**Name: - D V Ravi Sridhar**

**RA1811003010732**

**CSE D2**

**Experiment 1: -**

**Q1.) Study of necessary header files with respect to socket programming?**

## Ans.) The Socket Module: -Python provide a module named socket for socket programming. To create a socket, you must use the socket.socket() function available in socket module. Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.

**The general syntax** :−

s = socket.socket (socket\_family, socket\_type, protocol)

The following are the description of the parameters −

* **socket\_family:** - Defines family of protocols used as transport mechanism.

The **SOCKET\_FAMILY** element specifies whether Linux/UNIX domain sockets or TCP sockets are to be used on Linux/UNIX systems. The configured default is Linux/UNIX. To use TCP sockets, change the value to TCP.

AF\_INET (IP version 4 or IPv4).

* **socket\_type**: Defines the types of communication between the two end-points.

SOCK\_STREAM (for connection-oriented protocols, e.g., TCP), or

SOCK\_DGRAM (for connectionless protocols e.g. UDP).

* **protocol:**  A protocol is a standard set of rules for transferring data, such as **UDP**/IP and **TCP/IP**. An application program can specify a protocol only if more than one protocol is supported for this particular socket type in this domain. Each socket can have a specific protocol associated with it.We typically leave this field or set this field to zero.

**SOCKET VOCABULARY:**

* **Domain** The family of protocols that is used as the transport mechanism. These values are constants such as AF\_INET, PF\_INET, PF\_UNIX, PF\_X25, and so on.
* **Type** The type of communications between the two endpoints, typically SOCK\_STREAM for connection-oriented protocols and SOCK\_DGRAM for connectionless protocols.
* **Protocol** Typically zero, this may be used to identify a variant of a protocol within a domain and type.
* **Hostname**

The identifier of a network interface −

1. A string, which can be a host name, a dotted-quad address, or an IPV6 address in colon (and possibly dot) notation.
2. A string "<broadcast>", which specifies an INADDR\_BROADCAST address.
3. A zero-length string, which specifies INADDR\_ANY, or
4. An Integer, interpreted as a binary address in host byte order.

* **Port** Each server listens for clients calling on one or more ports. A port may be a Fixnum port number, a string containing a port number, or the name of a service.

**Experiment 2: -**

**Q2.) Study of Basic Functions of Socket Programming.**

Ans.) The following basic functions performed in a socket programming are: -

Server Socket Methods: - Server provides services to many clients which are received services from that centralized server-

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **s.bind() : -** *s.bind((host,port))*  This method binds address (hostname, port number pair) to socket. |
| 2 | **s.listen() : -** *s.listen(5)*  This method sets up and start TCP listener. |
| 3 | **s.accept() :-** *s.accept()*  This passively accept TCP client connection, waiting until connection arrives (blocking). |

Client Socket Methods: - The client in the client-server architecture requests the server and receives services from the server. For this, there is only one method dedicated for clients –

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **s.connect() : -** *s.connect((host,port))*  This method actively initiates TCP server connection. |

General Socket Methods: -

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **s.recv() :-** *s.recv(1024)*  This method receives TCP message |
| 2 | **s.send() :-** *s.send(“ connection accepted “ )*  This method transmits TCP message |
| 3 | **s.recvfrom() :-** *s.rcvfrom(can\_frame\_size)*  This method receives UDP message |
| 4 | **s.sendto() :-** *s.sendto(data, (addr,port))*  This method transmits UDP message |
| 5 | **s.close() :-** *s.close()*  This method closes socket |
| 6 | **socket.gethostname() :-** *s.gethostname()*  Returns the hostname. |

## A Simple Server Example: -

#!/usr/bin/python # This is server.py file

import socket # Import socket module

s = socket.socket() # Create a socket object

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

port = 12345 # Reserve a port for your service.

s.bind((host, port)) # Bind to the port

s.listen(5) # Now wait for client connection.

while True:

c, addr = s.accept() # Establish connection with client.

print 'Got connection from', addr

c.send('Thank you for connecting')

c.close() # Close the connection

## A Simple Client Example: -

#!/usr/bin/python # This is client.py file

import socket # Import socket module

s = socket.socket() # Create a socket object

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

port = 12345 # Reserve a port for your service.

s.connect((host, port))

print s.recv(1024)

s.close() # Close the socket when done

## Output: -

Got connection from ('127.0.0.1', 48437)

Thank you for connecting

**Experiment 3:-**

3.) Simple TCP/IP Client Server Communication

Ans.) Server code: -

import socket

s = socket.socket()

s.bind(('localhost', 9999))

s.listen(1)

print('Server is listening')

c, addr = s.accept()

print('Connected with ', addr)

d = c.recv(1024)

while (True):

data = input("Enter message:")

c.sendall(data.encode())

if(data=="stop"):

print("Stopping connection")

break

s.close()

Client Code: -

import socket

c = socket.socket()

c.connect(('localhost',9999))

print('Client waiting for connection')

c.sendall('Client Connected'.encode())

while(True):

msg = c.recv(1024)

if (msg.decode() == 'stop'):

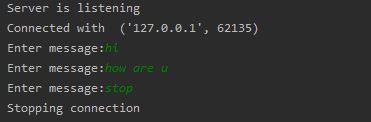
print("Stopping connection")

break

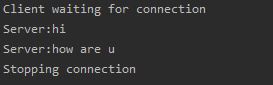
print('Server:'+msg.decode())

c.close()

