ASSIGNMENT - 1

Commands, Functions and Outputs

1. ipconfig

The ipconfig command displays information about the host (the computer you are sitting at) computer TCP/IP configuration.

Output-

```
Ethernet adapter local area connection :
IPv4 Address . . . . . : 13.0.1.84
Subnet Mask . . . . . : 255:255:0.0
Default Gateway . . . . . : 13.0.0.1
```

2. ipconfig/all

This command detailed configuration information about your TCP/IP connections including router, gateway, DNS, DHCP and type of Ethernet.

Output-

```
Host Name . . . . . : SYS421
Primary DNS suffix . . . . . :
IP Routing Enabled . . . . . : NO
WINS Proxy Enabled . . . . . : NO
```

Ethernet adapter Local Area Connection:

Connection-specific DNS Suffix . :
Description . . . . . : Realtek PCIe GBE Family
Physical Adress . . . . . : 1C-1B-0D-82-2F-1D
DHCP Enable . . . . . : Yes
Auto Configuration Enabled . . . . . : Yes
IPv4 Adress . . . . . : 13.0.1.84 (Preferred)
Subnet Mask . . . . . : 255.255.0.0
Lease Obtained . . . . . : Friday, August 06, 2021
Lease Expires . . . . . : Sunday, October 01, 2022
Default-Gateway . . . . . : 13.0.0.1
DHCP Server . . . . . : 13.0.0.1
DNS Servers . . . . . : 13.0.0.1
NetBIOS over Tcpip . . . . . : Enabled

3. nslookup

nslookup is used for diagnosing DNS problems. If you can access a resource by specifying an IP adress but not its DNS you have a DNS problem.

Output

Default Server : B1-LAB-Server
Adress : 13.0.0.1

> www.iemcnp.com

Server : B1 - LAB - SERVER

Address : 13.0.0.1

Non - authoritative answer :

Name : iem-prd-elb-551247763.us-east-1.elb.amazonaws.com

Addresses : 54.163.142.205
54.145.129.128

Aliases : www.iemcnp.com

>

4. ping <ip>

The ping command sends one datagram per seconds and prints one line of output for every response received. The ping command calculates round-trip times and packet loss statistics, and displays a brief summary on completion.

Output

pinging 13.0.0.1 with 32 bytes of data :
Reply from 13.0.0.1 : bytes = 32 time <1ms TTL = 64
Reply from 13.0.0.1 : bytes = 32 time <1ms TTL = 64
Reply from 13.0.0.1 : bytes = 32 time <1ms TTL = 64
Reply from 13.0.0.1 : bytes = 32 time <1ms TTL = 64
Ping statistics for 13.0.0.1 :
Packets : Sent = 4, Received = 4, Lost = 0 <0% loss>,
Approximate round trip times in milli-seconds :
Minimum = 0ms, Maximum = 0ms, Average = 0ms

5. tracert <ipaddress>

The tracert command displays a list of all the routers that a packet has to go through to get from the computer where tracert is run to any other computer on the internet.

Output

Tracing route to B1-LAB-SERVER [13.0.0.1]
over a maximum of 30 hops:

1 <1ms <1ms <1ms B1-LAB-SERVER [13.0.0.1]

Trace Complete.

6. netstat

Netstat displays a variety of statistics about a computer TCP/IP connections. This tool is useful when you are having trouble with TCP/IP applications such as HTTP and FTP.

Output

Active Connections

Proto	Local Address	Foreign Address	State
TCP	13.0.1.84:63746	ec2-54-145-129-128:https	ESTABLISHED
TCP	13.0.1.84:63747	ec2-54-145-129-128:https	ESTABLISHED

NETWORKING CABLES

- **CAT5** - Category 5 cable is twisted pair cable for carrying signals. This type of cable used in structured cabling for computer networks such as Ethernet. The cable provides performance of upto 100 MHz and is suitable for 10BASE-T, 100BASE-TX (Fast Ethernet) and 1000BASE-T (Gigabit Ethernet). CAT5 is also used to carry other signals such as telephony and video.

