SRM
INSTITUTE OF SCIENCE AND TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)
DELHI-NCR CAMPUS, GHAZIABAD (U.P.)

Mini Project
Indian Traditional Knowledge
(18LEM109T)

B.Tech CSE) - 3rd Year/5th Semester

Name: C.SREEKAR

Registration no: RA2011026030098

DEPARTMENT OF COMPUTER SCIENCEAND ENGINEERING

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

NCR CAMPUS, DELHI MEERUT ROAD, MODINAGAR GHAZIABAD(U.P) - 201204

Odd Semester (2022-2023)

SRM
INSTITUTE OF SCIENCE AND TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)
DELHI-NCR CAMPUS, GHAZIABAD (U.P.)

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

Registration no: **RA2011026030098**

Certified to be the bonafide record of work done by C.SREEKAR

of 5th semester 3rd year. B.Tech degree course in SRM IST, NCR Campus of Department of Computer Science and Engineering, in Embedded System Design using Arduino laboratory during the academic year 2022-2023(Odd).

Lab In-charge

Head of the Department

Submitted for external examination held on date__/____/____at SRM IST, NCRCampus.

Examiner-1

Examiner-2

INDEX

S. No.	EXPERIMENT TITLE	DATE	SIGNATURE
1	Concurrent TCP/IP Day-Time Server		
2	Half Duplex Chat Using TCP/IP		
3	Implementation of File Transfer Protocol		
4	UDP Echo Client Server Communication		
5	DNS Server Using UDP		
6	ARP Implementation Using UDP		
7	Remote Command Execution Using UDP		
8	Full Duplex Chat Using TCP/IP		

LAB-1

Date: 09/09/2022

CONCURRENT TCP/IP DAY-TIME SERVER

GIVEN REQUIREMENTS:

There are two hosts, Client and Server. The Client requests the concurrent server for the date and time. The Server

sends the date and time, which the Client accepts and prints.

TECHNICAL OBJECTIVE:

To implement a TCP/IP day time server (concurrent server) that handles multiple client requests.

Once the client establishes connection with the server, the server sends its day-time details to the client which the

client prints in its console.

METHODOLOGY:

Server:

➢ Include the necessary header files.

➢ Create a socket using socket function with family AF_INET, type as

SOCK_STREAM.

➢ Initialize server address to 0 using the bzero function.

➢ Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to statically assigned port

number.

➢ Bind the local host address to socket using the bind function.

➢ Within a for loop, accept connection request from the client using accept function.

➢ Use the fork system call to spawn the processes.

➢ Calculate the current date and time using the ctime() function. Change the format so that it is

appropriate for human readable form and send the date and time to the client using the write function.

Client:

➢ Include the necessary header files.

➢ Create a socket using socket function with family AF_INET, type as

SOCK_STREAM.

➢ Initialize server address to 0 using the bzero function.

➢ Assign the sin_family to AF_INET.

➢ Get the server IP address from the console.

➢ Using gethostbyname function assign it to a hostent structure, and assign it to sin_addr of the server

address structure.

➢ Request a connection from the server using the connect function.

➢ Within an infinite loop, receive the date and time from the server using the read function and print the

date and time on the console.

CODE:

SERVER

```
# server.py
import socket
import time

# create a socket object
serversocket = socket.socket(
      socket.AF_INET, socket.SOCK_STREAM)

# get local machine name
host = socket.gethostname()

port = 9999

# bind to the port
serversocket.bind((host, port))

# queue up to 5 requests
serversocket.listen(5)

while True:
   # establish a connection
   clientsocket,addr = serversocket.accept()

   print("Got a connection from %s" % str(addr))
   currentTime = time.ctime(time.time()) + "\r\n"
   clientsocket.send(currentTime.encode('ascii'))
   clientsocket.close()
```

CLIENT

```
# client.py
import socket

# create a socket object
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# get local machine name
host = socket.gethostname()

port = 9999

# connection to hostname on the port.
s.connect((host, port))

# Receive no more than 1024 bytes
tm = s.recv(1024)

s.close()

print("The time got from the server is %s" % tm.decode('ascii'))
```

OUTPUT

In [*]:

```
# server.py
import socket
import time
# create a socket object
serversocket = socket.socket(
        socket.AF_INET, socket.SOCK_STREAM)
# get local machine name
host = socket.gethostname()
port = 9999
# bind to the port
serversocket.bind((host, port))
# queue up to 5 requests
serversocket.listen(5)
while True:
    # establish a connection
    clientsocket,addr = serversocket.accept()
    print("Got a connection from %s" % str(addr))
    currentTime = time.ctime(time.time()) + "\r\n"
    clientsocket.send(currentTime.encode('ascii'))
    clientsocket.close()
```

```
Got a connection from ('192.168.29.226', 56824)
```

```
# client.py
import socket

# create a socket object
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# get local machine name
host = socket.gethostname()

port = 9999
# connection to hostname on the port.
s.connect((host, port))

# Receive no more than 1024 bytes
tm = s.recv(1024)
s.close()

print("The time got from the server is %s" % tm.decode('ascii'))
```

```
The time got from the server is Thu Sep 29 19:02:03 2022
```

LAB-2

Date: 09/09/2022

HALF DUPLEX CHAT USING TCP/IP

GIVEN REQUIREMENTS:

There are two hosts, Client and Server. Both the Client and the Server exchange message i.e. they send messages or

receive message from the other. There is only a single way communication between them.