Server Output: -



Client Output: -



**Experiment 4:-**

## 4.) UDP Echo Client Server Communication

## Ans.) Server code: -

## import socket

## s = socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM)

## s.bind(('localhost',9999))

## print('server is listening')

## while True:

## data =s.recvfrom(1024)

## s.sendto(data[0],data[1])

## s.close()

## Client Code: -

## import socket

## c = socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM)

## c.connect(('localhost',9999))

## print('client waiting for connection')

## while (True):

## msg=input("enter a message:")

## c.sendto(msg.encode(),('localhost',9999))

## msg=c.recvfrom(1024)

## if (msg[0].decode()=='stop'):

## print('Stopping connetion')

## break

## print("Server echoed:"+msg[0].decode())

## c.close()

## Server code: -

## 

## Client Code: -

## 

## Output: -

## Server output: -

## C:\Users\ADMIN\AppData\Local\Programs\Python\Python38-32\python.exe "F:/Python/UDP echo/sever2.py"

## Server is listening

## Client output:-

## C:\Users\ADMIN\AppData\Local\Programs\Python\Python38-32\python.exe "F:/Python/UDP echo/client2.py"

## Client waiting for connection

## Enter message:hello

## Server echoed:hello

## Enter message:this message is being echoed

## Server echoed:this message is being echoed

## Enter message:

Output: -

**Experiment 5:-**

## 6.) Half Duplex Chat Using TCP/IP

## Ans.) Server code: -

## import socket

## s = socket.socket()

## s.connect(('localhost', 1234))

## print('waiting for connection....')

## s.sendall('Connected to client'.encode())

## while (True):

## msg = s.recv(1024)

## if (msg.decode() == 'exit'):

## print('connection ended')

## break

## # else:

## print(msg.decode())

## data = input('enter something..')

## s.sendall(data.encode())

## if (data == 'exit'):

## break

## s.close()

## Client Server: -

## import socket

## s = socket.socket()

## s.bind(('localhost',1234))

## s.listen(5)

## print('server is listining')

## c,add = s.accept()

## print('connected to..' , add)

## d = c.recv(1024)

## ms = 'You are connected'

## c.sendall(ms.encode())

## while (True):

## msg = c.recv(1024)

## if (msg.decode() == 'exit'):

## print('Connection ended')

## break

## # else:

## print (msg.decode())

## data = input('enter something..')

## c.sendall(data.encode())

## if (data == 'exit'):

## break

## s.close()

## Server code: -

## 

## Client code: -

## 

## Output: -

Server output:

C:\Users\ADMIN\AppData\Local\Programs\Python\Python38-32\python.exe "F:/Python/TCP Half Duplex/server.py"

Server is listening

Connected with ('127.0.0.1', 64206)

Enter message:hello

Enter message:ji

Enter message:sup?

Enter message:

Client output:

C:\Users\ADMIN\AppData\Local\Programs\Python\Python38-32\python.exe "F:/Python/TCP Half Duplex/client.py"

Client waiting for connection

Server:hello

Server:ji

Server:sup?

**Concurrent TCP/IP Day - Time Server**

SERVER:

import socket

from datetime import datetime

today = datetime.today()

s = socket.socket()

s.bind(('localhost', 9999))

s.listen(1)

print('Server is listening')

c, addr = s.accept()

print('Connected with ', addr)

d = c.recvfrom(1024)

while (True):

    #data = input("Enter message:")

    c.sendall(str(today).encode())

    command = c.recvfrom(1024)

    if command[0].decode() == "stop":

        break

    elif command[0].decode() == "continue":

        continue

s.close()

CLIENT:

import socket

c = socket.socket()

c.connect(('localhost',9999))

print('Client waiting for connection')

c.sendall('Client Connected'.encode())

while(True):

    msg = c.recvfrom(1024)

    if (msg[0].decode() == ''):

        break

    print('Server:'+msg[0].decode())

    command = input("Enter command to continue or terminate: ")

    c.sendall(command.encode())