- **UTP** - Unshielded twisted pairs, a UTP cable is used in computer networking that consists of two shielded wires twisted around each other. As the name would imply, these cables do not have insulations (shielding) between each of the paired wires. Consequently, they do not block electromagnetic interference, resulting in a higher risk of packet-loss or corruption. Used in Ethernet cables and telephone lines.

Category	Use (Bandwidth)
1	Analog voice grade (0.4 MHz)
2	Older terminal systems up to 4 Mbps (4 MHz)
3	Digital Transmission up to 10 Mbps (16 MHz)
4	" " " " 16 Mbps (20 MHz)
5	" " " " 100 Mbps (100 MHz)
5e	" " " " 1 GBps (100 MHz)
6	" " " " 10 Gbps (250 MHz)
6a	" " " " 10 Gbps (500 MHz)
7	" " " " 10 Gbps (600 MHz)
7a	" " " " 10 Gbps (1000 MHz)
8	" " " " 40 Gbps (2000 MHz)
8.2	" " " " 40 Gbps (2000 MHz)

→ Connectors

- **RJ45** : A registered jack (RJ) is a standard physical network interface for connecting telecommunication or data equiptment. The physical connector that registered jacks used are mainly of the modular connector and 50-pin miniature ribbon connector types. The most common twisted-pair connector is an 8-position, 8-contact (8P8C) modular plug and jack commonly referred to as an RJ45 connector.

- **I-Connector** : I-Connectors can be used to split radio frequency power from a cable into two. They can be used to attach a piece of electronic test equiptment. I-connector were used on 10BASE2 Ethernet Network.

→ **Hubs** : A network hub is a node that broadcasts data to every computer or Ethernet-based device connected to it. A hub is less sophisticated than a switch, the later of which can isolate data transmissions to a specific devices. Network hubs are best suited for small, simple local area network (LAN) enviroments. They cannot filter the signals.

→ **Switches** : Switches are network devices operating at layer 2 or the data-link layer of OSI model. They connect devices in a network and use packet switching to send, receives or forwards data packets or data frames over the network.

A switch has many ports, to which computers are plugged in. When a data frame arrives at any port of network switch, it examines the destination address, performs necessary checks and sends the frame to the corresponding devices. It supports unicast, multicast as well as broadcast communication.

ASSIGNMENT - 2

1. Configure the host name of the switch as SW1.

```
Switch >ENABLE
Switch #CONFIGT
Enter configuration commands, one per line. End with CNTL/Z.
Switch (config) #hostname SW1
SW1(config)#S.
```

2. Set a message of the day (MOTD) banner for the switch -

```
**********
SW1 (config)#banner ?
  word  set message of the day banner
SW1(config)#  banner MOTD?
    LINE C banner-text C, where 'C' is a delimiting character.
SW1(config)#banner word $*******
Enter TEXT message. End with the character '$'
*********
  Only Authorized User Allowed
**********
$
SW1 (config)#exit
SW1 #
%SYS-5-CONFIG_I : Configured from console by console
exit
```

SW1 cons is now available
Press RETURN to get started

* * * * * * * * *
* * * * * * * * * * * * * *
Only Authorized Users Allowed
* * * * * * * * * * * * * * *

3) i) Configure a line console password - India@123

```
SW1>en
SW1#sh run
    Building configuration.....
    Current configuration !! 211 bytes
    !
    version 12.2
    no service time stamps log date time msec
    no service time stamp debug datetime msec
    no service password - encryption
    !
    .
    .
    .
SW1#config t
Enter configuration commands, one per line. End with CNTL/Z
SW1(config)# line con 0
SW1(config-line)# password India@123
SW1(config-line)
```

3) ii) Configure an enable secret password - VEM@123

Sw1 (config)# exit

Sw1 #

%SYS-5-CONFIG_I : Configured from console by console Sw1

configt

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

Sw1 (config)# enable secret VEM@123

Sw1 (config) # line con 0 (to check)

