CODING:

SERVER:

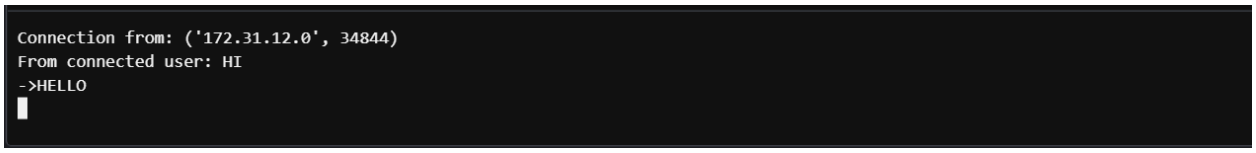
```
import socket
def server_program():
    host = socket.gethostname()
    port = 6969
    server_socket = socket.socket()
    server_socket.bind((host, port))
    server_socket.listen(5)
    conn, address = server_socket.accept()
    print ( "Connection from: " + str (address))
    while True:
        data = conn.recv(1024).decode()
        if not data:
            break
        print("From connected user: " + str (data))
        data = input('->')
        conn.send(data.encode())
    conn.close()


if __name__ == '__main__':
    server_program()
```

CLIENT:

```
import socket
def client_program():
    host = socket.gethostname()
    port = 6969
    client_socket = socket.socket()
    client_socket.connect((host, port))
    message = input (" -> ")
    while True:
        client_socket.send(message.encode())
        data = client_socket.recv(1024).decode()
        print('Received from server: ' + data)
        message = input (" -> ")
    client_socket.close()


if __name__ == '__main__':
    client_program()
```

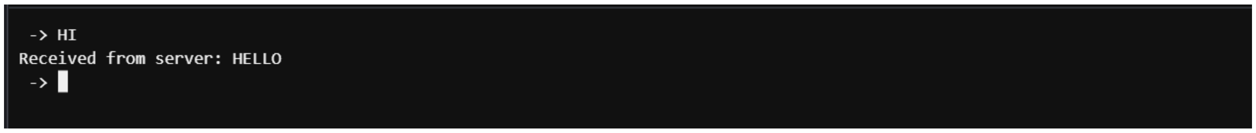
OUTPUT:

SERVER:



```
Connection from: ('172.31.12.0', 34844)
From connected user: HI
->HELLO
```

CLIENT:



```
 -> HI
Received from server: HELLO
 -> 
```

CODING:

SERVER:

```
import socket
localIP     = "127.0.0.1"
localPort   = 6191
bufferSize  = 1024
# Create a datagram socket
UDPServerSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
# Bind to address and ip
UDPServerSocket.bind((localIP, localPort))
print("UDP server up and listening")
# Listen for incoming datagrams
while(True):
    bytesAddressPair = UDPServerSocket.recvfrom(bufferSize)
    message = bytesAddressPair[0]
    address = bytesAddressPair[1]
    clientMsg = "Message from Client:{}".format(message)
    clientIP  = "Client IP Address:{}".format(address)
    print(clientMsg)
    print(clientIP)
    # Sending a reply to client
    bytesToSend         = message
    UDPServerSocket.sendto(bytesToSend, address)
```

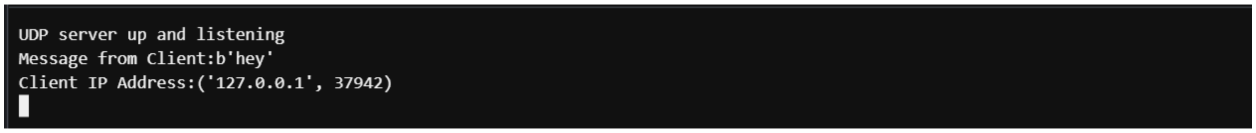
CLIENT:

```
import socket
msgFromClient       = input("Enter string:")
bytesToSend         = str.encode(msgFromClient)
serverAddressPort   = ("127.0.0.1", 6191)
```

```
bufferSize          = 1024
# Create a UDP socket at client side
UDPClientSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
# Send to server using created UDP socket
UDPClientSocket.sendto(bytesToSend, serverAddressPort)
msgFromServer = UDPClientSocket.recvfrom(bufferSize)
msg = "Message from Server: {}".format(msgFromServer[0])
print(msg)
```

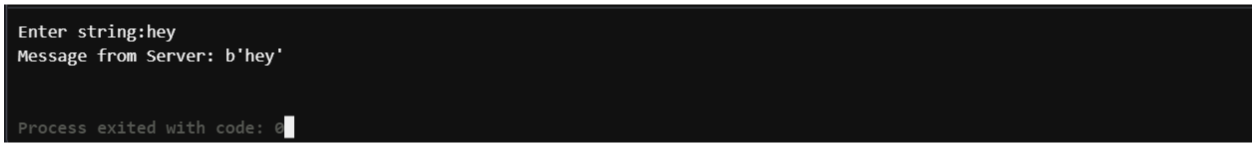
OUTPUT:

SERVER:



```
UDP server up and listening
Message from Client:b'hey'
Client IP Address:('127.0.0.1', 37942)
```

CLIENT:



```
Enter string:hey
Message from Server: b'hey'


Process exited with code: 0
```

CODING:

SERVER:

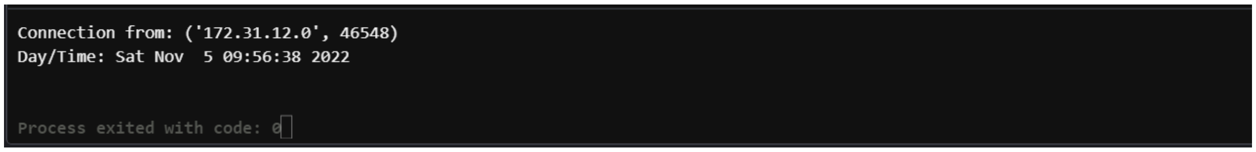
```
import socket
import time
def server_program():
    host = socket.gethostname()
    port = 6215
    server_socket = socket.socket()
    server_socket.bind((host, port))
    server_socket.listen(5)
    conn, address = server_socket.accept()
    print ( "Connection from: " + str (address))
    while True:
        ti = time.gmtime()
        data = (time.asctime(ti))
        conn.send(data.encode())
        print('Day/Time: ' + data)
        break
    conn.close()


if __name__ == '__main__':
    server_program()
```

CLIENT:

```
import socket
import time
def client_program():
    host = socket.gethostname()
    port = 6215
    client_socket = socket.socket()
    client_socket.connect((host, port))
    while (True) :
        data = client_socket.recv(1024).decode()
        print('Received from server: ' + data)
        break
    client_socket.close()


if __name__ == '__main__':
    client_program()
```

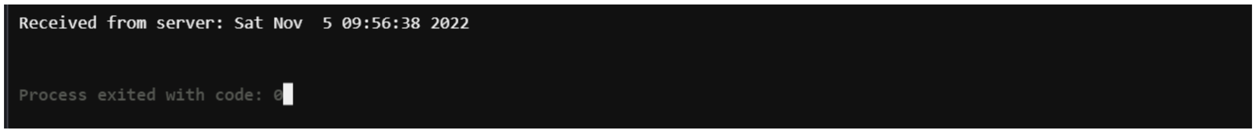
OUTPUT:

SERVER:



```
Connection from: ('172.31.12.0', 46548)
Day/Time: Sat Nov  5 09:56:38 2022


Process exited with code: 0
```

CLIENT:



```
Received from server: Sat Nov  5 09:56:38 2022


Process exited with code: 0
```

CODING:

SERVER:

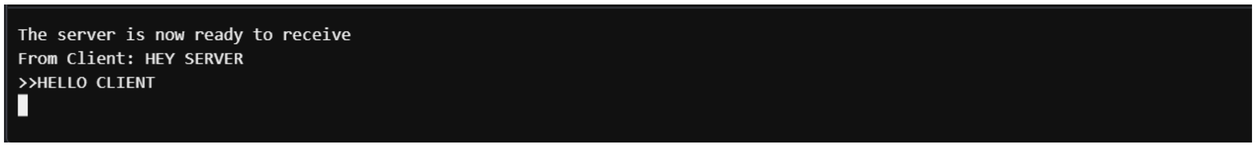
```
from socket import *
server_port = 6969
server_socket = socket(AF_INET,SOCK_STREAM)
server_socket.bind(('',server_port))
server_socket.listen(1)
print ("The server is now ready to receive")
connection_socket, address = server_socket.accept()
while True:
  sentence = connection_socket.recv(2048).decode()
  print('From Client:',sentence)
  message = input(">>")
  connection_socket.send(message.encode())
  if(message == 'q'):
    connectionSocket.close()
```

CLIENT:

```
from socket import *
server_name = 'localhost'
server_port = 6969
client_socket = socket(AF_INET, SOCK_STREAM)
client_socket.connect((server_name,server_port))
while True:
  sentence = input(">>")
  client_socket.send(sentence.encode())
  message = client_socket.recv(2048)
  print ("From Server:", message.decode())
  if(sentence == 'q'):
    client_socket.close()
```

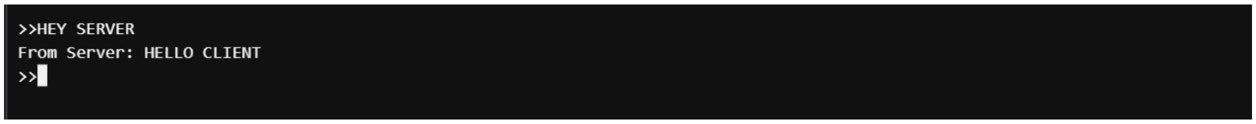
OUTPUT:

SERVER:



```
The server is now ready to receive
From Client: HEY SERVER
>>HELLO CLIENT
```

CLIENT:



```
>>HEY SERVER
From Server: HELLO CLIENT
>>
```

CODING:

SERVER:

```
import socket
import threading
import sys
FLAG = False
def recv_from_client(conn):
    global FLAG
    try:
        while True:
            if FLAG == True:
                break
            message = conn.recv(1024).decode()
            if message == 'quit':
                conn.send('quit'.encode())
                conn.close()
                print('Connection Closed')
                FLAG = True
                break
            print('Client: ' + message)
    except:
        conn.close()
def send_to_client(conn):
    global FLAG
    try:
        while True:
            if FLAG == True:
                break
            send_msg = input('')
            if send_msg == 'quit':
                conn.send('quit'.encode())
                conn.close()
                print('Connection Closed')
                FLAG = True
                break
            conn.send(send_msg.encode())
    except:
        conn.close()
def main():
    threads = []
    global FLAG
    HOST = 'localhost'
    serverPort = 6969
    serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    serverSocket.bind((HOST, serverPort))
    print('Socket binded')
    serverSocket.listen(1)
    print('Listening.....')
    connectionSocket, addr = serverSocket.accept()
    print('Connection Established with a Client on:', addr)
    t_rcv = threading.Thread(target=recv_from_client, args=(connectionSocket,))
    t_send = threading.Thread(target=send_to_client, args=(connectionSocket,))
    threads.append(t_rcv)
    threads.append(t_send)
    t_rcv.start()
    t_send.start()
    t_rcv.join()
    t_send.join()
    print('EXITING')
    serverSocket.close()
    sys.exit()
if __name__ == '__main__':
    main()
```

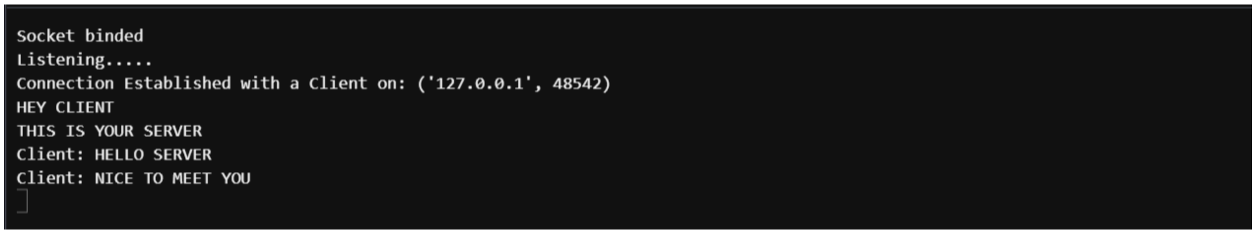
CLIENT:

```
import socket
import threading
import sys
FLAG = False
def send_to_server(clsock):
    global FLAG
    while True:
        if FLAG == True:
            break
        send_msg = input('')
        clsock.sendall(send_msg.encode())
def recv_from_server(clsock):
    global FLAG
    while True:
        data = clsock.recv(1024).decode()
        if data == 'quit':
            print('Closing connection')
            FLAG = True
            break
        print('Server:' + data)
```

```
def main():
    threads = []
    HOST = 'localhost';
    PORT = 6969
    clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    clientSocket.connect((HOST, PORT))
    print('Client is connected to the Server')
    t_send = threading.Thread(target=send_to_server, args=(clientSocket,))
    t_rcv = threading.Thread(target=recv_from_server, args=(clientSocket,))
    threads.append(t_send)
    threads.append(t_rcv)
    t_send.start()
    t_rcv.start()
    t_send.join()
    t_rcv.join()
    print('EXITING')
    sys.exit()
if __name__ == '__main__':
    main()
```

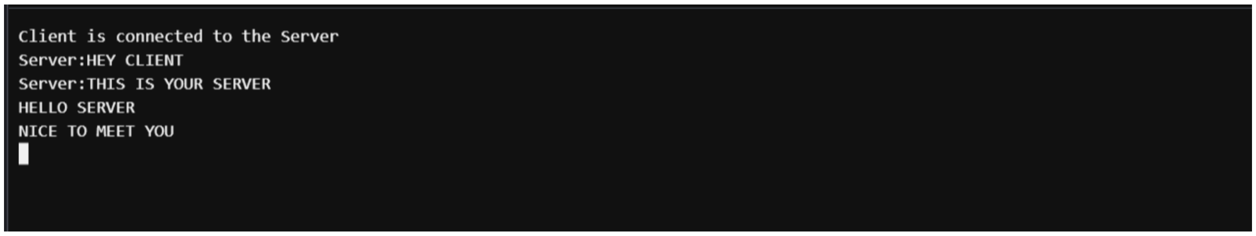
OUTPUT:

SERVER:



```
Socket binded
Listening.....
Connection Established with a Client on: ('127.0.0.1', 48542)
HEY CLIENT
THIS IS YOUR SERVER
Client: HELLO SERVER
Client: NICE TO MEET YOU
```

CLIENT:



```
Client is connected to the Server
Server:HEY CLIENT
Server:THIS IS YOUR SERVER
HELLO SERVER
NICE TO MEET YOU
```

CODING:

SERVER:

```
import socket
port = 6443
s = socket.socket()
host = socket.gethostname()
s.bind((host, port))
s.listen(5)
print ('Server listening....')
while True:
    conn, addr = s.accept()
    print ('Got connection from', addr)
    fileToBeOpened = conn.recv(1024).decode()

filename='/home/ubuntu/environment/RA2011003010113/SampleFilesEXP8/'+fileToBeOpened
    f = open(filename,'rb')
    l = f.read(1024)
    while (l):
       conn.send(l)
       print('Sent: ',l.decode('utf-8'))
       l = f.read(1024)
    f.close()
    print('Done sending')
conn.close()
```

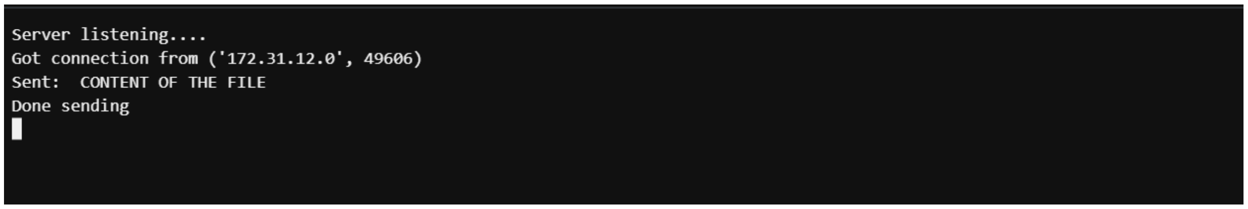
CLIENT:

```
import socket
s = socket.socket()
host = socket.gethostname()
port = 6443
```

```
s.connect((host, port))

filename = input('Enter Filename: ')

s.send(filename.encode())

print('Receiving data...')

while True:

    data = s.recv(1024).decode()

    print(data)

    fileReceived = data

filenameReceived='/home/ubuntu/environment/RA2011003010113/SampleFilesEXP8/Received
FilesEXP8/'+filename

    with open(filenameReceived,'w+',encoding = 'utf-8') as f:

        f.write(fileReceived)

        f.close()

    if not data:

        break

    print('Successfully got the file')

s.close()

print('Connection closed')
```

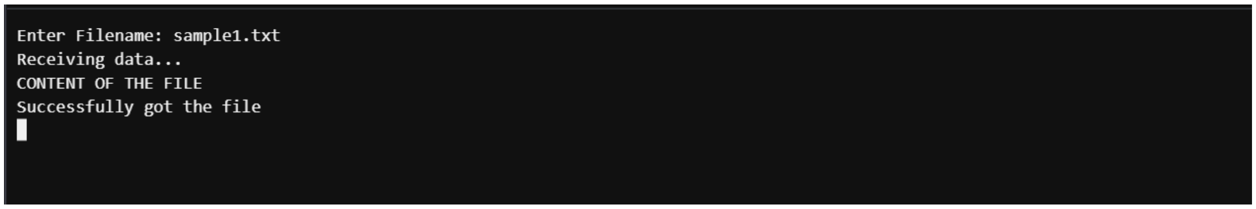
OUTPUT:

SERVER:



```
Server listening....
Got connection from ('172.31.12.0', 49606)
Sent:  CONTENT OF THE FILE
Done sending
```

CLIENT:



```
Enter Filename: sample1.txt
Receiving data...
CONTENT OF THE FILE
Successfully got the file
```

CODING:

SERVER:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<errno.h>
int main()
{
        int sd,acpt,len,bytes,port;
        char send[50],receiv[50];
        struct sockaddr_in serv,cli;
        if((sd=socket(AF_INET,SOCK_STREAM,0))<0)
        {
                printf("Error in socket\n");
                exit(0);}
        bzero(&serv,sizeof(serv));
        printf("Enter the port number:");
        scanf("%d",&port);
        serv.sin_family=AF_INET;
        serv.sin_port=htons(port);
        serv.sin_addr.s_addr=htonl(INADDR_ANY);
        if(bind(sd,(struct sockaddr *)&serv,sizeof(serv))<0) {
                printf("Error in bind\n");
```

```
		exit(0);}
	if(listen(sd,3)<0)
	{
		printf("Error in listen\n");
		exit(0);}
	if((acpt=accept(sd,(struct sockaddr*)NULL,NULL))<0)
	{
		printf("\n\t Error in accept");
		exit(0);}
	while(1)
	{
		bytes=recv(acpt,receiv,50,0);
		receiv[bytes]='\0';
		if(strcmp(receiv,"end")==0)
		{
			close(acpt);
			close(sd);
			exit(0);}
		else
		{
			printf("Command received:%s",receiv);
			system(receiv);
			printf("\n");}
	}
}
```

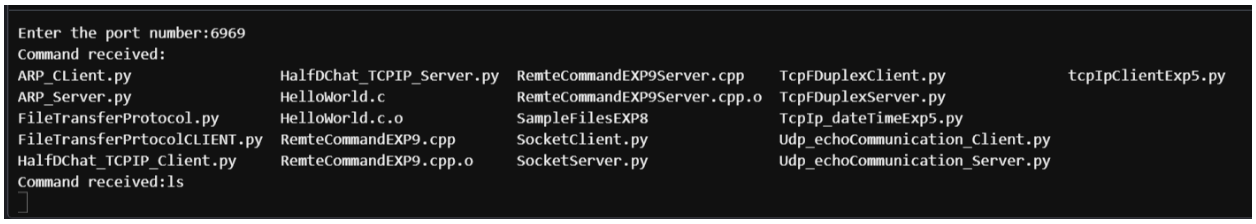
CLIENT:

```
#include<stdio.h>
#include<stdlib.h>
```

```
#include<string.h>
#include<unistd.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<errno.h>
int main() {
        int sd,acpt,len,bytes,port;
        char send1[50],receiv[50];
        struct sockaddr_in serv,cli;
        if((sd=socket(AF_INET,SOCK_STREAM,0))<0)
        {
                printf("Error in socket\n");
                exit(0);}
        bzero(&serv,sizeof(serv));
        printf("Enter the port number:");
        scanf("%d",&port);
        serv.sin_family=AF_INET;
        serv.sin_port=htons(port);
        serv.sin_addr.s_addr=htonl(INADDR_ANY);
        if(connect(sd,(struct sockaddr *)&serv,sizeof(serv))<0)
        {
                printf("Error in connection\n");
                exit(0);}
        while(1)
        {
                printf("Enter the command:");
                gets(send1);
                if(strcmp(send1,"end")!=0)
                {
                        send(sd,send1,50,0);}
                else
                {
                        send(sd,send1,50,0);
                        close(sd);
                        break;}
        }
}
```

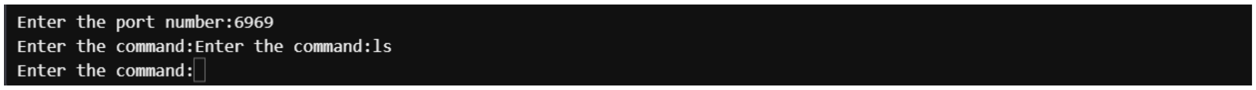
OUTPUT:

SERVER:



```
Enter the port number:6969
Command received:
ARP_CLient.py                HalfDChat_TCPIP_Server.py  RemteCommandEXP9Server.cpp    TcpFDuplexClient.py              tcpIpClientExp5.py
ARP_Server.py                HelloWorld.c               RemteCommandEXP9Server.cpp.o  TcpFDuplexServer.py
FileTransferProtocol.py      HelloWorld.c.o             SampleFilesEXP8               TcpIp_dateTimeExp5.py
FileTransferPrtocolCLIENT.py RemteCommandEXP9.cpp       SocketClient.py               Udp_echoCommunication_Client.py
HalfDChat_TCPIP_Client.py    RemteCommandEXP9.cpp.o     SocketServer.py               Udp_echoCommunication_Server.py
Command received:ls
```

CLIENT:



```
Enter the port number:6969
Enter the command:Enter the command:ls
Enter the command:
```

CODING:

SERVER:

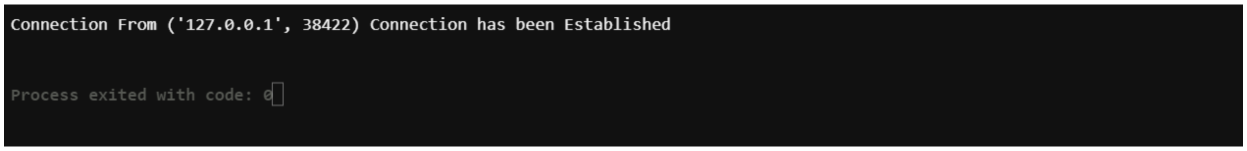
```
import socket
table = {
    '192.168.1.1':'1E.4A.4A.11',
    '192.168.2.1':'5E.51.4B.01',
    '192.168.1.3':'4B.35.CD.32',
    '192.168.4.1':'AF.4D.1F.FF',
    '192.168.3.2':'C3.C5.EE.C2',
}
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.bind(("",6968))
s.listen()
clientsocket,address = s.accept()
print("Connection From",address,"Connection has been Established")
ip = clientsocket.recv(1024)
ip = ip.decode("utf-8")
mac = table.get(ip,'No entry for given address')
clientsocket.send(mac.encode())
```

CLIENT:

```
import socket
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(('localhost',6968))
add = input('Enter IP: ')
s.send(add.encode())
mac = s.recv(1024)
mac = mac.decode('utf-8')
print('MAC of',add,' is: ',mac)
```

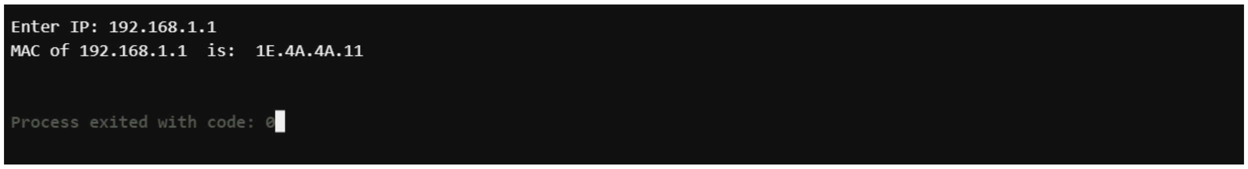
OUTPUT:

SERVER:



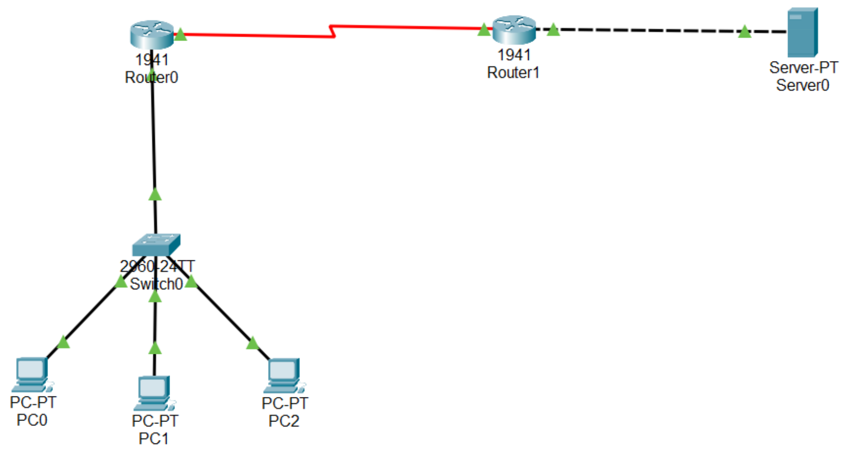
```
Connection From ('127.0.0.1', 38422) Connection has been Established


Process exited with code: 0
```

CLIENT:

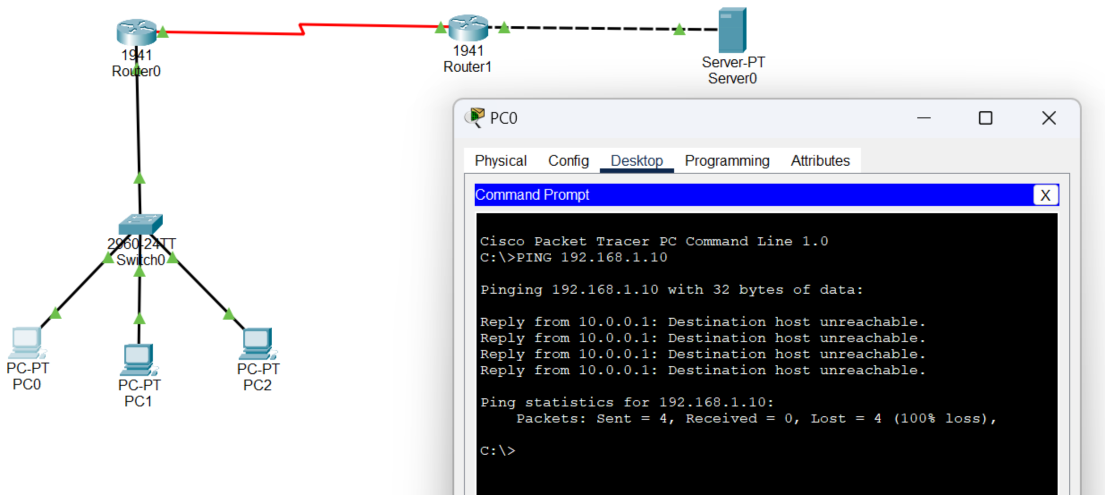


```
Enter IP: 192.168.1.1
MAC of 192.168.1.1  is:  1E.4A.4A.11


Process exited with code: 0
```

TOPOLOGY:

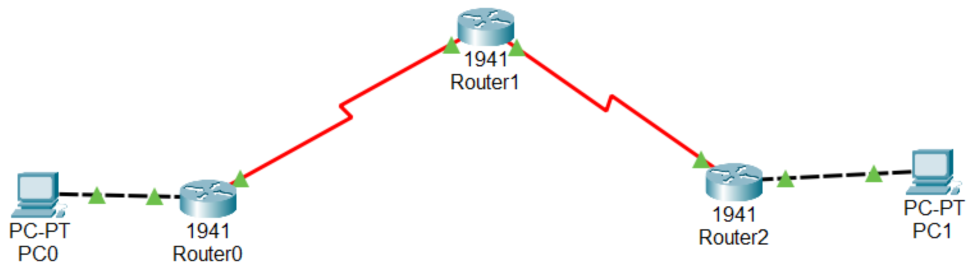


OUTPUT:

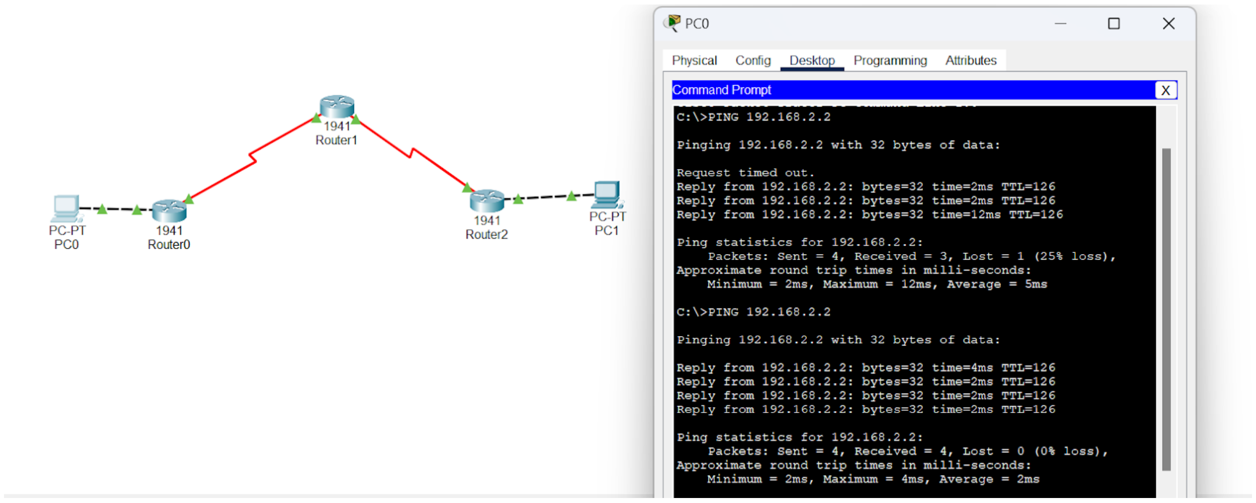


```
Cisco Packet Tracer PC Command Line 1.0
C:\>PING 192.168.1.10

Pinging 192.168.1.10 with 32 bytes of data:

Reply from 10.0.0.1: Destination host unreachable.
Reply from 10.0.0.1: Destination host unreachable.
Reply from 10.0.0.1: Destination host unreachable.
Reply from 10.0.0.1: Destination host unreachable.

Ping statistics for 192.168.1.10:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),

C:\>
```

TOPOLOGY:

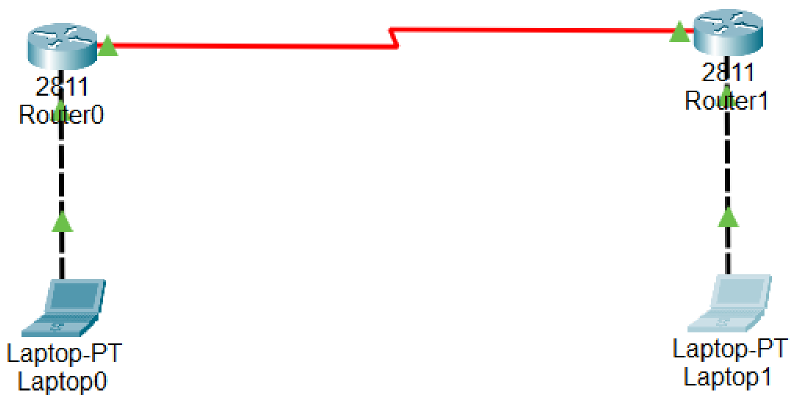


OUTPUT:

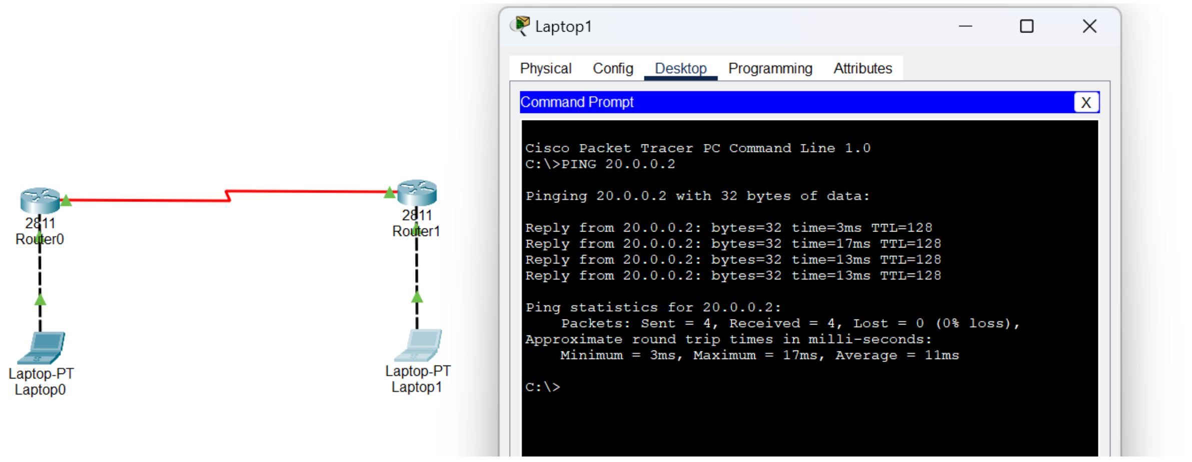


```
C:\>PING 192.168.2.2

Pinging 192.168.2.2 with 32 bytes of data:

Request timed out.
Reply from 192.168.2.2: bytes=32 time=2ms TTL=126
Reply from 192.168.2.2: bytes=32 time=2ms TTL=126
Reply from 192.168.2.2: bytes=32 time=12ms TTL=126

Ping statistics for 192.168.2.2:
    Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
    Minimum = 2ms, Maximum = 12ms, Average = 5ms

C:\>PING 192.168.2.2

Pinging 192.168.2.2 with 32 bytes of data:

Reply from 192.168.2.2: bytes=32 time=4ms TTL=126
Reply from 192.168.2.2: bytes=32 time=2ms TTL=126
Reply from 192.168.2.2: bytes=32 time=2ms TTL=126
Reply from 192.168.2.2: bytes=32 time=2ms TTL=126

Ping statistics for 192.168.2.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 2ms, Maximum = 4ms, Average = 2ms
```

TOPOLOGY:

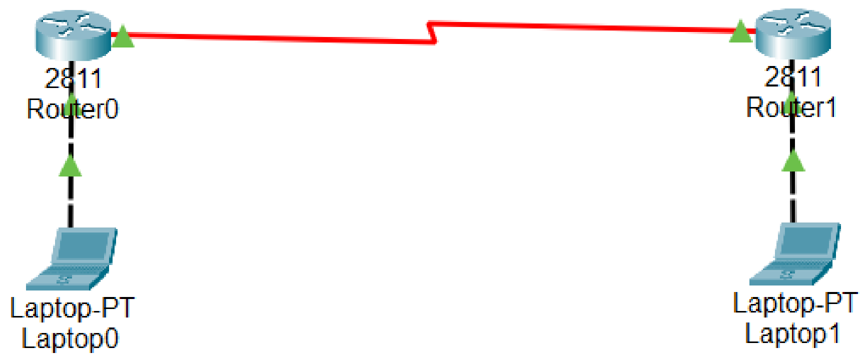


OUTPUT:

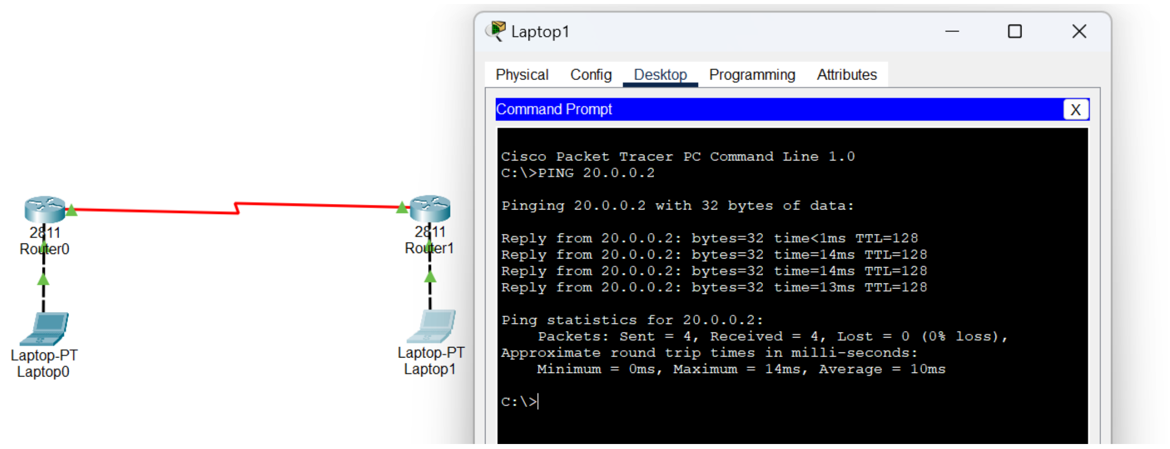


```
Cisco Packet Tracer PC Command Line 1.0
C:\>PING 20.0.0.2

Pinging 20.0.0.2 with 32 bytes of data:

Reply from 20.0.0.2: bytes=32 time=3ms TTL=128
Reply from 20.0.0.2: bytes=32 time=17ms TTL=128
Reply from 20.0.0.2: bytes=32 time=13ms TTL=128
Reply from 20.0.0.2: bytes=32 time=13ms TTL=128

Ping statistics for 20.0.0.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 3ms, Maximum = 17ms, Average = 11ms

C:\>
```

TOPOLOGY:



OUTPUT:



```
Cisco Packet Tracer PC Command Line 1.0
C:\>PING 20.0.0.2

Pinging 20.0.0.2 with 32 bytes of data:

Reply from 20.0.0.2: bytes=32 time<1ms TTL=128
Reply from 20.0.0.2: bytes=32 time=14ms TTL=128
Reply from 20.0.0.2: bytes=32 time=14ms TTL=128
Reply from 20.0.0.2: bytes=32 time=13ms TTL=128

Ping statistics for 20.0.0.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 14ms, Average = 10ms

C:\>
```