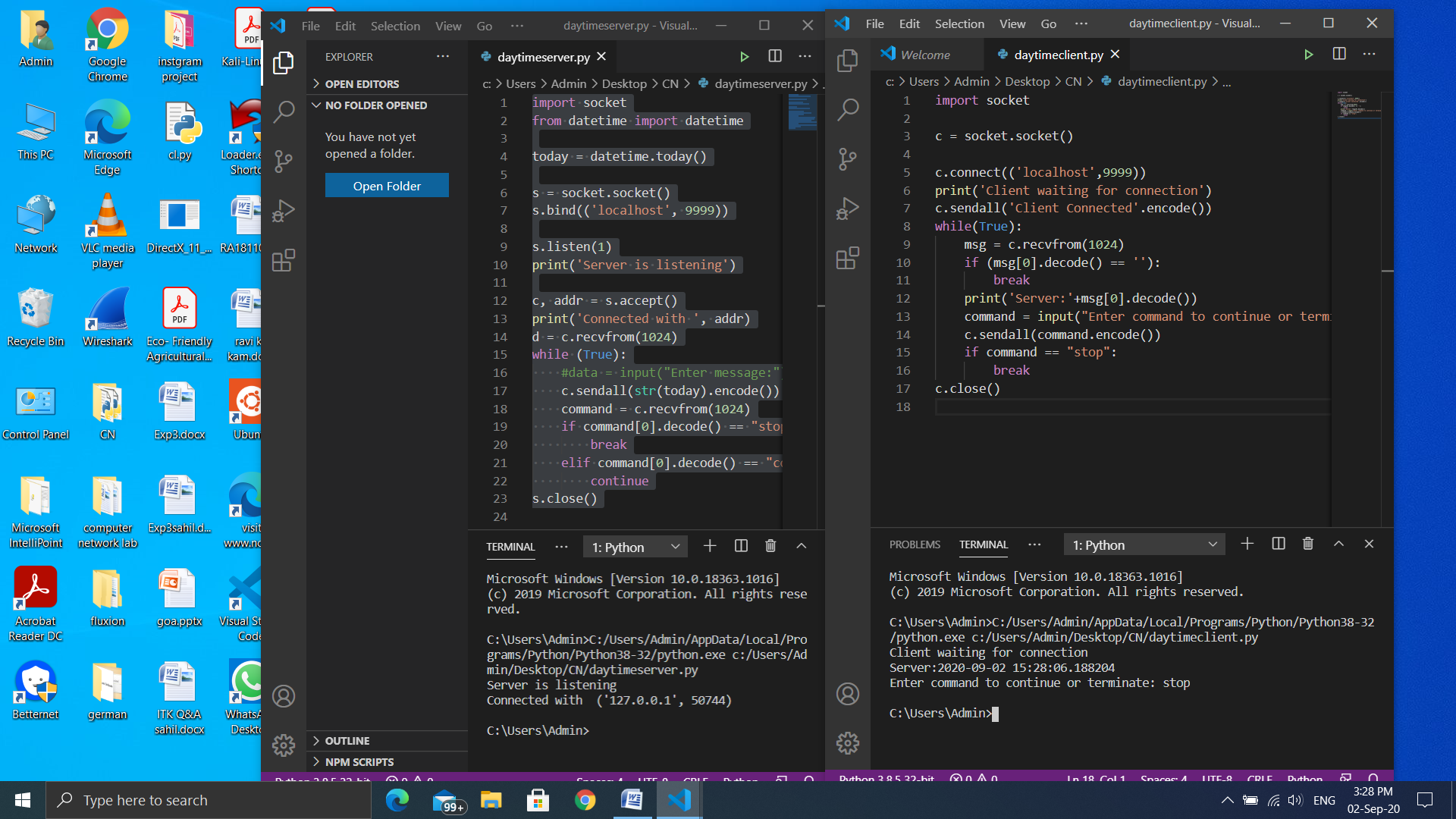
    if command == "stop":

        break

c.close()

OUTPUT:

SERVER: CLIENT:



**Half Duplex TCP C\S Program**

SERVER:

import socket

s = socket.socket()

s.bind(('localhost',8888))

s.listen(5)

print('server is listining')

c,add = s.accept()

print('connected to..' , add)

ms = 'You are connected'

c.sendall(ms.encode())

while (True):

        msg = c.recv(1024)

        if (msg.decode().strip() != 'exit'):

                print ('Client: ',msg.decode())

        else:

                print('Connection ended')

                break

        data = input('enter something..')

        c.sendall(data.encode())

        if (data.strip()=='exit'):

            print('Connection ended')

            break

s.close()

CLIENT:

import socket

s = socket.socket()

s.connect(('localhost',8888))

print('waiting for connection....')

print(s.recv(1024).decode())

while True:

    data = input('enter something..')

    s.sendall(data.encode())

    if (data.strip()=='exit'):

        print('Connection ended')

        break

    msg =  s.recv(1024)

    if (msg.decode().strip() != 'exit'):

         print ('Server: ',msg.decode())

    else:

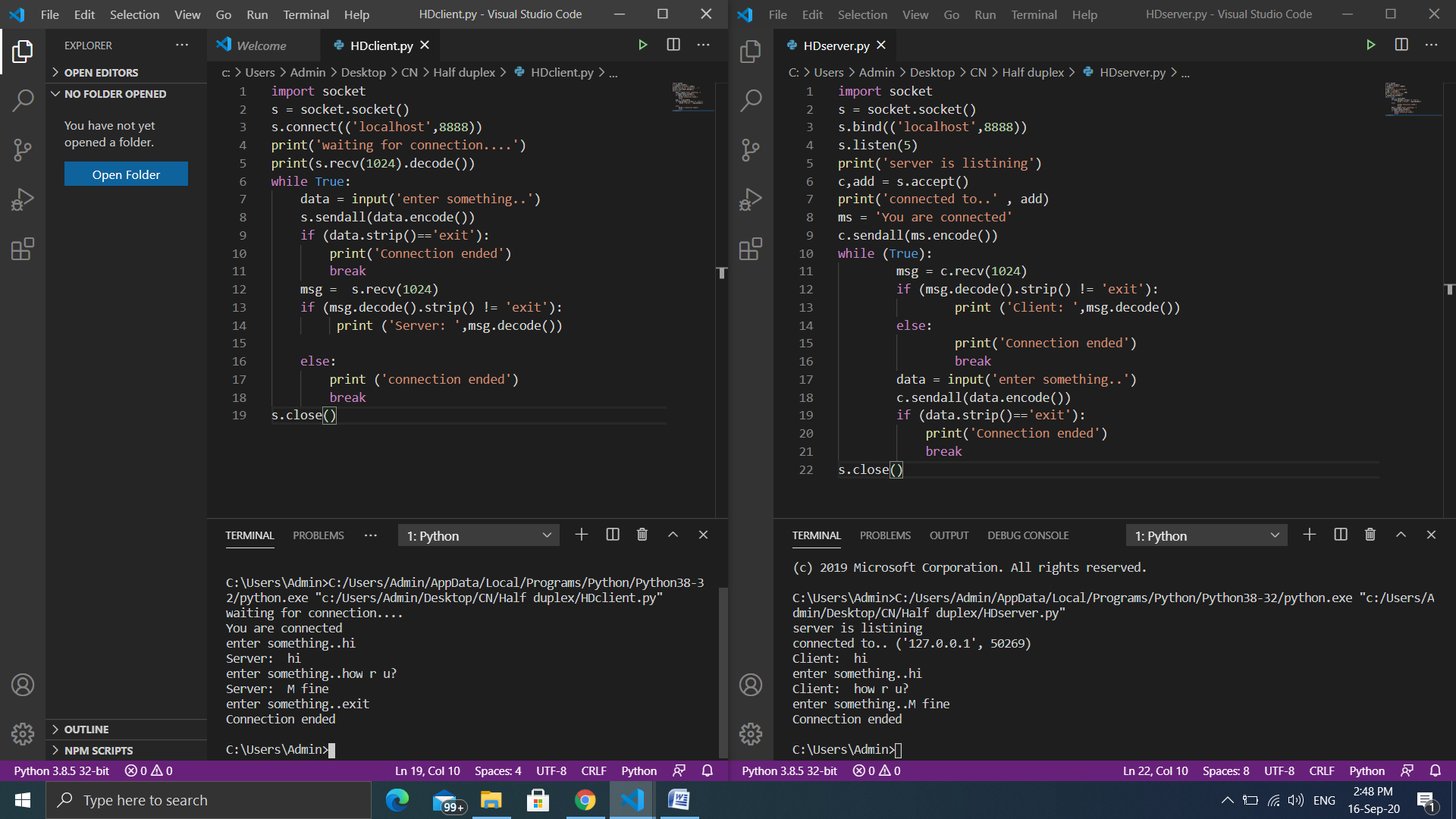
        print ('connection ended')

        break

s.close()

OUTPUT:

CLIENT: SERVER:



**Full Duplex C\S program**

SERVER:

import socket

from time import ctime

import threading

##Server Socket##

sADDR = ("127.0.0.1", 9999)

buff = 1024

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.bind(sADDR)

s.listen(5)

print ("Waiting for a connection...")

c, cADDR = s.accept()

print ("...Connection made with {0}".format(cADDR))

def receive():

    while True:

        rMessage = c.recv(buff)

        if (rMessage.decode() == 'exit'):

            print ("Ending connection")

            exit(0)

        print ('Client : ', rMessage.decode())

def send():

    while True:

        sMessage = input("message: ")

        if sMessage == 'exit':

            c.sendall(sMessage.encode())

            print("Ending the connection")

            exit(0)

        c.sendall(sMessage.encode())

t1 = threading.Thread(target=send, name=3)

t2 = threading.Thread(target=receive, name=4)

t1.start()

t2.start()

CLIENT:

import socket

from time import ctime

import threading

sADDR = ('127.0.0.1', 9999)

buff = 1024

c = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

c.connect(sADDR)

def receive():

    while True:

        rmes = c.recv(buff)

        if rmes.decode() == 'exit':

            print("Ending the connection")

            exit(0)

        print('Server : ' ,rmes.decode())

def send():

    while True:

        mes = input("message: ")

        if mes == 'exit':

            c.sendall(mes.encode())

            print("Ending the connection")

            exit(0)

        c.sendall(mes.encode())

t1 = threading.Thread(target = send, name = 1)