Sw1 (config-line)# login

* * * * * * * * * * * * *

Only Authorized User Allowed

* * * * * * * * * * * *

User access verification

Password : India@123

Press return to get started

Sw1 > enable

Password : VEM@123

Sw1 # configt

ASSIGNMENT - B

1. Write a program to find address of the system.

```
import socket
print("Host Name :", socket.gethostname(), "\n IP Address :", socket.gethostbyname(socket.gethostname()))
```

2. Write a socket program for implementation of echo.

Server side code :

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("127.0.0.1", 9090))
s.listen()
(c, cip) = s.accept()
c.send(c.recv(1024))
s.close()
```

Client- side code :

```
import socket
c = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
c.connect(("127.0.0.1", 9090))
data = input()
c.send(data.encode())
dataFromServer = c.recv(1024)
print(dataFromServer.decode())
c.close()
```

3. Write a client – server application for chat using TCP.

Server side code

```
import socket
s = socket.socket(socket.AF_INET, sock.SOCK_STREAM)
s.listen()
while True:
    (c, cip) = s.accept()
    data = c.recv(1024).decode()
    print("Client:", data)
    data = input("Enter text:")
    c.send(data.encode())
```

Client side code

```
import socket
c = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
c.connect(("127.0.0.1", 9090))
while True:
    data = input("Enter text:")
    c.send(data.encode())
    data = c.recv(1024).decode()
    print("Server:", data)
```

4. Write a program using client server socket programming:
Client needs to authenticate itself by entering a server defined string as a password (like OTP) and then to say Hi to server.
Server replies with Hello. Press any key to exit.

Server side code

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("127.0.0.1", 9090))
s.listen()
(c, ip) = s.accept()
c.send("Enter OTP".encode())
otp = c.recv(1024).decode()
if otp == '8894':
    c.send("You are authenticated".encode())
    data = c.recv(1024).decode()
    print("Client:"data)
    data = input("Enter text:")
    c.send(data.encode())
else:
    c.send("You are authenticated".encode())
s.close()
```

client side code :

```
import socket
c = socket.socket(socket.AF_INET , socket.SOCK_STREAM)
c.connect(("127.0.0.1", 9090))
print
data = c.recv(1024).decode()
print(data, end = " ")
otp = input()
c.send(otp.encode())
data = c.recv(1024).decode()
print(data)
if data == " You are Authenticated ":
    data = input(" Enter text :")
    c.send(data.encode())
    data = c.recv(1024).decode()
    print(" Server :", data)
else :
    c.close()
```

ASSIGNMENT - 4

Write a program to perform file transfer in client and server using TCP/IP.

Server side code:

```
import socket
import tqdm
import os

BUFFER_SIZE = 1024
s = socket.socket()
s.bind(('127.0.0.1', 5001))
s.listen(1)
print("Listening as 127.0.0.1:5001")
client_socket, address = s.accept()
print(f"Connected to {address[0]}:{str(address[1])}")
filename, filesize =
client_socket.recv(BUFFER_SIZE).decode().split('||')
filename = os.path.basename(filename)
filesize = int(filesize)
progress = tqdm.tqdm(range(filesize), f"Receiving
{filename}", unit="B", unit_scale=True,
unit_divisor=1024)
with open(filename, "wb") as f:
    while True:
        bytes_read = client_socket.recv(BUFFER_SIZE)
        if not bytes_read:
            break
        f.write(bytes_read)
        progress.update(len(bytes_read))
```

```
client_socket.close()
s.close()
```

