sockaddr_in client,server;
        int sd,clen=sizeof(client);
        char str[512];
        bzero((char*)&server, sizeof(server));
        server.sin_family = AF_INET;
        server.sin_port = htons(SERVER_PORT);
        server.sin_addr.s_addr = inet_addr(SERVER_IP);
        sd=socket(AF_INET, SOCK_DGRAM, 0);
        bind(sd, (struct sockaddr*)&server, sizeof(server));
        while(1){
            do{
```

```
        memset(str, 0x0, 512);
        recvfrom(sd, str, 512, 0, (struct sockaddr*)&client, &clen);
        sendto(sd, str, strlen(str)+1, 0, (struct sockaddr*)&client,
                clen);
    } while(strcmp(str, "stop"));
  }
}
```

Output:

Client 1 Side

```
gcc client1.c -o client1
./client1
Enter a message to send to server: hello
Echoed message received from server: hello
```

Client 2 Side

```
gcc client2.c -o client2
./client2
Enter a message to send to server: mckvie
Echoed message received from server: mckvie
```

Server Side

```
gcc server.c -o server
./server
```

ASSIGNMENT - EXTRA 2

Problem Statement: Write a program using TCP Socket to implement an iterative chat server.

Theory:

Iterative Chat Server → This type of server handles requests/messages from the client and also sends replies from the receipients side. Here in our implementation the server side acts as the receipient of the client's requests/messages.

Code:

Client Side :-

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#define CLIENT_IP "127.0.0.1"
#define CLIENT_PORT 5971
#define SERVER_IP "192.168.5.17"
#define SERVER_PORT 5970
int main(){
        struct sockaddr_in client, server;
        int sd;
        char str[512], str1[512];
        bzero((char*)&client, sizeof(client));
        client.sin_family = AF_INET;
        client.sin_port = htons(CLIENT_PORT);
        client.sin_addr.s_addr = inet_addr(CLIENT_IP);
        bzero((char*)&server, sizeof(server));
        server.sin_family = AF_INET;
        server.sin_port = htons(SERVER_PORT);
        server.sin_addr.s_addr = inet_addr(SERVER_IP);
        sd = socket(AF_INET, SOCK_STREAM, 0);
        bind(sd, (struct sockaddr*)&client, sizeof(client));
```

```
    connect (sd, (struct sockaddr*)&server, sizeof (server));
    memset (str1, 0x0, 512, 0);
    recv(sd, str1, 512, 0);
    printf ("\n%s \n", str1);
    do{
        printf ("\n Enter a message to send to server : ");
        scanf (" %s ", str);
        send(sd, str, strlen(str)+1 , 0);
        memset (str1, 0x0, 512);
        recv(sd, str1, 512, 0);
        printf("Message from server: %s \n" , str1);
    }while (strcmp(str, "stop"));
    close (sd);
}
```

Server Side:

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define CLIENT_IP "127.0.0.1"
#define SERVER_IP "192.168.5.17"
#define SERVER_PORT 5970
int main( ){
        struct sockaddr_in client, server;
        int sd, nsd, clen = sizeof(client);
        char str[512], reply[512];
        bzero((char*)&server, sizeof(server));
        server.sin_family = AF_INET;
        server.sin_port = htons (SERVER_PORT);
        server.sin_addr.s_addr = inet_addr (SERVER_IP);
        sd = socket (AF_INET, SOCK_STREAM, 0);
        bind (sd, (struct sockaddr*)&server, sizeof(server));
        listen (sd, 5);
```

```
while(1){
    nsd=accept(sd,(struct sockaddr*)&client,&clen);
    strcpy(reply,"Welcome to Chat Server !!");
    send(nsd,reply,strlen(reply)+1,0);
    do{
        memset(str,0x0,512);
        recv(nsd,str,512,0);
        printf("\n Message received: %s",str);
        printf("\n Enter your reply: ");
        scanf("%s",reply);
        send(nsd,reply,strlen(reply)+1,0);
    }while(strcmp(str,"stop"));
    close(nsd);
}
}
```

Output:

Client Side

```
gcc client.c -o client
./client
Welcome to Chat Server !!
Enter a message to send to server: hello
Message from server: hi
Enter a message to send to server: nice
Message from server: thanks
```

Server Side

```
gcc server.c -o server
./server
Message received: hello
Enter your reply: hi
Message received: nice
Enter your reply: thanks
```