TECHNICAL OBJECTIVE:

To implement a half duplex application, where the Client establishes a connection with the Server.

The Client can send and the server well receive messages at the same time.

METHODOLOGY:

Server:

➢ Include the necessary header files.

➢ Create a socket using socket function with family AF_INET, type as SOCK_STREAM.

➢ Initialize server address to 0 using the bzero function.

➢ Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to dynamically assigned port number.

➢ Bind the local host address to socket using the bind function.

➢ Listen on the socket for connection request from the client.

➢ Accept connection request from the Client using accept function.

➢ Fork the process to receive message from the client and print it on the console. ➢ Read message from

the console and send it to the client.

Client:

➢ Include the necessary header files.

➢ Create a socket using socket function with family AF_INET, type as SOCK_STREAM.

➢ Initialize server address to 0 using the bzero function.

➢ Assign the sin_family to AF_INET.

➢ Get the server IP address and the Port number from the console.

➢ Using gethostbyname function assign it to a hostent structure, and assign it to sin_addr of the server

address structure.

➢ Request a connection from the server using the connect function.

➢ Fork the process to receive message from the server and print it on the console. ➢ Read message from

the console and send it to the server.

CODE:

Server Side Script

```
# Supports Python v3.*
from socket import *

server_port = 5000

server_socket = socket(AF_INET,SOCK_STREAM)

server_socket.bind(('',server_port))

server_socket.listen(1)

print ("Welcome: The server is now ready to receive")

connection_socket, address = server_socket.accept()

while True:

  sentence = connection_socket.recv(2048).decode()

  print('>> ',sen#tence)

  message = input(">> ")

  connection_socket.send(message.encode())

  if(message == 'q'):

    connectionSocket.close()
```

 # Client Side Script

```
# Supports Python v3.*
from socket import *
server_name = 'localhost'
server_port = 5000
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 (">> ", message.decode())
  if(sentence == 'q'):
  client_socket.close()
```

SERVER

In [*]:
```
# Server Side Script
# Supports Python v3.*

from socket import *
server_port = 5000
server_socket = socket(AF_INET,SOCK_STREAM)
server_socket.bind(('',server_port))
server_socket.listen(1)
print ("Welcome: The server is now ready to receive")
connection_socket, address = server_socket.accept()
while True:
  sentence = connection_socket.recv(2048).decode()
  print('>> ',sentence)
  message = input(">> ")
  connection_socket.send(message.encode())
  if(message == 'q'):
    connectionSocket.close()
```

```
Welcome: The server is now ready to receive
>>  hello
>> how are you?
>>  Good! Fine morning innit?
>> Absolutely
```

In [ ]:

CLIENT

In [*]:
```
# Client Side Script
# Supports Python v3.*

from socket import *
server_name = 'localhost'
server_port = 5000
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 (">> ", message.decode())
  if(sentence == 'q'):
    client_socket.close()
```

```
>> hello
>>  how are you?
>> Good! Fine morning innit?
>>  Absolutely

>>
```

In [ ]:

INFERENCE:

Thus the FTP client-server communication is established and data is transferred between the client and server machines.

LAB-3

Date: 14/10/2022

IMPLEMENTATION OF FILE TRANSFER PROTOCOL

TECHNICAL OBJECTIVE:

To implement FTP application, where the Client on establishing a connection with the Server sends the name of the file it wishes to access remotely. The Server then sends the contents of the file to the Client, where it is stored.

METHODOLOGY:

Server:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to dynamically assigned port number.
- Bind the local host address to socket using the bind function.
- Listen on the socket for connection request from the client.
- Accept connection request from the Client using accept function.
- Within an infinite loop, receive the file name from the Client.
- Open the file, read the file contents to a buffer and send the buffer to the Client.

Client:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET.
- Get the server IP address and the Port number from the console.
- Using gethostbyname function assign it to a hostent structure, and assign it to sin_addr ofthe server address structure.
- Within an infinite loop, send the name of the file to be viewed to the Server.
- Receive the file contents, store it in a file and print it on the console.

