Real Time Chat Application using Socket Programming

TEAM SOCKET-

Ramakrishnan R - 21BLC1013

Kowshik S B - 21BLC1067

Rohith S - 21BLC1456

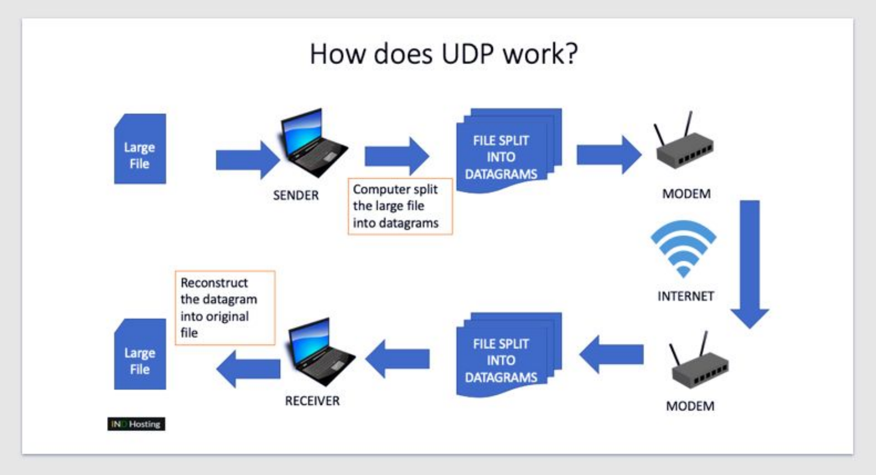
PROJECT DESCRIPTION

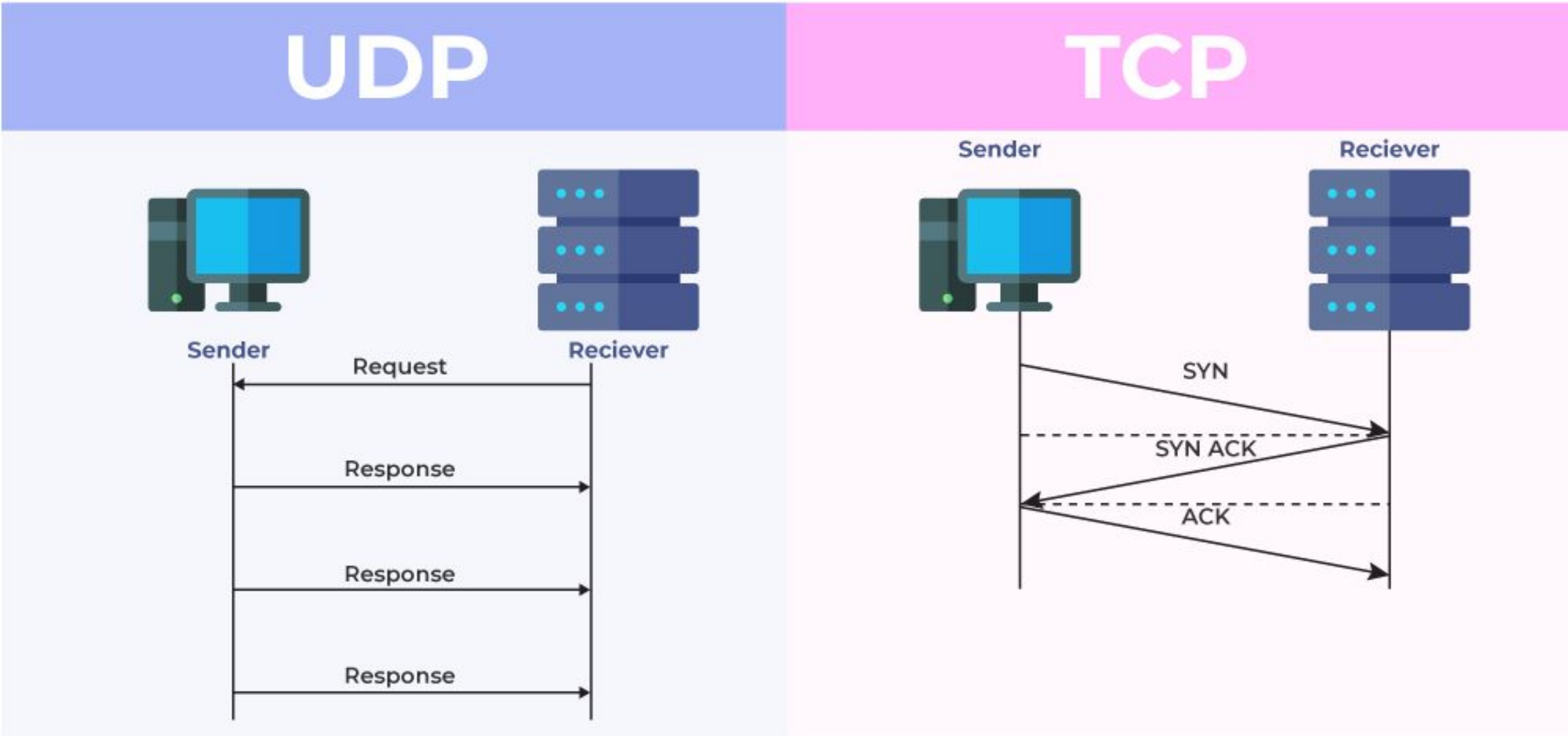
Implementation of Real Time Chat application using Python under Socket programming methodology which supports File transfer and various features.

