College of Engineering,Trivandrum

Experiment 9

Network Programming Lab

Author:
Alan Anto

Registration Number :
TVE16CS09

March 27, 2019

Contents

1 Multiuser Chat server

1.1 AIM

Implement a multi user chat server using TCP as transport layer protocol.

1.2 Theory

1.2.1 TCP Protocol

Transmission Control Protocol(TCP) is a standard that defines how to establish and main- tain a network conversation via which application programs can exchange data.

The Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP). Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of octets (bytes) between applications running on hosts communicating via an IP network. Major internet applications such as the World Wide Web, email, remote administration, and file transfer rely on TCP.

When loading a web page, the computer sends TCP packets to the web server's address, asking it to send the web page to you. The web server responds by sending a stream of TCP packets, which the web browser stitches together to form the web page and display it to you.TCP is enables two way communication — the remote system sends packets back to acknowledge it is received your packets.

TCP guarantees the recipient will receive the packets in order by numbering them. The recipient sends messages back to the sender saying it received the messages. If the sender does not get a correct response, it will resend the packets to ensure the recipient received them. Packets are also checked for errors.Packets sent with TCP are tracked so no data is lost or corrupted in transit.

1.2.2 Client

A client is requester of this service. A client program request for some resources to the server and server responds to that request.

1.2.3 Server

A server is a software that waits for client requests and serves or processes them accordingly . A server may communicate with other servers inorder to serve the client requests.

1.2.4 Socket

Socket is the endpoint of a bidirectional communications channel between server and client. Sockets may communicate within a process, between processes on the same machine, or between processes on different machines. For any communication with a remote program, we have to connect through a socket port.

1.2.5 Multi Threading

A thread is sub process that runs a set of commands individually of any other thread. So, every time a user connects to the server, a separate thread is created for that user and communication from server to client takes place along individual threads based on socket objects created for the sake of identity of each client. We will require two scripts to establish the chat room. One to keep the serving running, and another that every client should run in order to connect to the server.

1.2.6 Server Side Script

The server side script will attempt to establish a socket and bind it to an IP address and port specified by the user. The script will then stay open and receive connection requests, and will append respective socket objects to a list to keep track of active connections. Every time a user connects, a separate thread will be created for that user. In each thread, the server awaits a message, and sends that message to other users currently on the chat. If the server encounters an error while trying to receive a message from a particular thread, it will exit that thread.

1.2.7 Client Side Script

The client side script will simply attempt to access the server socket created at the specified IP address and port. Once it connects, it will continuously check as to whether the input comes from the server or from the client, and accordingly redirects output. If the input is from the server, it displays the message on the terminal. If the input is from the user, it sends the message that the users enters to the server for it to be broadcasted to other users.This is the client side script, that each user must use in order to connect to the server.

2 Algorithm

2.1 Server

```
1 START
2 Create TCP SOCKET
3 Bind SOCKET t o a PORT
4 Start listing at the binded PORT for connection from CLIENTs
5 append new CLIENT to list clients
6 CREATE a new th re ad for each CLIENT which accepts messages
from CLIENT and broadcast to all in list clients
7 STOP
```

2.2 Client

```
1 START
2 Create TCP SOCKET
3 CONNECT to SERVER at IP ,PORT
4 CREATE a thread
5 WHILE true :
6 RECEIVE user input
7 SEND input to SERVER
8 WHILE true :
9 RECEIVE message from SERVER
10 PRINT message
11 STOP
```

3 Program

3.0.1 Server Side

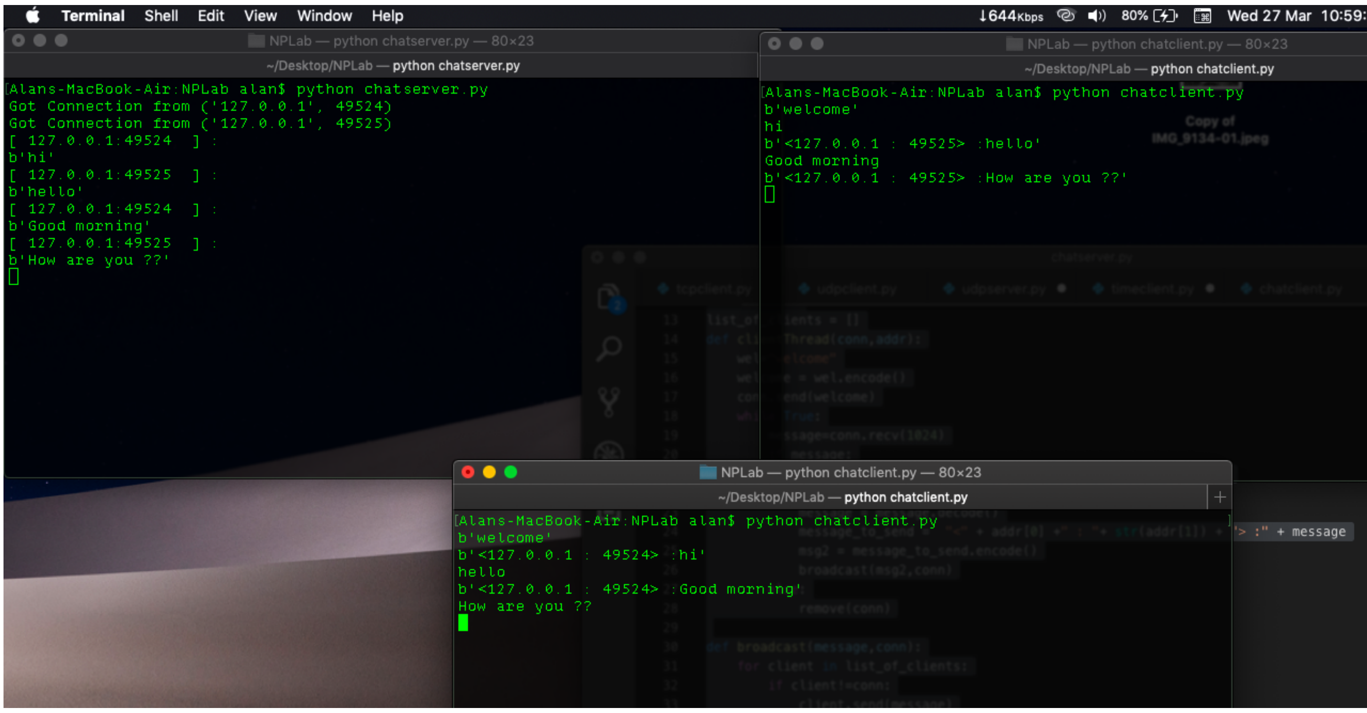
```
import socket
import socket
import select
import sys
from _thread import *

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

port=8001

s.bind(('',port))

s.listen(5)
list_of_clients = []
def clientThread(conn,addr):
    wel=”welcome”
    welcome = wel.encode()
    conn.send(welcome)
    while True:
        message=conn.recv(1024)
        if message:
            print(”[ ” + str(addr[0]) + ”:”+ str(addr[1]) +”
] :” )
            print(message)
            message = message.decode()
            message_to_send = ”<” + addr[0] +” : ”+ str(addr[1]) + ”> :”
            msg2 = message_to_send.encode()
            broadcast(msg2,conn)
        else:
            remove(conn)

def broadcast(message,conn):
    for client in list_of_clients:
        if client!=conn:
            client.send(message)

def remove(conn):
    if conn in list_of_clients:
        list_of_clients.remove(conn)
```

```
while True:
    conn,addr=s.accept()
    print ('Got Connection from',addr)
    list_of_clients.append(conn)
    start_new_thread(clientThread,(conn,addr))


conn.close()
```

3.0.2 Client Side

```
import socket
import select
import sys
from _thread import *

s= socket.socket(socket.AF_INET, socket.SOCK_STREAM)

port = 8001
s.connect(('127.0.0.1',port))
def sendMessage(so):
    while True:
        message=input()
        msg = message.encode()
        so.send(msg)

start_new_thread(sendMessage,(s,))
while True:
    data = s.recv(1024)

    print (data)


s.close()
```

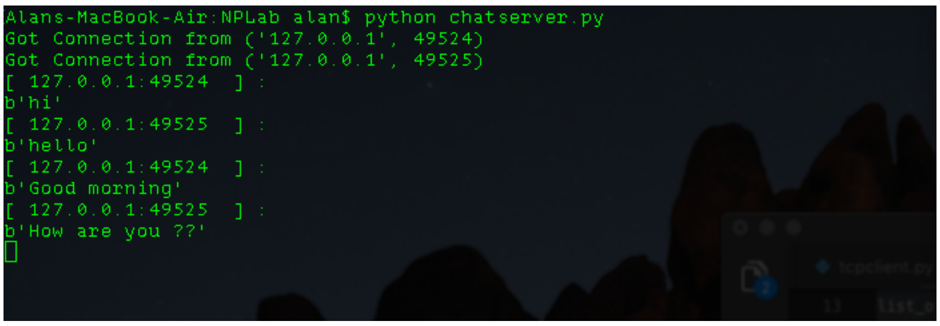
4 Output


Terminal Shell Edit View Window Help
644Kbps 80% Wed 27 Mar 10:59
NPLab — python chatserver.py — 80×23
~/Desktop/NPLab — python chatserver.py
NPLab — python chatclient.py — 80×23
~/Desktop/NPLab — python chatclient.py
NPLab — python chatclient.py — 80×23
~/Desktop/NPLab — python chatclient.py

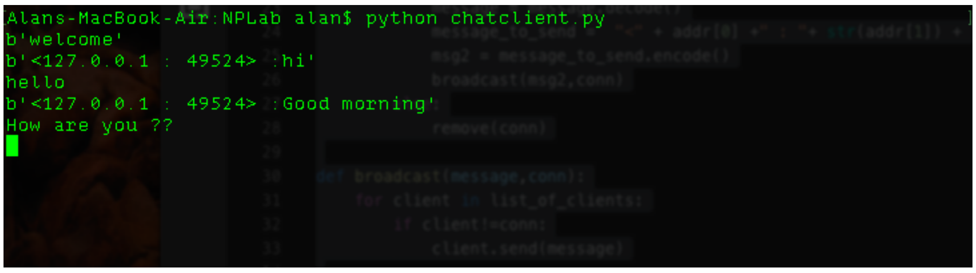
```
Alans-MacBook-Air:NPLab alan$ python chatserver.py
Got Connection from ('127.0.0.1', 49524)
Got Connection from ('127.0.0.1', 49525)
[ 127.0.0.1:49524  ] :
b'hi'
[ 127.0.0.1:49525  ] :
b'hello'
[ 127.0.0.1:49524  ] :
b'Good morning'
[ 127.0.0.1:49525  ] :
b'How are you ??'
```

```
Alans-MacBook-Air:NPLab alan$ python chatclient.py
b'welcome'
hi
b'<127.0.0.1 : 49525> :hello'
Good morning
b'<127.0.0.1 : 49525> :How are you ??'
```

```
Alans-MacBook-Air:NPLab alan$ python chatclient.py
b'welcome'
b'<127.0.0.1 : 49524> :hi'
hello
b'<127.0.0.1 : 49524> :Good morning'
How are you ??
```



```
Alans-MacBook-Air:NPLab alan$ python chatserver.py
Got Connection from ('127.0.0.1', 49524)
Got Connection from ('127.0.0.1', 49525)
[ 127.0.0.1:49524  ] :
b'hi'
[ 127.0.0.1:49525  ] :
b'hello'
[ 127.0.0.1:49524  ] :
b'Good morning'
[ 127.0.0.1:49525  ] :
b'How are you ??'
```



```
Alans-MacBook-Air:NPLab alan$ python chatclient.py
b'welcome'
b'<127.0.0.1 : 49524> :hi'
hello
b'<127.0.0.1 : 49524> :Good morning'
How are you ??
```

5 Result

Implemented TCP Multi Client Chat Server on Python 3.7.0 and executed on macOS Majove and outputs were verified.

Server code creates a TCP socket using the socket library.Then binds the server to port 8080.The socket then listens for communications to this port.In a infinite while loop the server accepts the connection and address from connecting clients.These connections appended to list clients.A thread is created for each of the clients which accepts messages from clients in its own thread and then broadcasts the message to all clients in list clients. Client code creates a TCP socket same as above.Then connects to the ip and port of the server.It then creates a thread for an infinite loop for getting user input and sending this message to server.The parent thread is also a infinite while loop that displays messages it receives from the server.