By: Aryan Singh

Peer-to-Peer Multiple Client Chat Application: Socket Programming

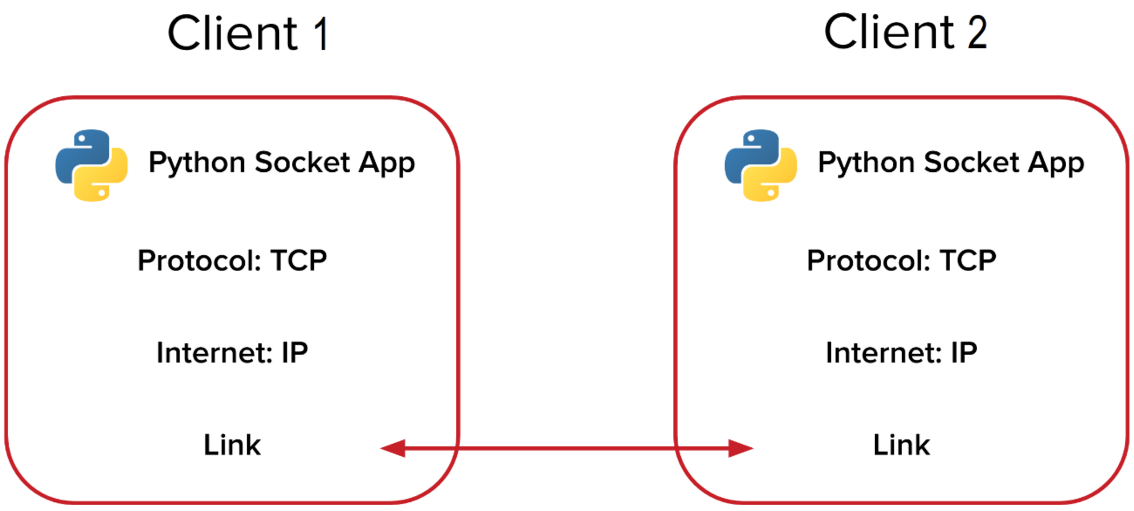
Problem Statement: To develop a **peer-to-peer chat application** and **file transfer** software using Socket Prgramming.

Solution: I am going to design a peer-to-peer chat application with file transfer using socket programming in Python language since Python provides a convenient and consistent API that maps directly to system calls required for IPC.

Protocol: TCP, I'll create a socket object using socket.socket() and specify the socket type as **socket.SOCK_STREAM**. When i do that, the default protocol that's used is the Transmission Control Protocol (TCP).

I would be covering 2 cases which are as follows:

Case I: Only 2 clients



- As you can see in above figure, we would be implementing TCP connection between 2 Python Socket Application.
- This setup is actually using Server – client model where client 1 will act like Server and the other client 2 would be client
- I have not used UDP because that protocol isn't reliable, and data read by the receiver can be out-of-order from what the sender's writes.

- Since in this application both clients can send and receive Msg at the same time so to implement this I need **threading** one thread for sending msg and another thread for receiving msg.

Following is the pseudo code for client 1:

```
PORT = 65432
hostname = socket.gethostname()
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.bind((socket.gethostname(),PORT))
s.listen()
conn, adrr = s.accept()

#function for sending msg
def sending (conn):
   while True:
      inp = input("")
      conn.send(bytes(inp,"utf-8"))
      if inp == "exit":
         break
#function for receiving msg
def recieving(conn):
   while True:
      msg = conn.recv(1024)
      if msg.decode("utf-8") == "exit":
         break
      print(f"{socket.gethostname()}:"+msg.decode("utf-8"))

t1 = threading.Thread(target = sending,args=(conn,))
t2 = threading.Thread(target = recieving,args =(conn,))
t1.start()
t2.start()
t1.join()
t2.join()
```

- Explanation for Above pseudo code
    - First create a socket “s” using socket.socket(socket.AF_INET,socket.SOCK_STREAM) where AF_INET means ip4 and SOCK_STREAM means I would be using TCP protocol.
    - After creating socket “s” I would be binding it to Host Address and a Port No (Random > 1023)
    - Then socket would be listening to connection from Client 2 using s.listen().
    - accept() blocks and waits for an incoming connection When a client connects, it returns a new socket object representing the connection and a tuple holding the address of the client. The tuple will contain (host, port) for IPv4 connections.

- I can send and receive msg using this socket object, this socket object is distinct from the listening socket that the client 1 is using to accept new connections:
- Now for implementing sending and receiving I need to create two separate function for each operation so that each thread can be used to tackle each task.
- Thread “t1” is used to send msg.
- Thread “t2” is used to receive msg from client 2.
- Receiving function:
    - Infinite loop till client 2 sends “exit” msg.
    - Printing msg received by client 1 from client 2 {Maximum limit of msg size is 1024 bytes}.
- Sending function:
    - Infinite loop till client 1 types “exit”.
    - Every time client 1 hits enter key msg is sent over stream {encoded in utf-8}.