REVIEW 2

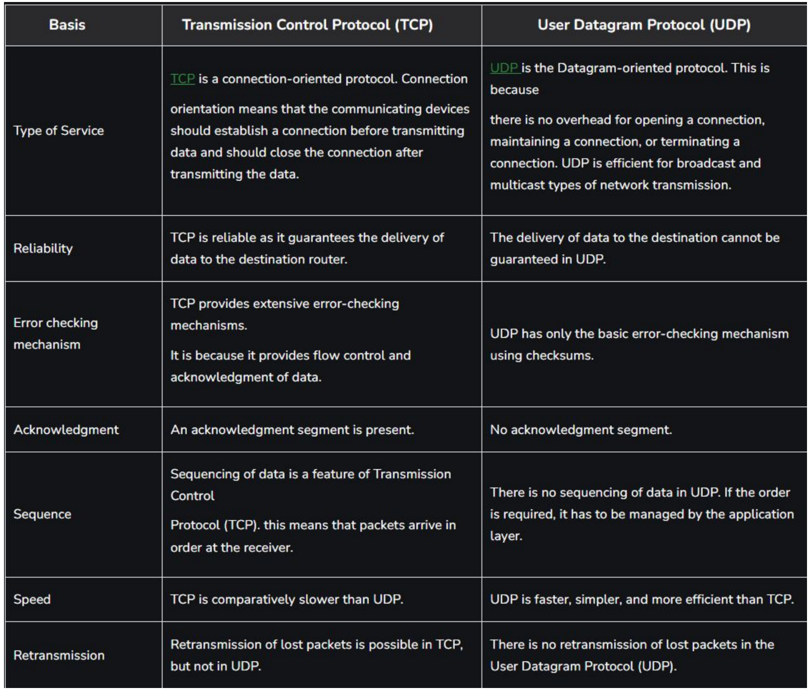
THIS TIME WE HAVE USED UDP INSTEAD OF TCP

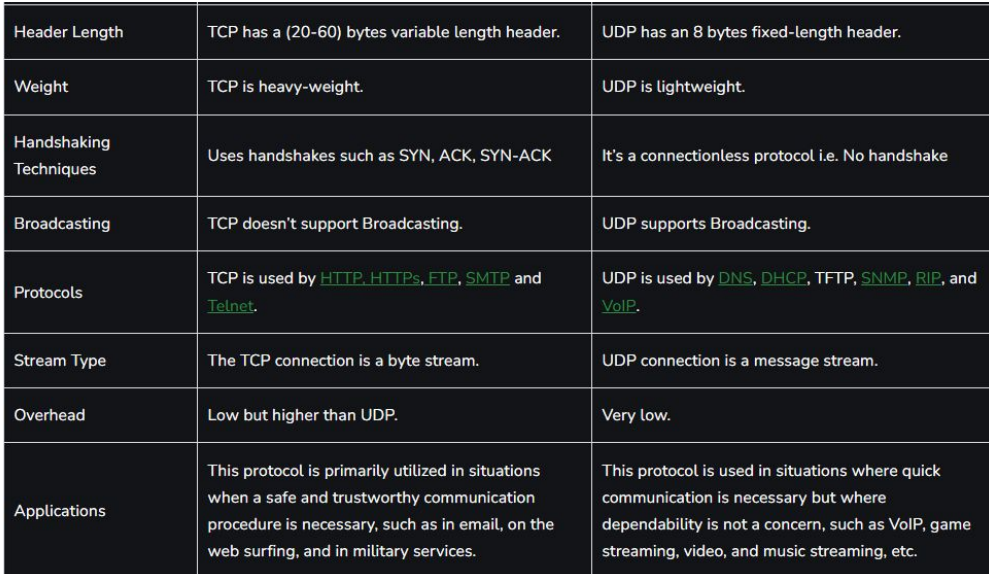
User Datagram Protocol (UDP) is a communications protocol that is primarily used to establish low-latency and loss-tolerating connections between applications on the internet.