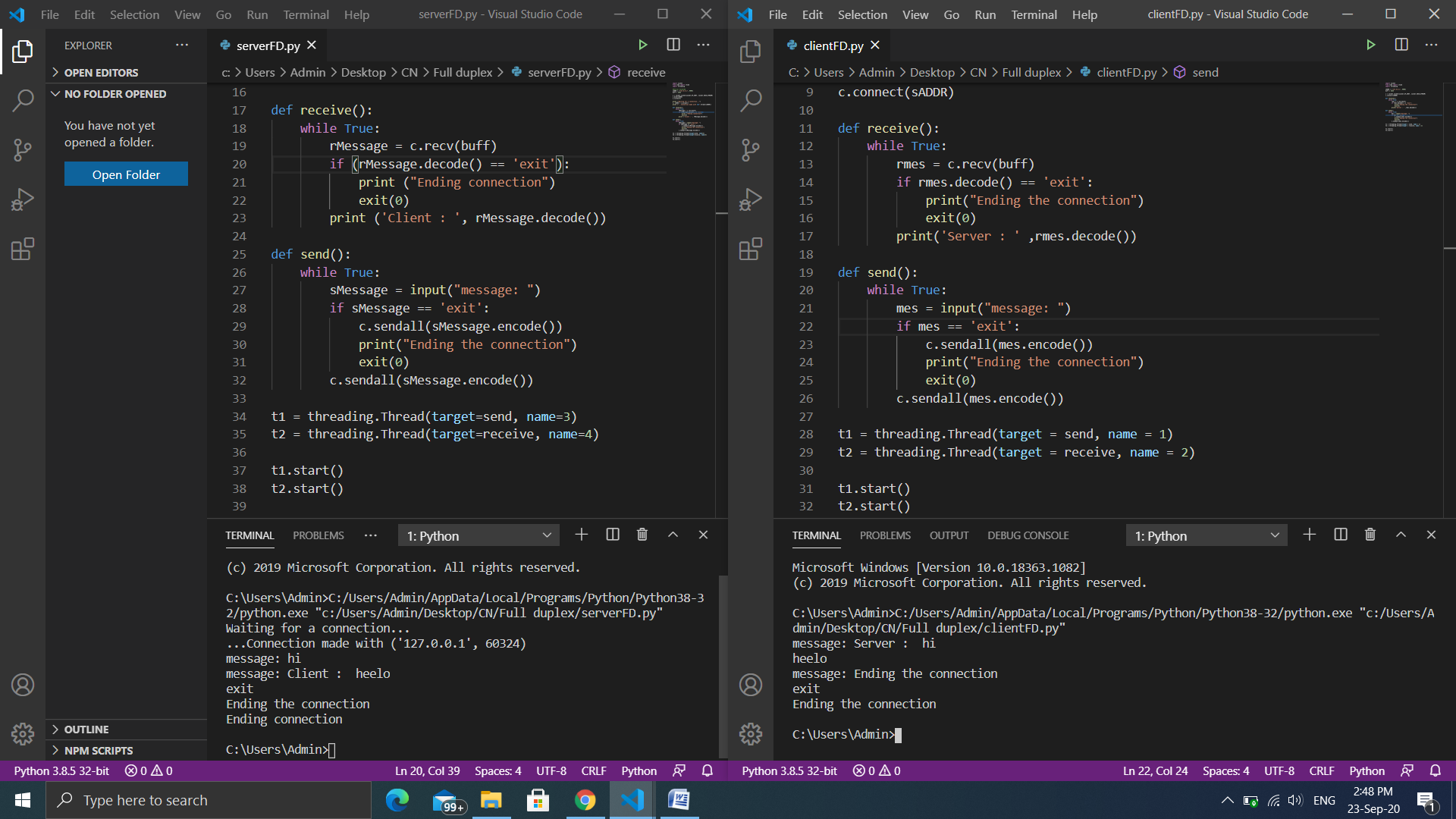
t2 = threading.Thread(target = receive, name = 2)

t1.start()

t2.start()

OUTPUT:

SERVER: CLIENT:



**FTP C\S program**

SERVER:

import socket

import os

s = socket.socket()

s.bind(('localhost', 9999))

s.listen(1)

print('Server is listening')

c, addr = s.accept()

print('Connected with ', addr)

with open("open.txt") as f:

    while True:

        data = f.read(1024)

        if not data:

            print("Stopping Connection")

            break

        c.sendall(data.encode())

f.close()

s.close()

CLIENT:

import socket

c = socket.socket()

c.connect(('localhost', 9999))

print('Client waiting for connection')

f = open("new\_file.txt","w")

while True:

    msg = c.recv(1024).decode()

    #print(msg)

    f.write(msg)

    if msg=='':

        break

    #f.write(msg)

f.close()

f = open("new\_file.txt","r")

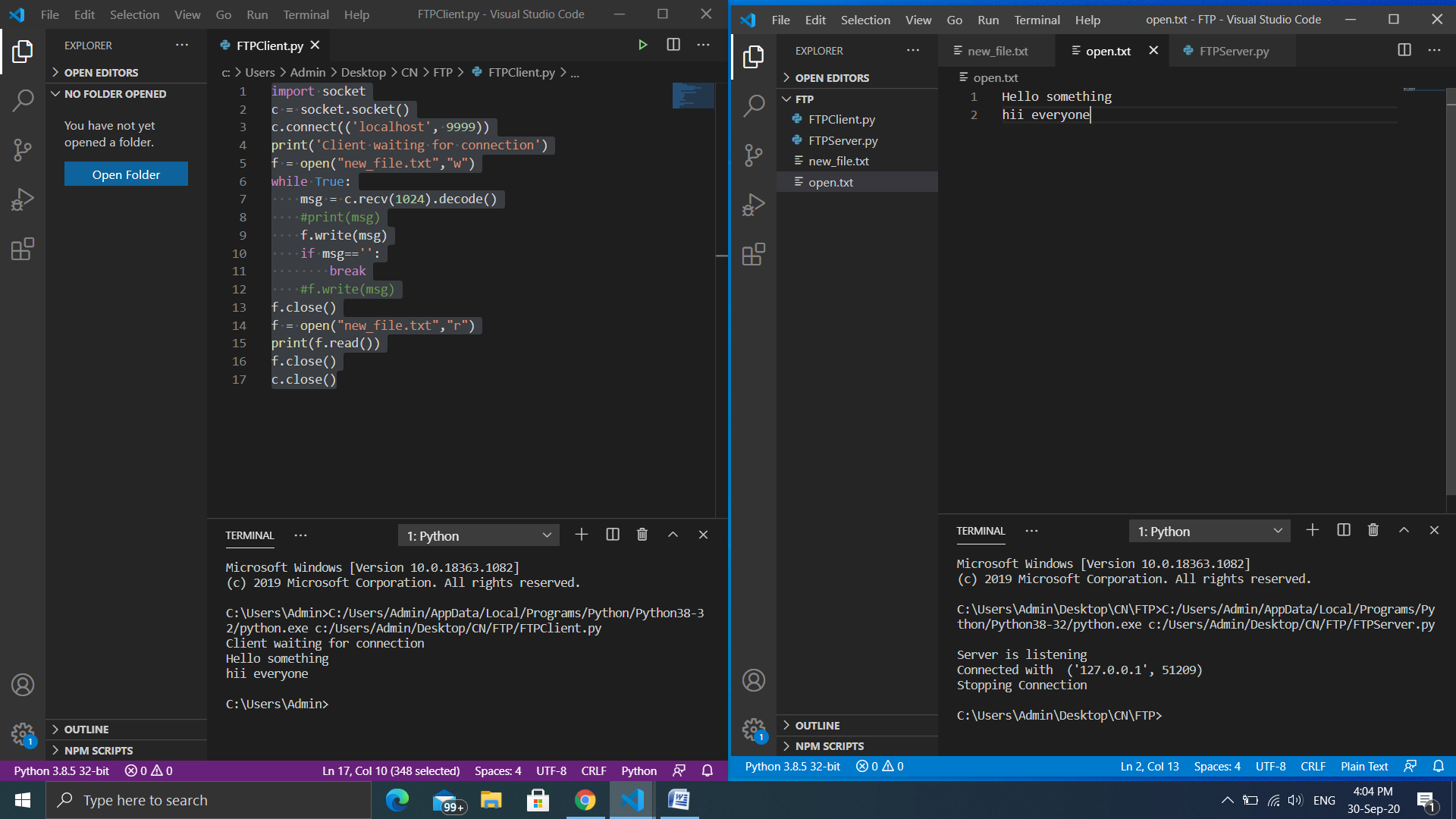
print(f.read())