Client-Side Code :

```
import socket
import tqdm
import os

BUFFER_SIZE = 1024
filename = input("Enter the file path or filename : ")
filesize = os.path.getsize(filename)
s = socket.socket()
print("Connecting to 127.0.0.1:5001")
s.connect(("127.0.0.1", 5001))
print("Connected.")
s.send(f"{filename}||{filesize}".encode())
progress = tqdm.tqdm(range(filesize), f"Sending
{filename}", unit="B", unit_scale=True,
unit_divisor=1024)
with open(filename, "rb") as f:
    while True:
        bytes_read = f.read(BUFFER_SIZE)
        if not bytes_read:
            break
        s.sendall(bytes_read)
        progress.update(len(bytes_read))

s.close()
```

Assignment - 05

1) **Write a program to Perform File Transfer in Client & Server Using TCP/IP.**

Server side code:

```
import socket
import tqdm
import os

BUFFER_SIZE = 1024
s = socket.socket()
s.bind(('127.0.0.1', 5001))
s.listen(1)
print("Listening as 127.0.0.1:5001")
client_socket, address = s.accept()
print(f"Connected to {address[0]}:{str(address[1])}")
filename, filesize =
client_socket.recv(BUFFER_SIZE).decode().split('||')
filename = os.path.basename(filename)
filesize = int(filesize)
progress = tqdm.tqdm(range(filesize), f"Receiving
{filename}", unit="B", unit_scale=True,
unit_divisor=1024)
with open(filename, "wb") as f:
    while True:
        bytes_read = client_socket.recv(BUFFER_SIZE)
        if not bytes_read:
            break
        f.write(bytes_read)
        progress.update(len(bytes_read))
client_socket.close()
s.close()
```

Client side code:

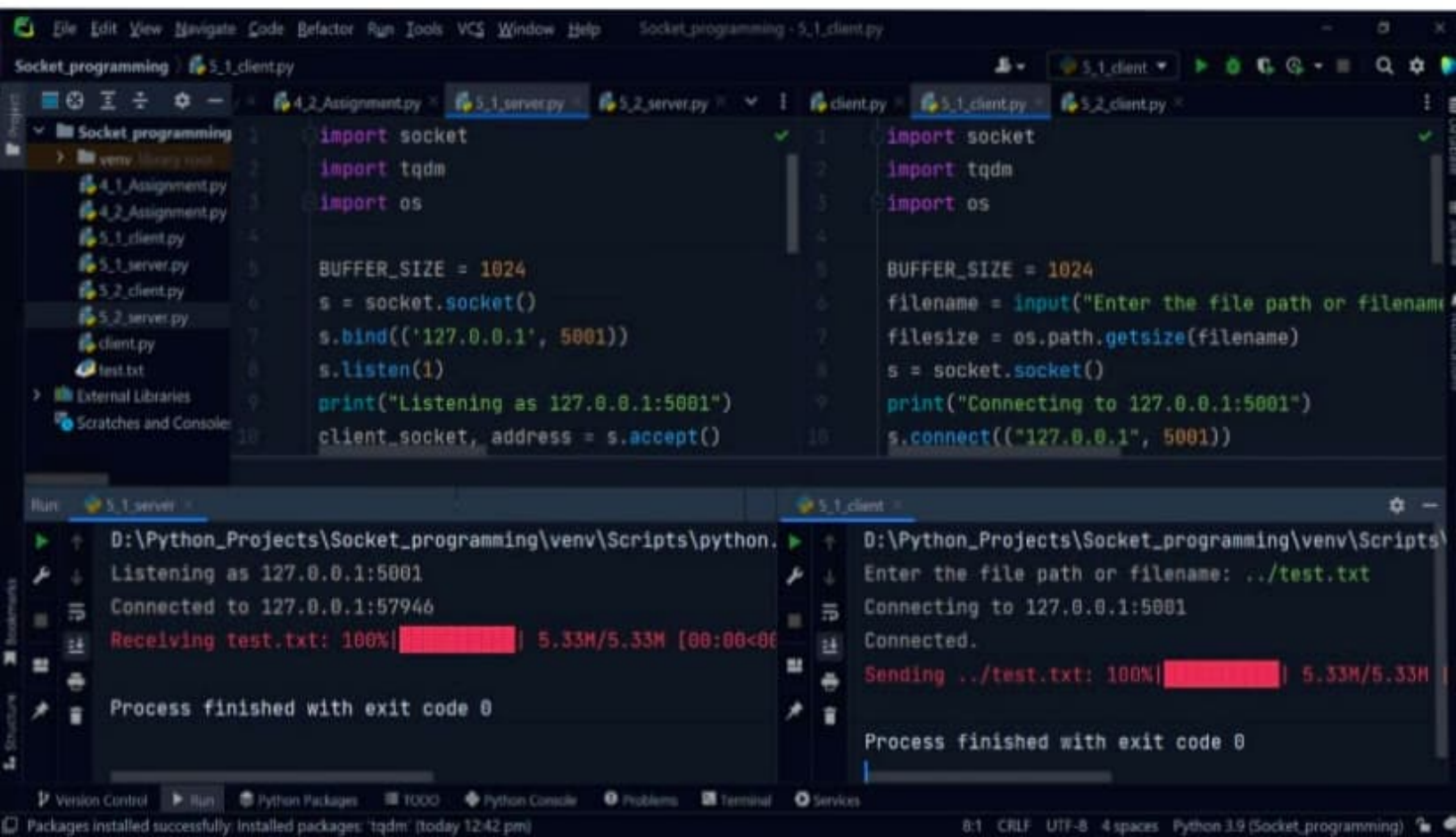
```
import socket
import tqdm
import os

BUFFER_SIZE = 1024
filename = input("Enter the file path or filename: ")
filesize = os.path.getsize(filename)
s = socket.socket()
print("Connecting to 127.0.0.1:5001")
s.connect(("127.0.0.1", 5001))
print("Connected.")
s.send(f"{filename}||{filesize}".encode())
```

```
progress = tqdm.tqdm(range(filesize), f"Sending
{filename}", unit="B", unit_scale=True,
unit_divisor=1024)
with open(filename, "rb") as f:
    while True:
        bytes_read = f.read(BUFFER_SIZE)
        if not bytes_read:
            break
        s.sendall(bytes_read)
        progress.update(len(bytes_read))
s.close()
```