How does UDP work?
Large File
SENDER
Computer split the large file into datagrams
FILE SPLIT INTO DATAGRAMS
MODEM
INTERNET
MODEM
FILE SPLIT INTO DATAGRAMS
RECEIVER
Reconstruct the datagram into original file
Large File


UDP
Sender
Reciever
Request
Response
Response
Response
TCP
Sender
Reciever
SYN
SYN ACK
ACK

Differences BETWEEN UDP AND TCP





Basis	Transmission Control Protocol (TCP)	User Datagram Protocol (UDP)
Type of Service	TCP is a connection-oriented protocol. Connection orientation means that the communicating devices should establish a connection before transmitting data and should close the connection after transmitting the data.	UDP is the Datagram-oriented protocol. This is because there is no overhead for opening a connection, maintaining a connection, or terminating a connection. UDP is efficient for broadcast and multicast types of network transmission.
Reliability	TCP is reliable as it guarantees the delivery of data to the destination router.	The delivery of data to the destination cannot be guaranteed in UDP.
Error checking mechanism	TCP provides extensive error-checking mechanisms. It is because it provides flow control and acknowledgment of data.	UDP has only the basic error-checking mechanism using checksums.
Acknowledgment	An acknowledgment segment is present.	No acknowledgment segment.
Sequence	Sequencing of data is a feature of Transmission Control Protocol (TCP). this means that packets arrive in order at the receiver.	There is no sequencing of data in UDP. If the order is required, it has to be managed by the application layer.
Speed	TCP is comparatively slower than UDP.	UDP is faster, simpler, and more efficient than TCP.
Retransmission	Retransmission of lost packets is possible in TCP, but not in UDP.	There is no retransmission of lost packets in the User Datagram Protocol (UDP).
Header Length	TCP has a (20-60) bytes variable length header.	UDP has an 8 bytes fixed-length header.
Weight	TCP is heavy-weight.	UDP is lightweight.
Handshaking Techniques	Uses handshakes such as SYN, ACK, SYN-ACK	It's a connectionless protocol i.e. No handshake
Broadcasting	TCP doesn't support Broadcasting.	UDP supports Broadcasting.
Protocols	TCP is used by HTTP, HTTPs, FTP, SMTP and Telnet.	UDP is used by DNS, DHCP, TFTP, SNMP, RIP, and VoIP.
Stream Type	The TCP connection is a byte stream.	UDP connection is a message stream.
Overhead	Low but higher than UDP.	Very low.
Applications	This protocol is primarily utilized in situations when a safe and trustworthy communication procedure is necessary, such as in email, on the web surfing, and in military services.	This protocol is used in situations where quick communication is necessary but where dependability is not a concern, such as VoIP, game streaming, video, and music streaming, etc.