Following is the pseudo code for client 2:

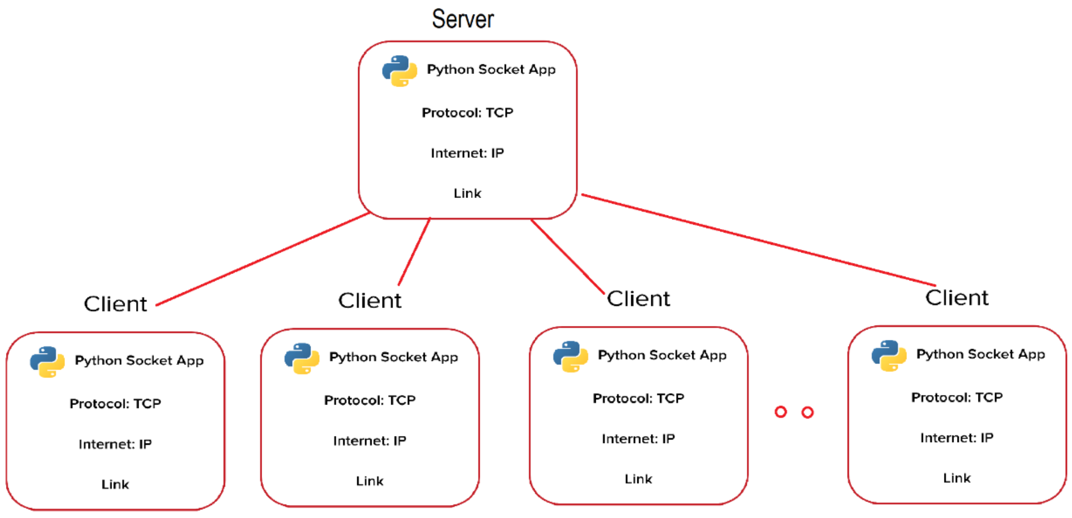
```
PORT = 65432
IP_Address = '192.168.10.1' #write IP Address of Client 1

#function for sending msg
def sending (conn):
   while True:
      inp = input("")
      conn.send(bytes(inp,"utf-8"))
      if inp == "exit":
         break
#function for receiving msg
def recieving(conn):
   while True:
      msg = conn.recv(1024)
      if msg.decode("utf-8") == "exit":
         break
      print(f"{socket.gethostname()}:" + msg.decode("utf-8"))

s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect((IP_Address,PORT))

t1 = threading.Thread(target = sending,args=(s,))
t2 = threading.Thread(target = recieving,args=(s,))

t1.start()
t2.start()
t1.join()
t2.join()
```

- Explanation for Above pseudo code
    - First create a socket “s” using socket.socket(socket.AF_INET,socket.SOCK_STREAM) where AF_INET means ip4 and SOCK_STREAM means I would be using TCP protocol
    - Since client 2 would only be connecting to socket created by client 1 so s.connect() would be having 2 arguments , first argument would be Ip address of client 1 and 2nd would be the port no on which client 1 is listening connection.
    - Similar to client 1 , client 2 would also be having 2 threads for each operation like thread “t1” for sending msg and thread “t2” for receiving msg.
    - Sending and Receiving function are same as in client 1

This was the case for chat application between two clients.

Case II: Multiple clients

- Sending data between 2 or more client devices over the internet is tricky. Due to protections implemented by network security, not all devices connected to the world wide web have a publicly accessible internet protocol (IP) address.
- I would be accomplishing this task using a server in the middle.
    - Client devices using the internet can connect to a server with a public IP address (or a website domain).
    - Then, this broker in the middle can pass messages routed to 1 or many clients.



    -

- Idea is to have 2 threads in Server one thread will be accepting connections and the other would be make communication between peers.
- All clients would be assigned a unique id by which they can inter communicate.
- If client 1 wants to communicate to client 2 it needs to specify client id with msg.
    - For example, msg is “Hello there” and he wants to send this msg to client 2 with id “123” so the way client needs to sends msg in this format “123 Hello there”.
    - This above task would be done by 2nd thread on server side.

    - So, if msg is sent in right format then server would be able to forward it to correct destination.
- There should be a restriction on no of simultaneous connections so that network does not gets overloaded.

Case of sending Files:

- It is similar to sending string converted to utf-8 bytes in place of taking input from user, there we would be sending through file as shown below

```
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))
s.send("Hello server!")
f = open('file.png','rb')
l = f.read(1024)
while (l):
   print 'Sending...'
   s.send(l)
f.close()
print "Done Sending"
s.close
```

- In above example “file.png” is sent over socket just like text or string.

Setup

Case I Only 2 Clients

#Chat History feature is also built.
#Note Write ipaddress of peer 1 in peer2.py file line 4 manually

1> Run Peer1.py on one device
2> Run Peer2.py
#Note Peer1 should be always runned first
3> History will be already loaded on screen of both peers
4>chat
5>write "exit" to stop sending text from your side

#Note if both the peer wrote "exit" then the connection will be terminated

6> again to chat repeat steps from 1 to 5

Case II Multiple Clients

1> Run Server.py file on your machine.
2> Copy the ipAddress showing on the screen to client's file line 4
3> Run client#.py file from 0 to 2 on either same or different machines {#note The order in which clients are connected will be same in their id's ie 0 to 2}
4> Currently you need to connect 3 clients for starting chatting .
5> Once 3 clients are connected Server will show the notification "Receiving Task started".
6> Start chatting by following messaging convention as shown on client's screen.
7> Send "exit" to stop sending msg but you will still be receiving msg.

Thank You