CODING:

SERVER:

```
import socket import threading import os  class Server:     def __init__(self):
self.s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)         self.accept_connections()
def accept_connections(self):
ip = socket.gethostbyname(socket.gethostname())
port = int(input('Enter desired port --> '))
self.s.bind((ip,port))
 self.s.listen(100)
print('Running on IP: '+ip)
print('Running on port: '+str(port))

 while 1:
c, addr = self.s.accept()
print(c)

threading.Thread(target=self.handle_client,args=(c,addr,)).start()
  def handle_client(self,c,addr):
 data = c.recv(1024).decode()
            if not os.path.exists(data):
            c.send("file-doesn't-exist".encode())          else:
            c.send("file-
exists".encode())             print('Sending',data)
if data != '': file = open(data,'rb')
data = file.read(1024)
while data:
c.send(data)
data = file.read(1024)

c.shutdown(socket.SHUT_RDWR)
c.close()
server = Server()
```

CLIENT:

```
import socket
import os  class Client:
def __init__(self):
        self.s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
self.connect_to_server()
     def connect_to_server(self):
        self.target_ip = input('Enter ip --> ')
self.target_port = input('Enter port --> ')
         self.s.connect((self.target_ip,int(self.target_port)))
        self.main()
     def reconnect(self):
self.s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
self.s.connect((self.target_ip,int(self.target_port)))
```

```
def ain(self):
while 1:
          file_name = input('Enter file name on server --> ')
          self.s.send(file_name.encode())
           confirmation = self.s.recv(1024)
if confirmation.decode() == "file-doesn't-exist":
             print("File doesn't exist on server.")
             self.s.shutdown(socket.SHUT_RDWR)
self.s.close()
self.reconnect()
           else:
             write_name = 'from_server '+file_name
             if os.path.exists(write_name):
                    os.remove(write_name)
              with open(write_name,'wb') as file:
,              while 1:
                    data = self.s.recv(1024)

if not data:
break

 file.write(data)

 print(file_name,'successfully downloaded.')

             self.s.shutdown(socket.SHUT_RDWR)
self.s.close()
self.reconnect()
client = Client()
```

SAMPLE OUTPUT:

Server:

(Host Name:Root1)

```
[root@localhost 4ita33]# vi ftps.c
[root@localhost 4ita33]# cc ftps.c [root@localhost 4ita33]#
./a.out

Server is Running…
FILE REACHED
File output : this is my network lab
```

Client:

(Host Name:Root2)

```
[root@localhost 4ita33]# vi ftpc.c
[root@localhost 4ita33]# cc ftpc.c
[root@localhost 4ita33]# ./a.out
Enter the filename:
ita.txt
Sending the file
contentData sent.....
```

INFERENCE:

Thus the FTP client-server communication is established and data is transferred between theclient and server machines.

LAB-4

Date: 14/10/2022

UDP ECHO CLIENT SERVER COMMUNICATION

GIVEN REQUIREMENTS:

There are two hosts, Client and Server. The Client accepts the message from the user and sends it to the Server. The Server receives the message, prints it and echoes the message back to the Client.

TECHNICAL OBJECTIVE:

To implement an UDP Echo Client-Server application , where the Client on establishing a connection with the Server, sends a string to the Server. The Server reads the String, prints it and echoesit back to the Client.

METHODOLOGY:

Server:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port toSERVER_PORT, a macro defined port number.
- Bind the local host address to socket using the bind function.
- Within an infinite loop, receive message from the client using recvfrom function, print it on the console and send (echo) the message back to the client using sendto function.

Client:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET.
- Get the server IP address from the console.
- Using gethostbyname function assign it to a hostent structure, and assign it to sin_addr ofthe server address structure.
- Within an infinite loop, read message from the console and send the message to the serverusing the sendto function.
- Receive the echo message using the recvfrom function and print it on the console.

CODING:

Server: udpserver.py

```
import socket

localIP     = "127.0.0.1"

localPort   = 20001

bufferSize  = 1024

msgFromServer       = "Hello UDP Client"

bytesToSend         = str.encode(msgFromServer)

# 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

    UDPServerSocket.sendto(bytesToSend, address)
```

Client: udpclient.py

```
import socket

msgFromClient       = "Hello UDP Server"
```

```
bytesToSend         = str.encode(msgFromClient)

serverAddressPort   = ("127.0.0.1", 20001)

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)
```

SAMPLE OUTPUT:

Server:

(Host Name:Root1)

```
[root@localhost 4ita33]# vi udpserver.c [root@localhost 4ita33]# cc udpserver.c [root@localhost 4ita33]# ./a.out
Server is Running…

Message is received
Send data to UDP Client: hi

Message is received
Send data to UDP Client: how are u
```

Client:

(Host Name:Root2)

```
[root@localhost 4ita33]# vi udpclient.c
[root@localhost 4ita33]# cc udpclient.c [root@localhost 4ita33]# ./a.out 127.0.0.1Enter input data :
hi
Data sent to UDP Server:hi
Received Data from server:
hi

Enter input data :
how are u
Data sent to UDP Server:how are u
Received Data from server: how are u

Enter input data :
```

INFERENCE:

Thus, the UDP ECHO client server communication is established by sending the message from theclient to the server and server prints it and echoes the message back to the client.