UDP BASED CHAT ROOM

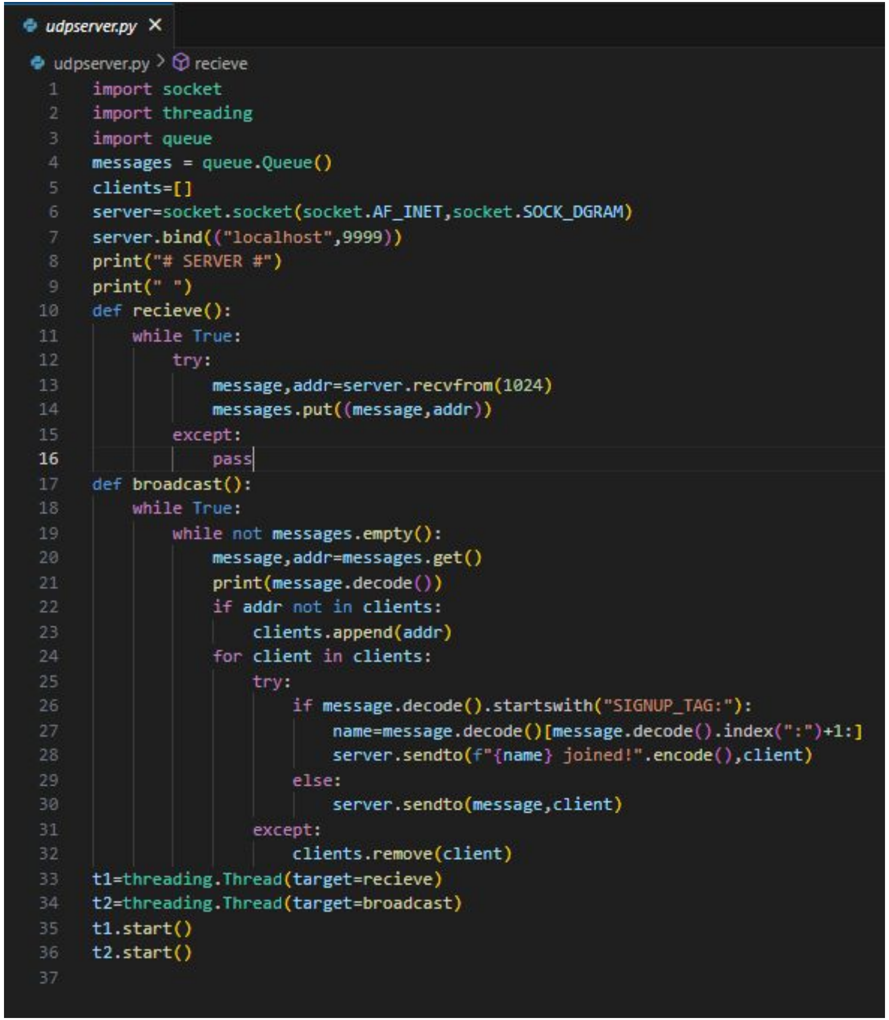
HAS THREE MAJOR CODE FILES

1. server.py
2. client1.py
3. client2.py

Common functions used:-

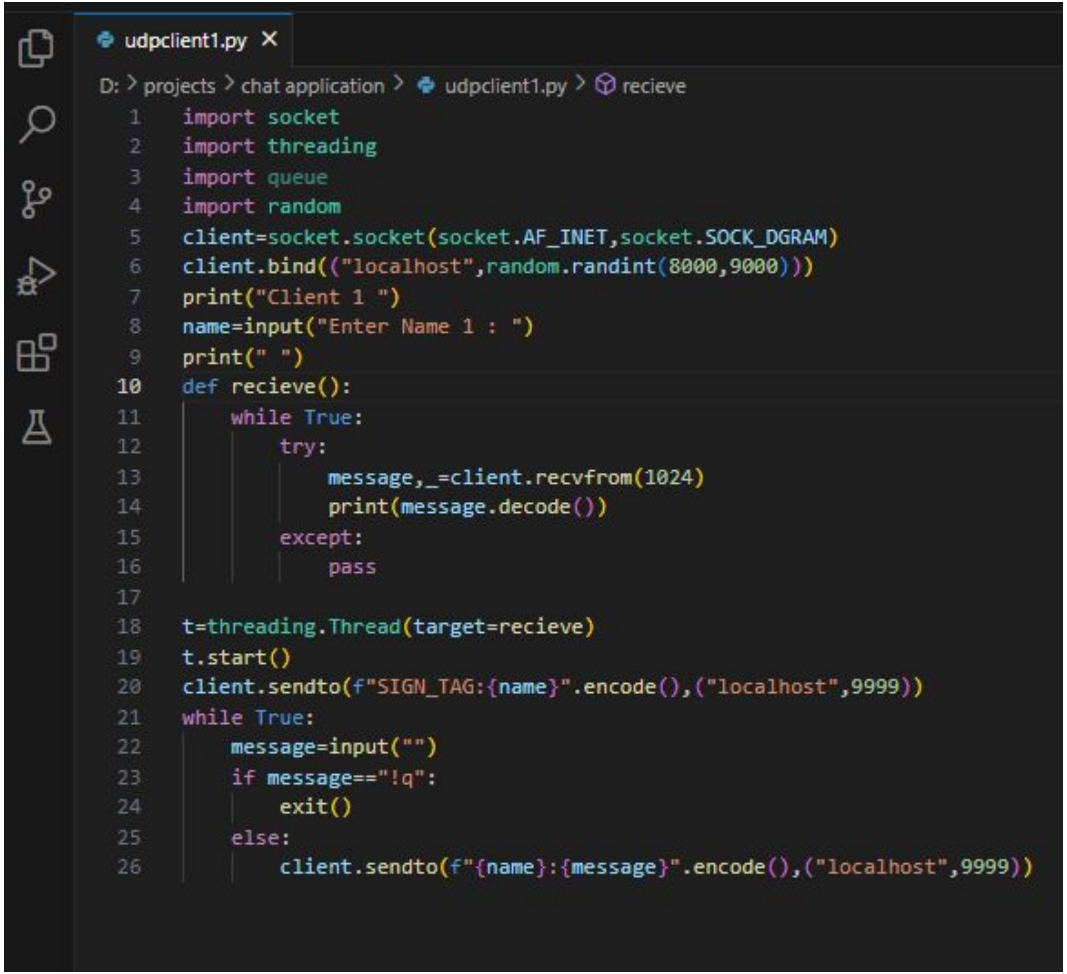
- socket(): This method is used to create the socket and takes two arguments first is a family or domain like AF_INET (IPv4) or INET6 (IPv6) and the second defines the type of sockets like SOCK_STREAM (TCP) or SOCK_DGRAM (UDP).
- bind(): This method is used to bind your socket with a specific host and port which will be passed as an argument to this function and that means your socket will be sitting at a specific location where the client socket can send its data.
- recvfrom(): This method can be used with a UDP server to receive data from a UDP client or it can be used with a UDP client to receive data from a UDP server. It accepts a positional parameter called bufsize which is the number of bytes to be read from the UDP socket. It returns a byte object read from a UDP socket and the address of the client socket as a tuple.
- sendto(): It is a method of Python's socket class that is used to send datagrams to a UDP socket. The communication could be from either side. It could be from client to server or from the server to a client. The data to be sent must be in bytes format. If the data is in string format, the str. encode() method can be used to convert the strings to bytes. We must also pass a tuple consisting of IP address and port number.

SERVER.PY



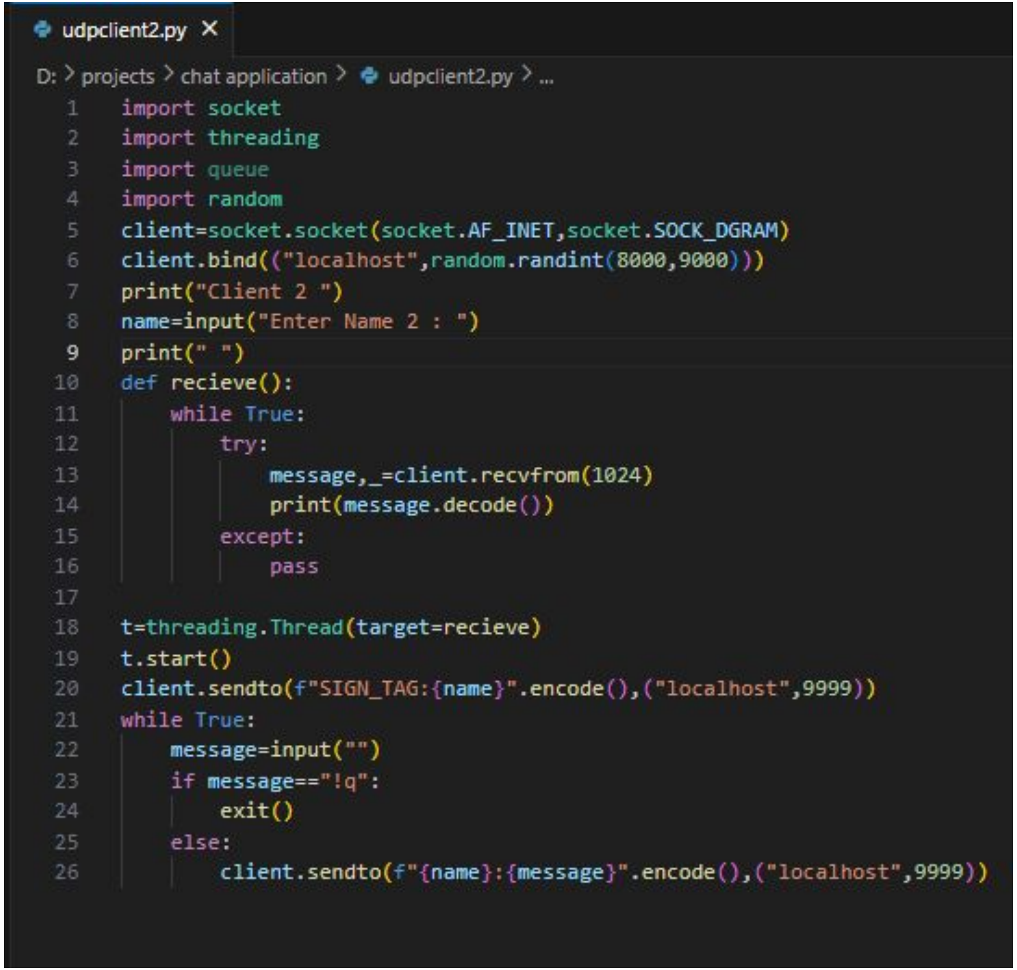
```python
import socket
import threading
import queue
messages = queue.Queue()
clients=[]
server=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
server.bind(("localhost",9999))
print("# SERVER #")
print(" ")
def recieve():
    while True:
        try:
            message,addr=server.recvfrom(1024)
            messages.put((message,addr))
        except:
            pass
def broadcast():
    while True:
        while not messages.empty():
            message,addr=messages.get()
            print(message.decode())
            if addr not in clients:
                clients.append(addr)
            for client in clients:
                try:
                    if message.decode().startswith("SIGNUP_TAG:"):
                        name=message.decode()[message.decode().index(":")+1:]
                        server.sendto(f"{name} joined!".encode(),client)
                    else:
                        server.sendto(message,client)
                except:
                    clients.remove(client)
t1=threading.Thread(target=recieve)
t2=threading.Thread(target=broadcast)
t1.start()
t2.start()

```

CLIENT1.py



```python
import socket
import threading
import queue
import random
client=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
client.bind(("localhost",random.randint(8000,9000)))
print("Client 1 ")
name=input("Enter Name 1 : ")
print(" ")
def recieve():
    while True:
        try:
            message,_=client.recvfrom(1024)
            print(message.decode())
        except:
            pass

t=threading.Thread(target=recieve)
t.start()
client.sendto(f"SIGN_TAG:{name}".encode(),("localhost",9999))
while True:
    message=input("")
    if message=="!q":
        exit()
    else:
        client.sendto(f"{name}:{message}".encode(),("localhost",9999))
```