Output:

```
progress = tqdm.tqdm(range(filesize), f"Sending
{filename}", unit="B", unit_scale=True,
unit_divisor=1024)
with open(filename, "rb") as f:
    while True:
        bytes_read = f.read(BUFFER_SIZE)
        if not bytes_read:
            break
        s.sendall(bytes_read)
        progress.update(len(bytes_read))
s.close()
```

Output:



2) **Write a program to implement Remote Command Execution (RCE)**

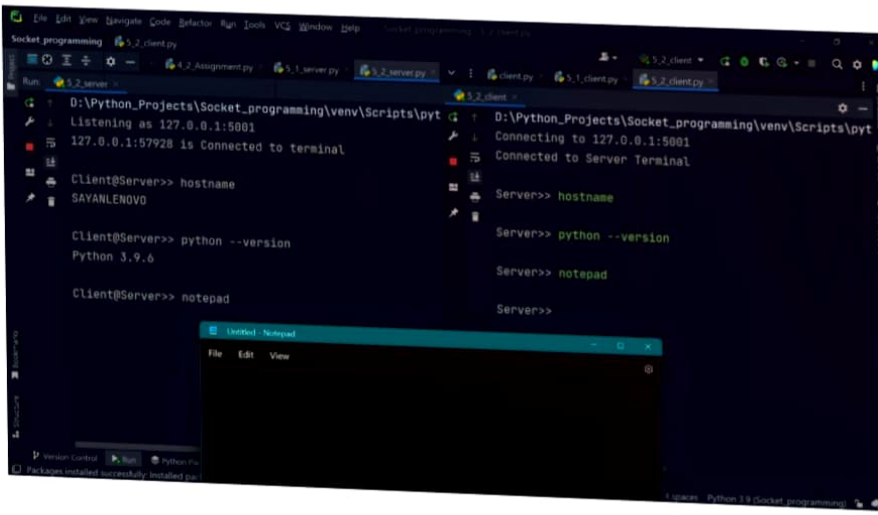
Server side code:

```
import socket
import os

BUFFER_SIZE = 4096
s = socket.socket()
s.bind(('127.0.0.1', 5001))
s.listen(1)
print("Listening as 127.0.0.1:5001")
client_socket, address = s.accept()
print(f"{address[0]}:{str(address[1])} is Connected to
terminal")
while True:
    print("\nClient@Server>>", end=" ")
    command = client_socket.recv(BUFFER_SIZE).decode()
    print(command)
    if command == 'exit':
        break
    os.system(command)
print("Closing remote connection with client")
client_socket.close()
s.close()
```

Client side code:

```
import socket

BUFFER_SIZE = 4096
s = socket.socket()
print("Connecting to 127.0.0.1:5001")
s.connect(("127.0.0.1", 5001))
print("Connected to Server Terminal")
while True:
    command = input("\nServer>> ")
    s.sendall(command.encode())
    if command == 'exit':
        break
print("Closing remote connection with Server")
s.close()
```

Output:



ASSIGNMENT - 7

- **Simulation of ARP :**

Server

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <string.h>

main(){
    int shmid, a, i;
    char *ptr, *shmptr;
    shmid = shmget(3000, 10, IPC_CREAT|0666);
    shmptr = shmat(shmid, NULL, 0);
    ptr = shmptr;
    for (i=0; i<3; i++){
        puts("Enter the name");
        scanf("%s", ptr);
        a = strlen(ptr);
        printf("string length : %d", a);
        ptr[a] = '.';
        puts("Enter ip.");
        ptr = ptr + a + 1;
    }
    ptr[strlen(ptr)] = '\0';
    printf("\n ARP table at serverside is = \n %s", shmptr);
    shmdt(shmptr);
}
```

Client

```
#include<stdio.h>
#include<string.h>
#include<sys/types.h>
#include<sys/shm.h>
```

```
main()
{
 int shmid, a;
 char *ptr, *shmptr;
   char ptr2[51], ip[12], mac[26];
   shmid = shmget(3000, 10, 0666);
   shmptr = shmat(shmid, NULL, 0);
   puts("The ARP table is :");
   printf("%s", shmptr);
   printf("\n1. ARP\n2. RARP\n3. Exit\n");
   scanf("%d", &a);
   switch(a)
   {
    case 1:
      puts("Enter ip address :");
      scanf("%s", ip);
      ptr = strstr(shmptr, ip);
      ptr -= 8;
      sscanf(ptr, "%s %s", ptr2);
      printf("mac addr is :%s", ptr2);
       break;
    case 2:
      puts("Enter mac addr");
      scanf("%s", mac);
      ptr = strstr(shmptr, mac);
      sscanf(ptr, "%s %s %s", ptr2);
      printf("%s", ptr2);
       break;
    case 3:
      exit(1);
   }
 }
```

ASSIGNMENT - 8

• Implementation of RMI :

Server

```
import java.rmi.*;
import java.rmi.server.*;
public class remoServer extends UnicastRemoteObject
implements remointer
{
 public remoServer() throws RemoteException {
    super();
 }
 public String message() throws remoteException {
    return "Hi";
 }
 public static void main (String args[])
 {
   try
   {
   remoServer a = new remoServer();
   Naming.rebind("TestServer", a);
   System.out.println("The server is ready");
   }
   catch (Exception e)
   {
   }
   }
   }
```

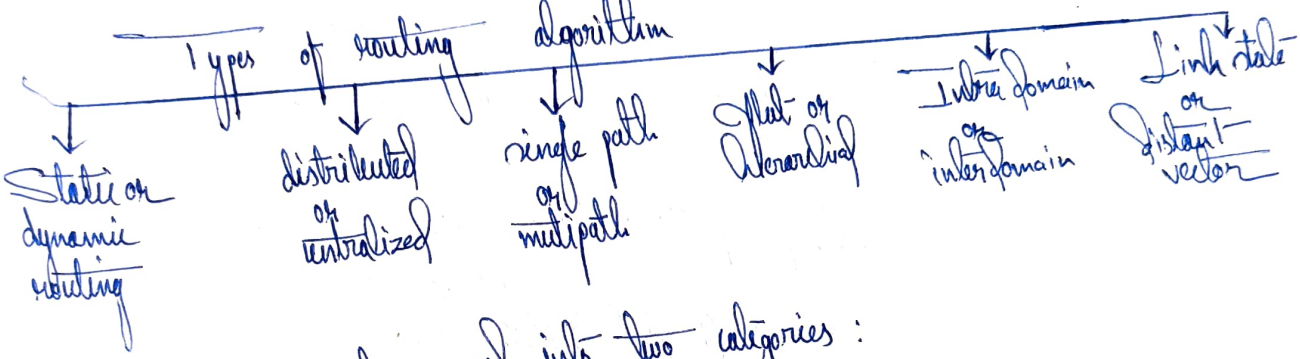
Client-

```
import java.rmi.*;
import java.rmi.registry.*;
public class remoClient
{
  public static void main (String args[])
{
  try {
  remoInter s = (remoInter)
  Naming.lookup ("rmi-Server");
   System.out.println (s.message[]);
  }
  catch (Exception e)
   {
   }
  }
}
```

ASSIGNMENT - 9

• Case study of different routing algorithm :

A routing algorithm is a routing protocol determined by network layer for transmitting data packets from source to destination. The algorithm determines the best or least cost path for data transmission from sender / source to receiver / destination.



Further routing can be grouped into two categories :

i) Non-adaptive routing : Once the pathway to the destination has been selected, the router sends all the packets for that destination along that one route. The routing decisions are not made based on the condition or topology of the network.

ii) Adaptive routing : A router may select a new route for each packet in response to change in the condition and topology of the networks.

✓ Shortest path routing –
It comes under adaptive routing.
eg. Dijkstra's algo

✓ Distance vector routing –
It comes under non-adaptive routing.
eg. Bellman Ford.

✓ Flooding Algorithm – It is a non-adaptive algorithm or static algorithm.

ASSIGNMENT - 10

- Implement the datalink layer framing methods such as character count, character stuffing and bit stuffing.

```
#include <stdio.h>
#include <string.h>
main()
{
int a[20], b[30], i, j, k, count, n;
printf("Enter frame length:");
scanf("%d", &n);
printf("Enter input frame (0's & 1's only):");
for (i=0; i<n; i++)
scanf("%d", &a[i]);
i = 0, count = 1; j=0;
while (i<n)
{
if (a[i] == 1)
{
b[j] = a[i];
for (k = i+1; a[k]==1 && k<n && count <5; k++)
{
j++;
b[j] = a[k];
count++;
if (count == 5){
j++; b[j] = 0;
}
i = k;
}}
else {b[j] = a[i];}
i++;
j++;
}
printf("After stuffing the frame is :");
for (i=0; i<j; i++)
printf("%d", b[i]);
}
```

- Write a client server socket programming : write a client — server cyclic redundancy check (CRC) program steps to follow :

a) client sends a message to server with an appendent CRC.
b) server checks the data for any error & accepts it.
c) server replies with 'good data' / 'bad data' depending upon there's no error or with error.

Server :

```
import socket :
def xor(a,b):
    result = []
    for i in range(1, len(b)):
        if a[i] != b[i]:
            result.append('0')
        else:
            result.append('1')
    return ''.join(result)
def mod2div(dividend, divisor):
    pic = len(divisor)
    tmp = dividend[0 : pick]
    while pick < len(dividend):
        if tmp[0] == '1':
            tmp = xor(divisor, tmp) + dividend[pick]
        else:
            tmp = xor('0'*pick, tmp) + dividend[pick]
        pick += 1
        if tmp[0] == '1':
            tmp = xor(divisor, tmp)
        else:
            tmp = xor('0'*pick, tmp)
    checkword = tmp
    return checkword
```

```
def encodeData(data, key):
    L_key = len(key)
    appended_data = data + '0' * (L_key - 1)
    remainder = mod2div(appended_data, key)
    codeword = data + remainder
    return codeword
s = socket.socket()
port = 12345
s.connect(('127.0.0.1', port))
input_string = input("Enter data you want to send :")
data = ''.join(format(ord(x), 'b') for x in input_string))
print("Entered data in binary format :", data)
key = "1001"
ans = encodeData(data, key)
print("Encoded data to be sent to server in binary format :" ans)
result(ans.encode(), ("127.0.0.1", 12345))
print("Received feedback from server :", s.recv(1024).decode())
s.close()
```

client

```
import socket
def xor(a,b):
    result = []
    for i in range(1, len(b)):
        if a[i]==b[i]:
            result.append('0')
        else:
            result.append('1')
        return ''.join[result]
def mod2div(dividend, divisor):
    pick = len(divisor)
    tmp = dividend[0 = pick]
```

```
while pick < len(divident):
    if tmp[0] == '1':
        tmp = xor(divisor, tmp) + divident[pick]
    else:
        tmp = xor('0'*pick, tmp) + divident[pick]
    pick += 1
if tmp[0] == '1':
    tmp = xor(divisor, tmp)
else:
    tmp = xor('0'*pick, tmp)
checkword = tmp
return checkword
def decodeData(data, key):
    l_key = len(key)
    appended_data = data.decode() + '0'*(l_key-1)
    remainder = mod2div(appended_data, key)
    return remainder
s = socket.socket()
print("socket successfully created")
port = 12345
s.bind(('', port))
print("socket binded to %s" %(port))
s.listen(5)
print("socket is listening")
while True:
    c, addr = s.accept()
    print('Got connection from', addr)
    data = c.recv(1024)
    print("Received:", data.decode())
    if not data:
        break
```

```
key = "1001"
ans = decodeData(data, key)
print("Remainder after decoding is : " +ans)
temp = "0" * (len(key)-1)
if ans == temp :
    c.sendto(("Thanks you" + data.decode() + "Received").encode(),
    ('127.0.0.1', 12345))
else :
    c.sendto(("Error").encode(),('127.0.0.1', 12345))
    c.close()
```

ASSIGNMENT - 11

- Implement on a data set of characters the three CRC polynomials CRC12, CRC16 and CRC CCIP.

```
#include <stdio.h>
int gen[4], genl, fel, rem[4];
main(){
 int i, j, fr[8], dupfr[11], recfr[11], then, flag;
 frl = 8; genl = 4;
 printf("Enter frame : ");
for (i=0 ; i<frl ; i++){
 scanf("%d", &fr[i]);
 dupfr[i] = fr[i];
 }
printf("Enter generator : ");
for (i=0 ; i<genl ; i++)
 scanf("%d", &gen[i]);
 then = frl + genl - 1
 for (i=frl ; i<then ; i++){
 dupfr[i] = 0
 }
 remainder(dupfr);
for (i=0 ; i<frl ; i++){
 recfr[i] = fr[i];
 }
 remainder(recfr);
 flag = 0;
for (i=0 ; i<4 ; i++){
 if (rem[i] != 0) flag++;}
 if (flag == 0){
 printf("Frame received correctly");
}
```

```
else
  printf("The received frame is wrong");
}
remainder(int fr[])
{
 if (fr[k]==1){
    k1 = k;
    for(i=0; j=k; i<genl; i++; j++)
  {
    rem[i] = fr[j] . gen[i];
  }
  for(i=0; i<genl; i++)
  {
    fr[k1] = rem[i];
    k1++;
  }
 }
}
}
```