LAB-5

Date: 14/10/2022

DNS SERVER USING UDP

GIVEN REQUIREMENTS:

There are multiple web domains. The IP addresses of all the domains are stored in the DNS server to which a client can make a connection to query IP address of the respective domain from the server. Input of domain name is given by user.

TECHNICAL OBJECTIVE:

Domain Name Server (DNS) is implemented through this program. The web address of any website is given by the Client as the input. The DNS Server looks up for the corresponding domain and returns the IP address as the output.

METHODOLOGY:

1.Include the necessary header files.
2.Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.
3.Declare a dictionary which will hold the DNS records with all the domain names and their IP addresses.
4.Create an object of the socket as s which will bind to localhost at port 1234.
5.Using the created socked the client will send the hostname for which the IP address needs to be looked up by the DNS server.
6.Ping the client and send the IP address that is fetched from the dictionary in the server.
7.Print the size of data sent in the output console.
8.Wait for more connections / requests for more domain to IP lookups.

CODING:

DNS: server.py –

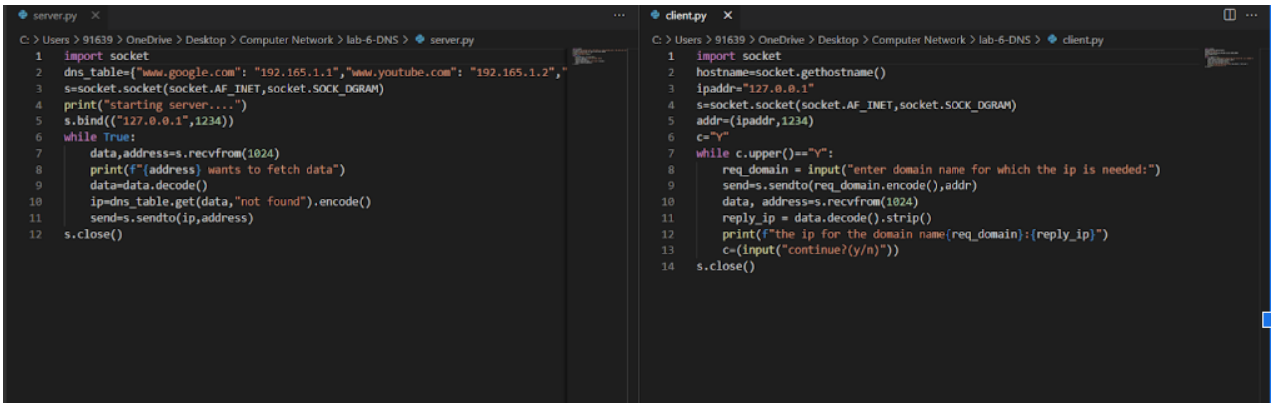
```
import socket
dns_table={"www.google.com": "192.165.1.1","www.youtube.com": "192.165.1.2","www.gmail.com": "192.165.1.3"}
print("starting server....")
while True:
s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
s.bind(("127.0.0.1",1234))
data,address=s.recvfrom(1024)
print(f"{address} wants to fetch data")
data=data.decode()
ip=dns_table.get(data,"not found").encode()
send=s.sendto(ip,address)
s.close()
```

client.py –

```
import socket
```

```
hostname=socket.gethostname()
addr=("127.0.0.1",1234)
c = "Y"
while c.upper()=="Y":
s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
req_domain = input("enter domain name for which the ip is needed:")
send=s.sendto(req_domain.encode(),addr)
data, address=s.recvfrom(1024)
reply_ip = data.decode().strip()
print(f"the ip for the domain name{req_domain}:{reply_ip}")
c=(input("continue?(y/n)"))
s.close()
```

OUTPUT:



```
server.py
C: > Users > 91639 > OneDrive > Desktop > Computer Network > lab-6-DNS > server.py
import socket
dns_table={"www.google.com": "192.165.1.1","www.youtube.com": "192.165.1.2","
s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
print("starting server....")
s.bind(("127.0.0.1",1234))
while True:
    data,address=s.recvfrom(1024)
    print(f"{address} wants to fetch data")
    data=data.decode()
    ip=dns_table.get(data,"not found").encode()
    send=s.sendto(ip,address)
s.close()
```

```
client.py
C: > Users > 91639 > OneDrive > Desktop > Computer Network > lab-6-DNS > client.py
import socket
hostname=socket.gethostname()
ipaddr="127.0.0.1"
s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
addr=(ipaddr,1234)
c="Y"
while c.upper()=="Y":
    req_domain = input("enter domain name for which the ip is needed:")
    send=s.sendto(req_domain.encode(),addr)
    data, address=s.recvfrom(1024)
    reply_ip = data.decode().strip()
    print(f"the ip for the domain name{req_domain}:{reply_ip}")
    c=(input("continue?(y/n)"))
s.close()
```

INFERENCE:

Thus the DNS Server is developed to get the domain name from the remote machine and to send the IP address of the domain as response from server's DNS table.

LAB-6

Date: 03/11/2022

ARP IMPLEMENTATION USING UDP

GIVEN REQUIREMENTS:

There is a single host. The IP address of any Client in the network is given as input and the corresponding hardware address is got as the output.

TECHNICAL OBJECTIVE:

Address Resolution Protocol (ARP) is implemented through this program. The IP address of any Client is given as the input. The ARP cache is looked up for the corresponding hardware address. This is returned as the output. Before compiling that Client is pinged.

METHODOLOGY:

- Include the necessary header files.
- Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.
- Declare structures arpreq (as NULL structure, if required) and sockaddr_in.
- Initialize server address to 0 using the bzero function.
- Assign the sin_family to AF_INET and sin_addr using inet_aton().
- Using the object of arpreq structure assign the name of the Network Device to the data member arp_dev like, arp_dev=”eth0”.
- Ping the required Client.
- Using the ioctl() we get the ARP cache entry for the given IP address.
- The output of the ioctl() function is stored in the sa_data[0] datamember of the arp_ha structure which is in turn a data member of structure arpreq.
- Print the hardware address of the given IP address on the output console.

CODING:

Server.py –

```
import socket
table={
'192.168.1.1':'1E.4A.4A.11',
'192.168.2.1':'1E.4B.4C.21',
'192.168.3.2':'CE.C5.FC.F1',
'1E.4A.4A.11':'192.168.1.1',
'1E.4B.4C.21':'192.168.2.1',
'CE.C5.FC.F1':'192.168.3.2'
}
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.bind(('',1234))
s.listen()
clientsocket,address=s.accept()
print("connection from",address,"Has Established")
ip=clientsocket.recv(1024)
ip=ip.decode("utf-8")
mac=table.get(ip,'no entry for given address')
```

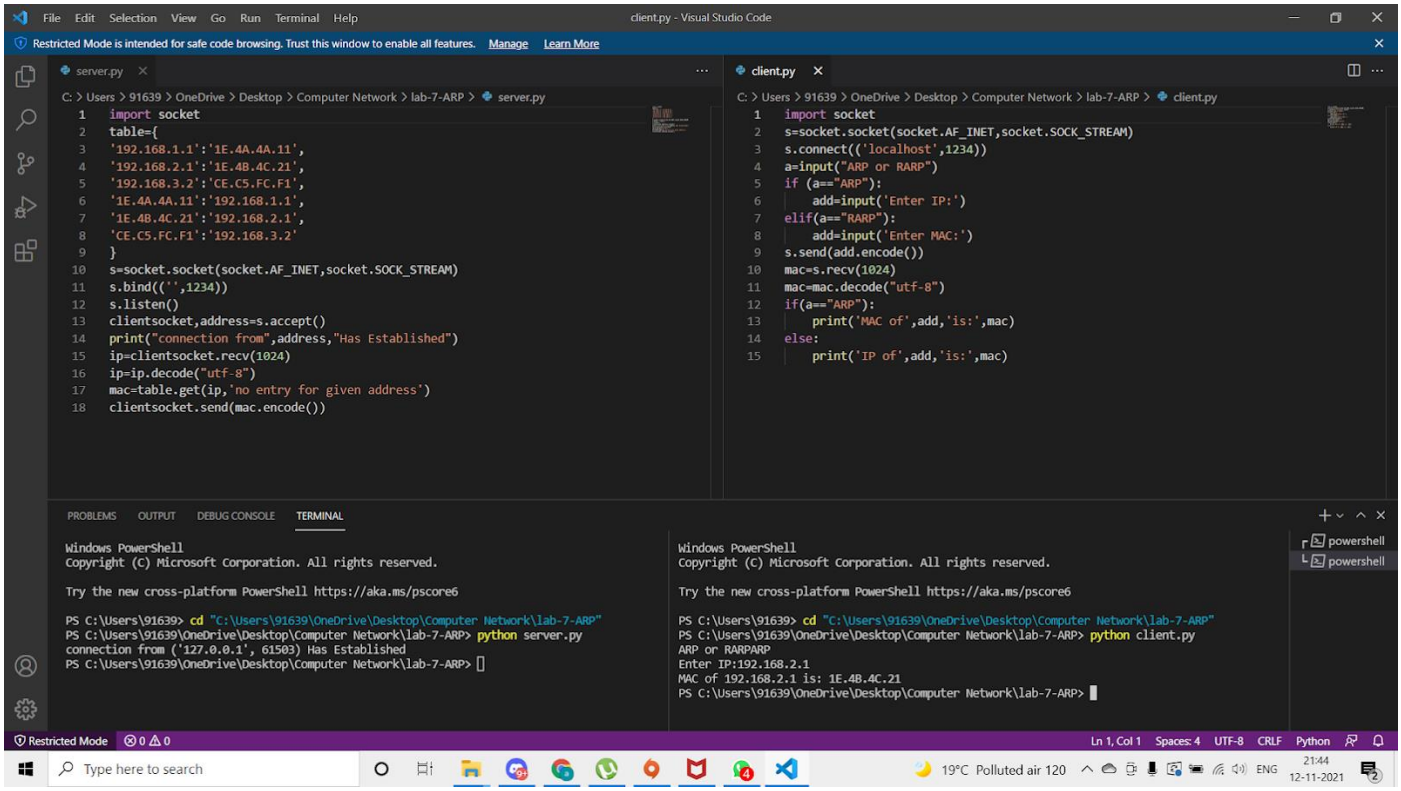
clientsocket.send(mac.encode())

Client.py –

```
import socket
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(('localhost',1234))
a=input("ARP or RARP")
if (a=="ARP"):
   add=input('Enter IP:')
elif(a=="RARP"):
   add=input('Enter MAC:')
s.send(add.encode())
mac=s.recv(1024)
mac=mac.decode("utf-8")
if(a=="ARP"):
   print('MAC of',add,'is:',mac)
else:
   print('IP of',add,'is:',mac)
```

OUTPUT:

ARP-



INFERENCE:

Thus the ARP implementation is developed to gets the MAC address of the remote machine's IP address from ARP cache and prints it.

Experiment No: 7

Date:03/11/2021

Remote Command Execution Using UDP

GIVEN REQUIREMENTS:

There are two hosts, Client and Server. The Client sends a command to the Server, which executes the command and sends the result back to the Client.

TECHNICAL OBJECTIVE:

Remote Command execution is implemented through this program using which client is able to execute commands at the Server. Here, the Client sends the command to the Server for remote execution. The Server executes the command and the send result of the execution back to the Client.

METHODOLOGY:

Server:
1.Include the necessary header files.
2.Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.
3.Initialize server address to 0 using the bzero function.
4.Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to dynamically assigned port number.
5.Bind the local host using the bind() system call.
6.Within an infinite loop, receive the command to be executed from the client.
7.Append text "> temp.txt" to the command.
8.Execute the command using the "system()" system call.
9.Send the result of execution to the Client using a file buffer.

Client:
1.Include the necessary header files.
2.Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.
3.Initialize server address to 0 using the bzero function.
4.Assign the sin_family to AF_INET.
5.Get the server IP address and the Port number from the console.
6.Using gethostbyname() function assign it to a hostent structure, and assign it to sin_addr of the server
7.address structure.
8.Obtain the command to be executed in the server from the user.
9.Send the command to the server.
10.Receive the output from the server and print it on the console.

CODING:

Server: server.py

```
import sys, socket
import os
#from pypsexec.client import
Client
#socket.setdefaulttimeout(150)
host = ''
```

```
port = 50103
BUFSIZE = 1024
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((host, port))
print("Server started on port: %s"%port)
s.listen(1)
print("Now
listening...\n") #conn =
client socket conn, addr
= s.accept() while
True:
    print('New connection from %s:%d' % (addr[0],
addr[1]))     data =
conn.recv(BUFSIZE)     os.system(data.decode())     if not
data:         break     elif data ==
'exit':         conn.send('\0')     else:
        conn.send(data)
    def
quit(connection):
     connection.c
lose()
```

Client: client.py

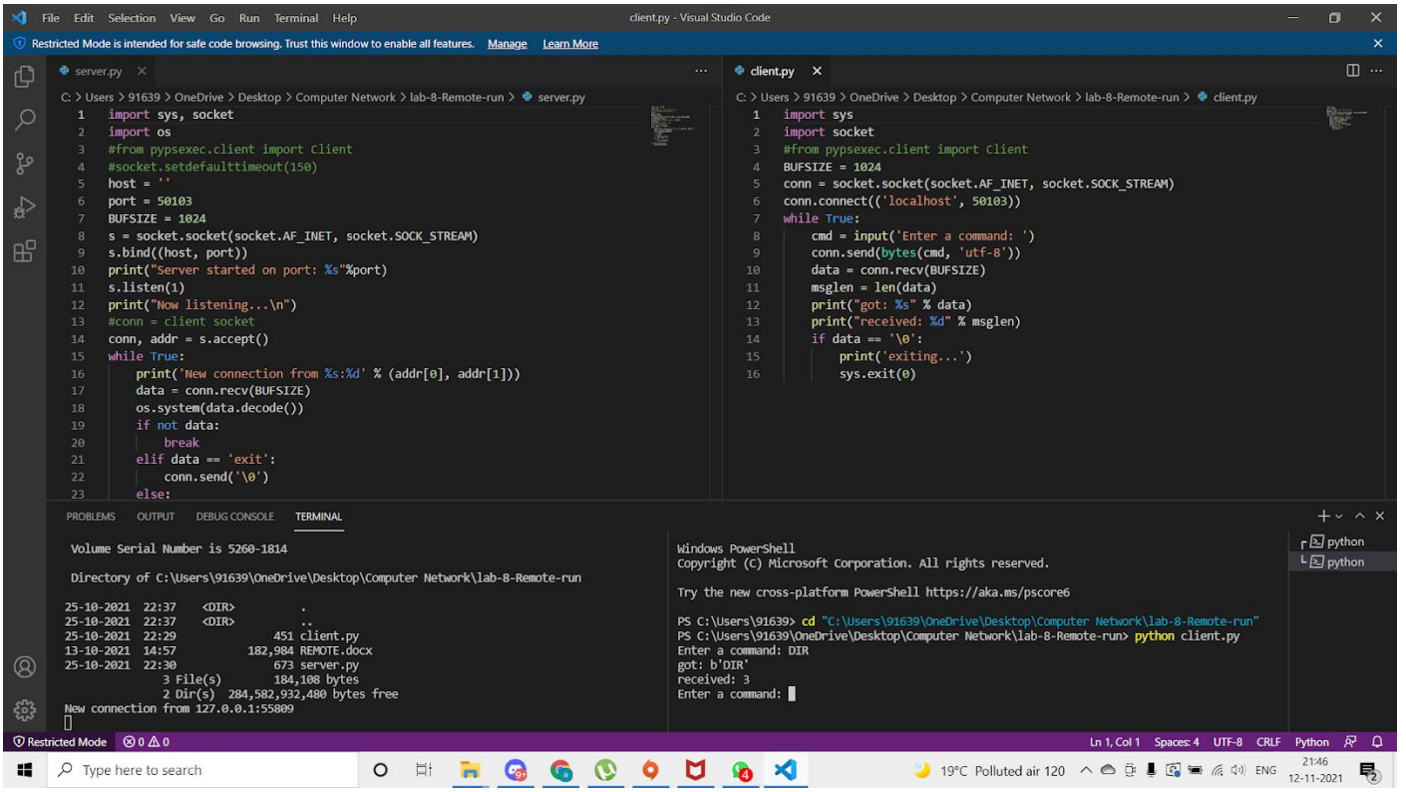
```
Import
sys
import
socket
#from pypsexec.client import
Client BUFSIZE = 1024
conn = socket.socket(socket.AF_INET,
socket.SOCK_STREAM) conn.connect(('localhost',
50103)) while True:
    cmd = input('Enter a
command:
')     conn.send(bytes(cmd,
'utf-8'))     data =
conn.recv(BUFSIZE)     msglen
=
len(data)     print("got: %s"
 %
data)     print("received: %d
" % msglen)     if data ==
'\0':         print('exiting.
..')         sys.exit(0)
```

client - Notepad

File Edit Format View Help

```
import sys
import socket
#from pypsexec.client import Client
BUFSIZE = 1024
conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
conn.connect(('localhost', 50103))
while True:
    cmd = input('Enter a command: ')
    conn.send(bytes(cmd, 'utf-8'))
    data = conn.recv(BUFSIZE)
    msglen = len(data)
    print("got: %s" % data)
    print("received: %d" % msglen)
    if data == '\0':
        print('exiting...')
        sys.exit(0)
```

OUTPUT:



INFERENCE:

Thus the Remote Command Execution between the client and server is implemented.

Experiment No: 8

Date:16-10-2021

FULL DUPLEX CHAT USING TCP/IP

GIVEN REQUIREMENTS:
There are two hosts, Client and Server. Both the Client and the Server exchange message i.e. they send messages to and receive message from the other. There is a two way communication between them.

TECHNICAL OBJECTIVE:
To implement a full duplex application, where the Client establishes a connection with the Server. The Client and Server can send as well as receive messages at the same time. Both the Client and Server exchange messages.

METHODOLOGY:

Server:
1.Include the necessary header files.
2.Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
3.Initialize server address to 0 using the bzero function.
4.Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to dynamically assigned port number.
5.Bind the local host address to socket using the bind function.
6.Listen on the socket for connection request from the client.
7.Accept connection request from the Client using accept function.
8.Fork the process to receive message from the client and print it on the console.
9.Read message from the console and send it to the client.

Client:
1.Include the necessary header files.
2.Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
3.Initialize server address to 0 using the bzero function.
4.Assign the sin_family to AF_INET.
5.Get the server IP address and the Port number from the console.
6.Using gethostbyname function assign it to a hostent structure, and assign it to sin_addr of the server address structure.
7.Request a connection from the server using the connect function.
8.Fork the process to receive message from the server and print it on the console.
9.Read message from the console and send it to the server.

CODING:

Server: server.py

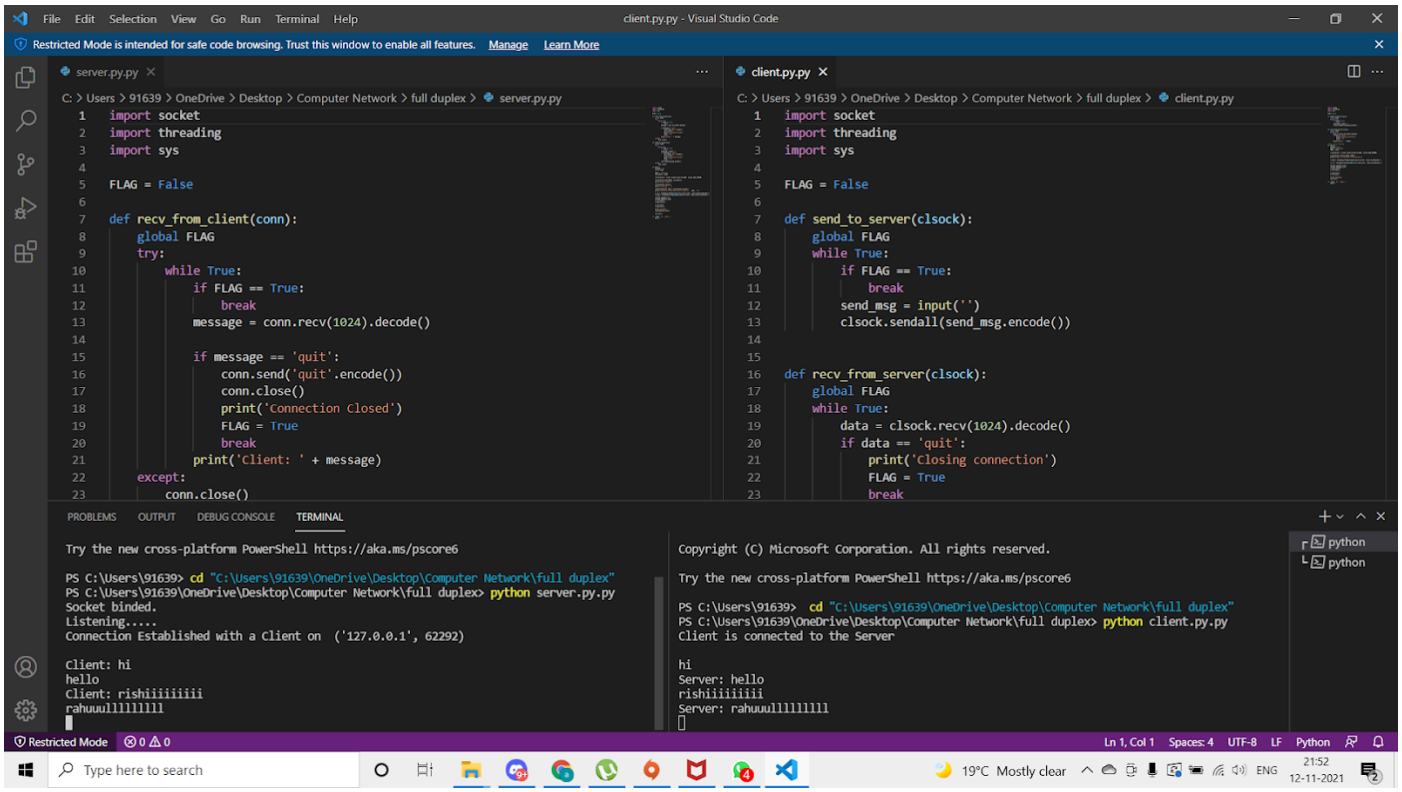
```
import threading
import socket

HOST = '127.0.0.1'
PORT = 9000

print('==== Full Duplex Chat TCP Server ====')
```

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()

    def msg_listener(conn):
        '''Continously Listens for messages'''
        while True:
            message = conn.recv(1024).decode('utf-8')
            print('[Client] {}'.format(message))

    while True:
        conn, addr = s.accept()

        listener = threading.Thread(target=msg_listener, args=(conn,))
        listener.start()

        with conn:
            print(f'Client {addr} connected')

            while True:
                reply = input('')
                conn.sendall(bytes(reply, 'utf-8'))
```

Client: client.py

```
import threading
import socket

HOST = '127.0.0.1'
PORT = 9000

print('==== Full Duplex Chat TCP Client ====')

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))

    def msg_listener(s):
        '''Continously Listens for messages'''
        while True:
            message = s.recv(1024).decode('utf-8')
            print('[Server]: {}'.format(message))

    listener = threading.Thread(target=msg_listener, args=(s,))
    listener.start()

    while True:
        reply = input('')
        s.sendall(bytes(reply, 'utf-8'))
```

SAMPLE OUTPUT:



INFERENCE:

Thus the chat application full duplex communication is established by sending the request from the client to the server, server gets the message and gives response to the client and prints it.