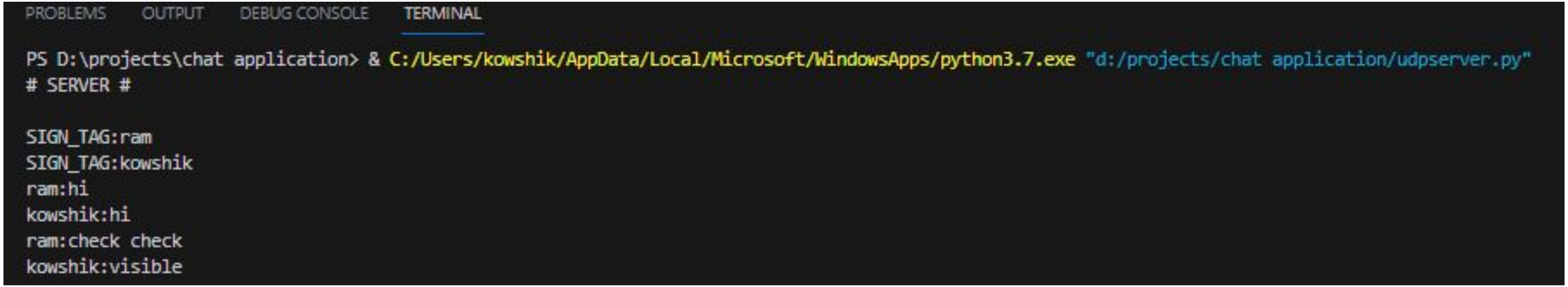
CLIENT2.py



```python
import socket
import threading
import queue
import random
client=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
client.bind(("localhost",random.randint(8000,9000)))
print("Client 2 ")
name=input("Enter Name 2 : ")
print(" ")
def recieve():
    while True:
        try:
            message,_=client.recvfrom(1024)
            print(message.decode())
        except:
            pass

t=threading.Thread(target=recieve)
t.start()
client.sendto(f"SIGN_TAG:{name}".encode(),("localhost",9999))
while True:
    message=input("")
    if message=="!q":
        exit()
    else:
        client.sendto(f"{name}:{message}".encode(),("localhost",9999))
```

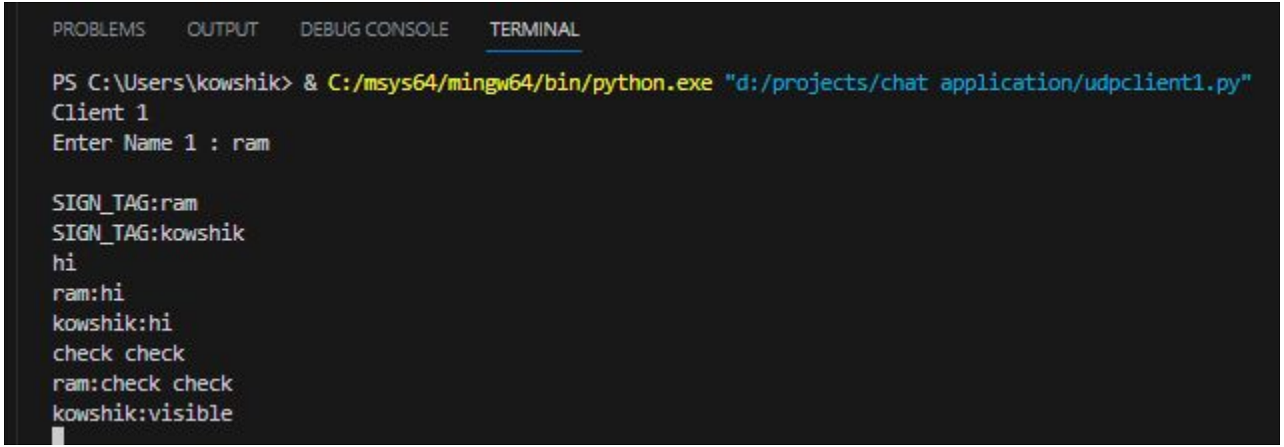
OUTPUTS

Server.py output:



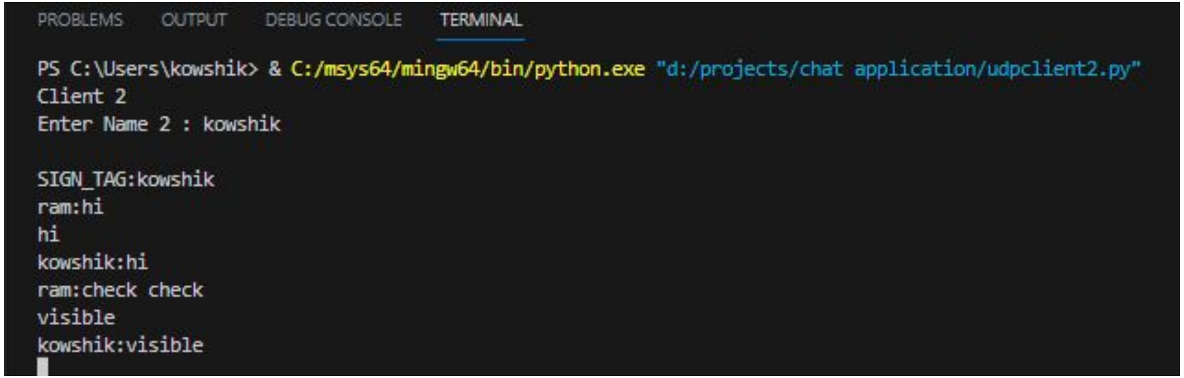
```
PS D:\projects\chat application> & C:/Users/kowshik/AppData/Local/Microsoft/WindowsApps/python3.7.exe "d:/projects/chat application/udpserver.py"
# SERVER #

SIGN_TAG:ram
SIGN_TAG:kowshik
ram:hi
kowshik:hi
ram:check check
kowshik:visible
```

Client1.py output



```
PS C:\Users\kowshik> & C:/msys64/mingw64/bin/python.exe "d:/projects/chat application/udpclient1.py"
Client 1
Enter Name 1 : ram

SIGN_TAG:ram
SIGN_TAG:kowshik
hi
ram:hi
kowshik:hi
check check
ram:check check
kowshik:visible
```

Client2.py output



```
PS C:\Users\kowshik> & C:/msys64/mingw64/bin/python.exe "d:/projects/chat application/udpclient2.py"
Client 2
Enter Name 2 : kowshik

SIGN_TAG:kowshik
ram:hi
hi
kowshik:hi
ram:check check
visible
kowshik:visible
```

TRANSFER OF FILES USING SOCKET PROGRAMMING

File Transfer: SERVER

The server performs the following functions:

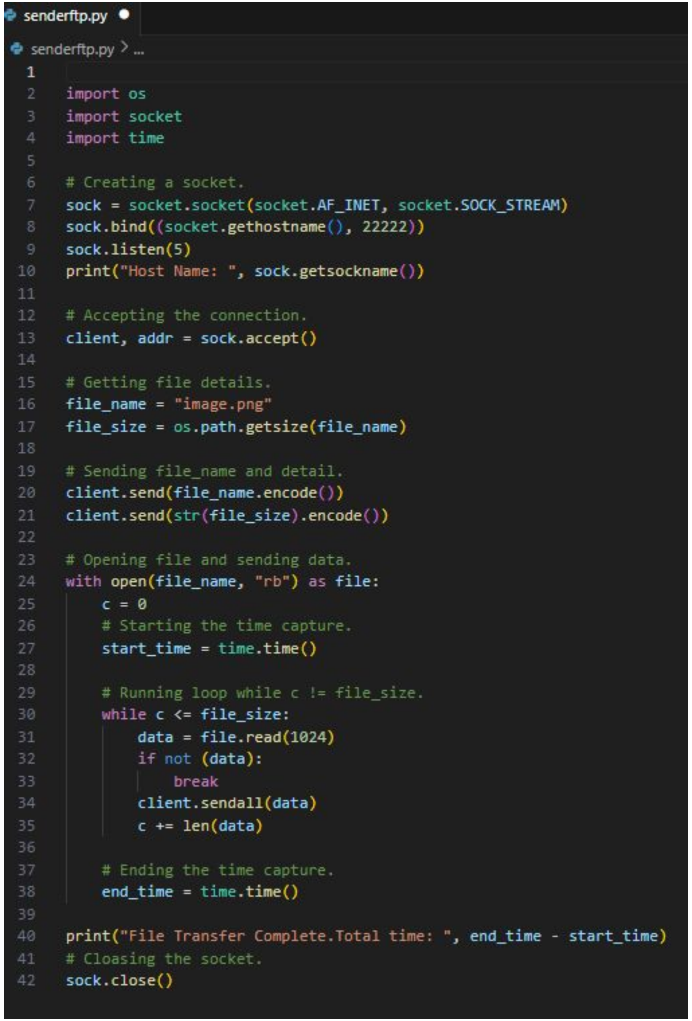
1. Create a TCP socket.
2. Bind the IP address and PORT to the server socket.
3. Listening for the clients.
4. Accept the connection from the client.
5. Receive the filename from the client and create a text file.
6. Send a response back to the client.
7. Receive the text data from the client.
8. Write (save) the data into the text file.
9. Send a response message back to the client.
10. Close the text file.
11. Close the connection.

File Transfer: CLIENT

The client performs the following functions:

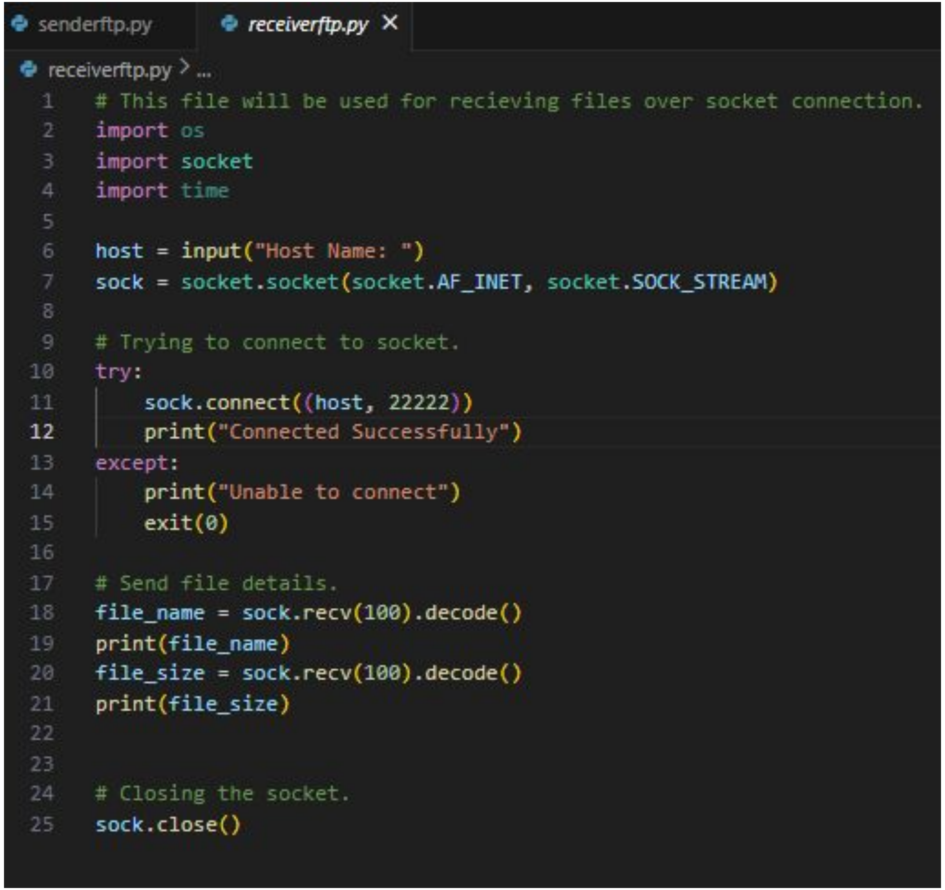
1. Create a TCP socket for the client.
2. Connect to the server.
3. Read the data from the text file.
4. Send the filename to the server.
5. Receive the response from the server.
6. Send the text file data to the server.
7. Receive the response from the server.
8. Close the file.
9. Close the connection.

sender.py



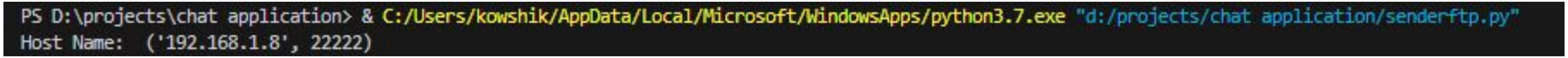
```python

import os
import socket
import time

# Creating a socket.
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind((socket.gethostname(), 22222))
sock.listen(5)
print("Host Name: ", sock.getsockname())

# Accepting the connection.
client, addr = sock.accept()

# Getting file details.
file_name = "image.png"
file_size = os.path.getsize(file_name)

# Sending file_name and detail.
client.send(file_name.encode())
client.send(str(file_size).encode())

# Opening file and sending data.
with open(file_name, "rb") as file:
    c = 0
    # Starting the time capture.
    start_time = time.time()

    # Running loop while c != file_size.
    while c <= file_size:
        data = file.read(1024)
        if not (data):
            break
        client.sendall(data)
        c += len(data)

    # Ending the time capture.
    end_time = time.time()

print("File Transfer Complete.Total time: ", end_time - start_time)
# Cloasing the socket.
sock.close()
```

reciver.py

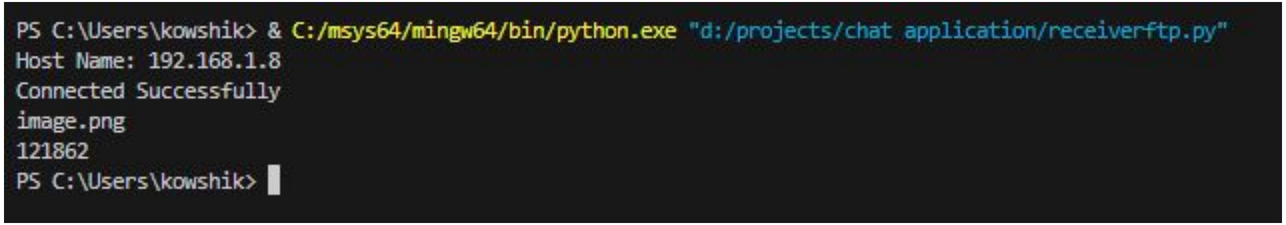


```python
# This file will be used for recieving files over socket connection.
import os
import socket
import time

host = input("Host Name: ")
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Trying to connect to socket.
try:
    sock.connect((host, 22222))
    print("Connected Successfully")
except:
    print("Unable to connect")
    exit(0)

# Send file details.
file_name = sock.recv(100).decode()
print(file_name)
file_size = sock.recv(100).decode()
print(file_size)


# Closing the socket.
sock.close()
```

OUTPUT



```
PS D:\projects\chat application> & C:/Users/kowshik/AppData/Local/Microsoft/WindowsApps/python3.7.exe "d:/projects/chat application/senderftp.py"
Host Name:  ('192.168.1.8', 22222)
```



```
PS C:\Users\kowshik> & C:/msys64/mingw64/bin/python.exe "d:/projects/chat application/receiverftp.py"
Host Name: 192.168.1.8
Connected Successfully
image.png
121862
PS C:\Users\kowshik> 
```

THANK YOU