f.close()

c.close()

OUTPUT:

CLIENT: SERVER:



**Remote Procedure Call**

# Server:

import socket

import os

s=socket.socket(type=socket.SOCK\_DGRAM)

s.bind(('localhost', 9999))

d=s.recvfrom(1024)

os.system('cmd/c "{}"'.format(d[0].decode()))

s.close()

# Client:

import socket

c=socket.socket(type=socket.SOCK\_DGRAM)

c.connect(('localhost', 9999))

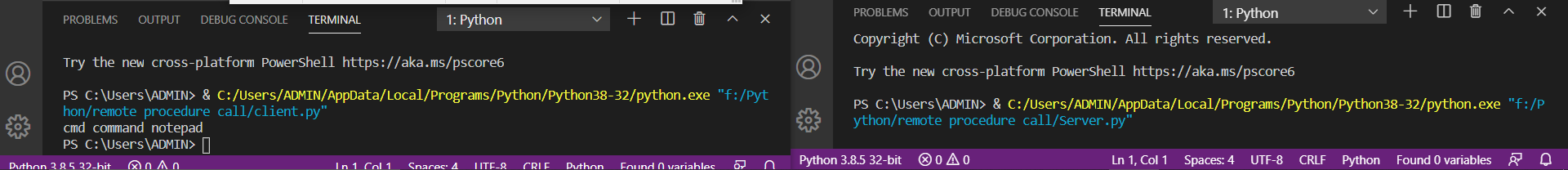
command=input("cmd command ")

c.sendto(command.encode(), ('localhost', 9999))

c.close()

# Output:

Client: Server:



**ARP Implementation using UDP**

SERVER:

#ARP implementation using UDP

import socket

import getmac

s = socket.socket(type=socket.SOCK\_DGRAM)

s.bind(('localhost',9999))

data = s.recvfrom(1024)

ip\_address=data[0].decode()

destination=data[1]

mac = getmac.get\_mac\_address(ip=ip\_address)

if mac is not None:

    s.sendto(mac.encode(),destination)

else:

    s.sendto("Not found".encode(),destination)

s.close()

CLIENT:

import socket

c = socket.socket(type=socket.SOCK\_DGRAM)

c.connect(('localhost',9999))

ip\_address = input("Ip address:")

c.sendto(ip\_address.encode(),('localhost',9999))

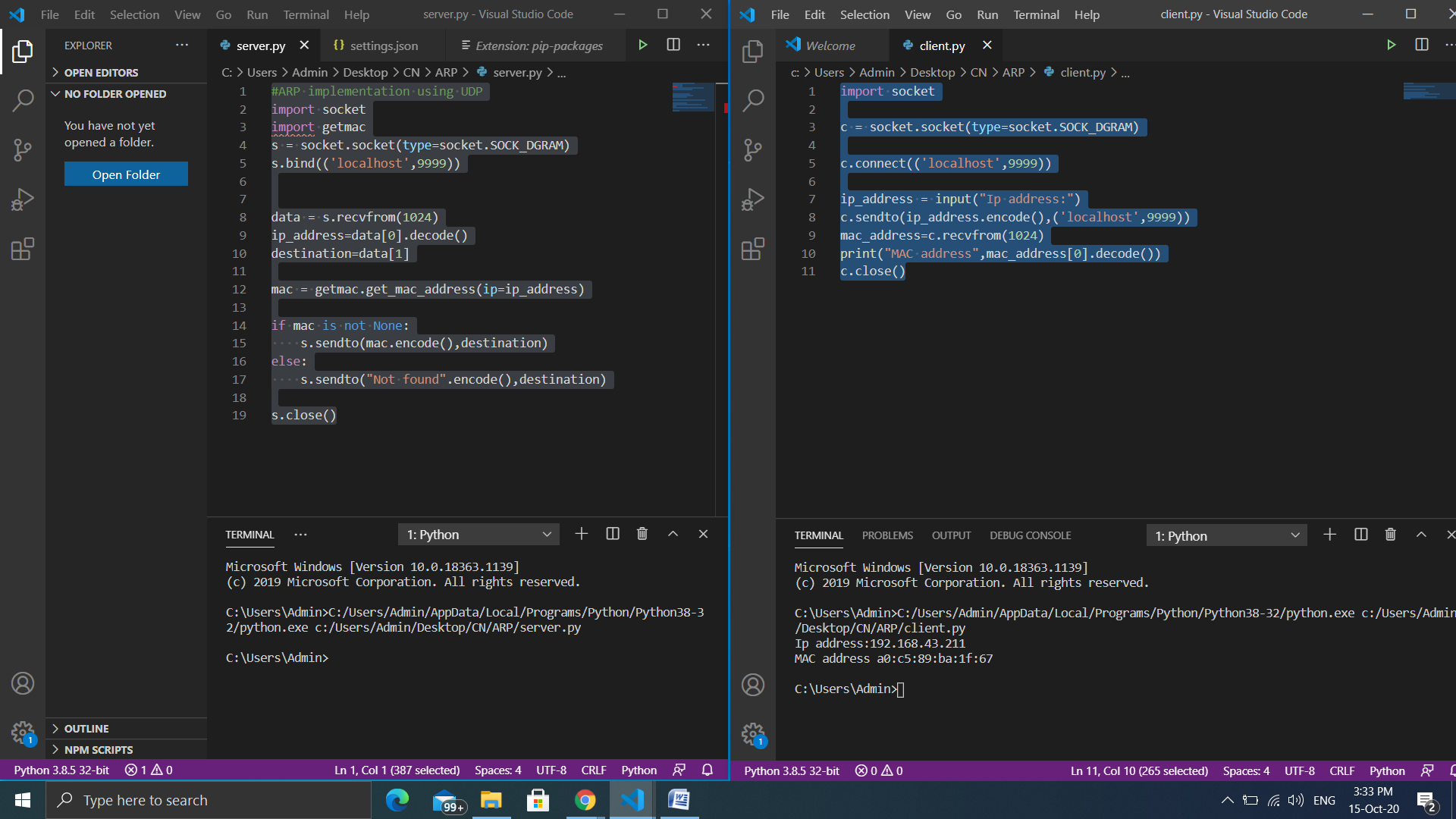
mac\_address=c.recvfrom(1024)

print("MAC address",mac\_address[0].decode())

c.close()

OUTPUT:

SERVER: CLIENT:



**Study of IPV6 Addressing & Subnetting**

* What is IPV6 address?

An IPv6 address is represented as eight groups of four hexadecimal ; digits, each group representing 16 bits (two octets, a group sometimes also called a hextet. The groups are separated by colons (:). An example of an IPv6 address is:

2001:0db8:85a3:0000:0000:8a2e:0370:7334

An IPv6 network uses an address block that is a contiguous group of IPv6 addresses of a size that is a power of two. The leading set of bits of the addresses are identical for all hosts in a given network, and are called the network's address or routing *prefix*.

There are three address models used in IPV6, namely

* Unicast
* Anycast
* Multicast

The IPV6 address can be divided into address blocks og various sizes.The size of a block of addresses is specified by writing a slash (/) followed by a number in decimal whose value is the length of the network prefix in bits, rather than by explicitly specifying which addresses are in the block. For example, an address block with 48 bits in the prefix is indicated by */48*. Such a block contains 2128 − 48 = 280 addresses. The smaller the value of the network prefix, the larger the block: a */21* block is 8 times larger than a */24* block.

* Subnetting

EXAMPLE:

Identify the first four /64 address blocks out of 2003:0DB8:75A3::/48

Original block: 2003:0DB8:75A3::/48

Rewrite as a /64 block: 2003:0DB8:75A3:0000:/48

How many /64 blocks are there in a /48?

(/48 ) / (/64) = 2^(128−48) / 2^( 128−64 )= 2^(80) / 2^(64) = 2^(16)

i.e. ,Find only the first 4 /64 blocks…

2003:0DB8:75A3:0000:/64 can be written as (by manipulating the LSB of your network prefix in BITS)

0000 0000 0000 0000

0000 0000 0000 0001

0000 0000 0000 0010

0000 0000 0000 0011

THE FIRST FOUR ADDRESS BLOCKS ARE AS FOLLOWS:

2003:0DB8:75A3:0001:/64

2003:0DB8:75A3:0002:/64

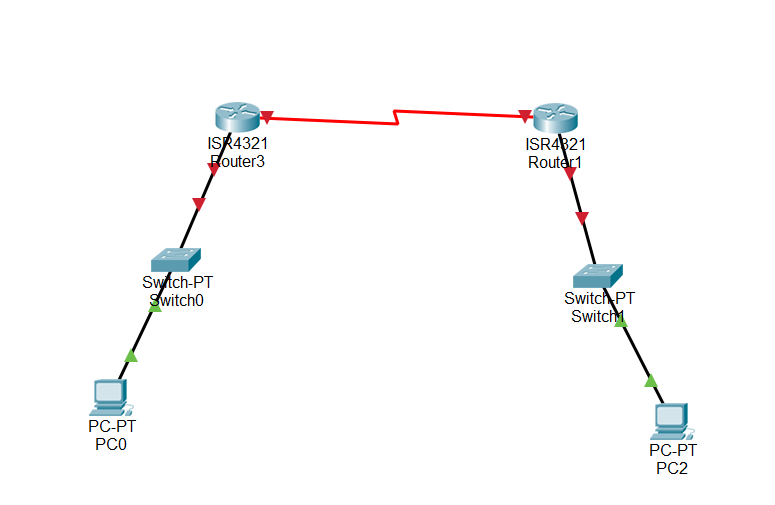
2003:0DB8:75A3:0003:/64

2003:0DB8:75A3:0004:/64

**Communication Using HDLC and PPP**

12.A HDLC

Network Diagram



CLI commands for Router1

R1>**enable**

R1#**configure terminal**

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

R1(config)#**interface Serial0/1**

R1(config-if)#**encapsulation hdlc**

R1(config-if)#**no shutdown**

R1(config-if)#**end**

R1#

%SYS-5-CONFIG\_I: Configured from console by console

R1#

%LINK-3-UPDOWN: Interface Serial0/1, changed state to up

R1#

%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/1,

changed state to up

R1#

CLI commands for Router2

R2>**enable**

R2#**configure terminal**

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

R2(config)#**interface Serial0/1**

R2(config-if)#**encapsulation hdlc**

R2(config-if)#**no shutdown**

R2(config-if)#**end**

R2#

%SYS-5-CONFIG\_I: Configured from console by console

R2#

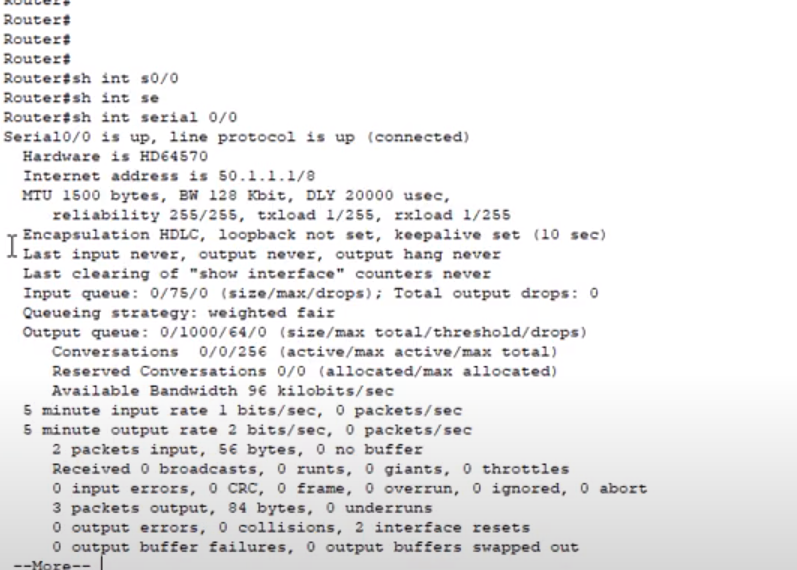
%LINK-3-UPDOWN: Interface Serial0/1, changed state to up

R2#

%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/1,

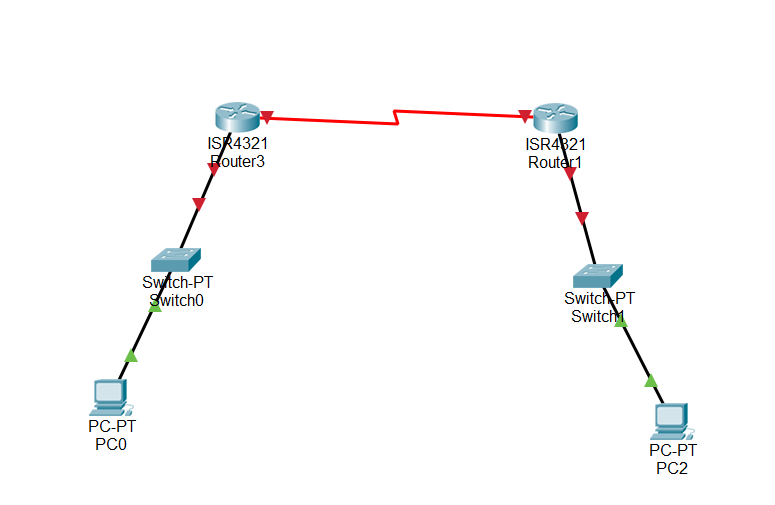
changed state to up

Output:



12.B PPP

Network Diagram



CLI commands for Router1

R1#**configure terminal**

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

R1(config)#**interface Serial0/1**

R1(config-if)#**encapsulation ppp**

R1(config-if)#**end**

R1#

%SYS-5-CONFIG\_I: Configured from console by console

R1#

%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/1, changed

state to down

R1#

CLI commands for Router2

R2#

%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/1, changed

state to down

R2#**configure terminal**

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

R2(config)#**interface Serial0/1**

R2(config-if)#**encapsulation ppp**

R2(config-if)#**end**

R2#

%SYS-5-CONFIG\_I: Configured from console by console

%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/1, changed

state to up

R2#

Output:

