Networking Lab Assignment 9

Multi user chat server using TCP

Albin Antony

14 March 2019

1 Multi User Chat Server

1.1 Aim

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

1.2 Theory

1.2.1 Client, Server and Socket

• Server-A server is a software that waits for client requests and serves or processes them accordingly.
• Client- a client is requester of this service. A client program request for some resources to the server and server responds to that request.
• 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.

1.2.2 Multithreading

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.

1.2.3 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.

1.2.4 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.

1.3 Algorithm

1.3.1 Server

Algorithm 1 Algorithm for creating a tcp chat server

```
START
Create TCP SOCKET
Bind SOCKET to a PORT
Start listing at the binded PORT for connection from CLIENTs
append new CLIENT to list_clients
CREATE a new thread for each CLIENT which accepts messages
    from CLIENT and broadcast to all in list_clients
STOP
```

1.3.2 Client

Algorithm 2 Algorithm for creating a tcp chat client

```
START
Create TCP SOCKET
CONNECT to SERVER at IP,PORT
CREATE a thread
    WHILE true:
        RECEIVE user input
        SEND input to SERVER
WHILE true:
    RECEIVE message from SERVER
    PRINT message
STOP
```

1.4 Program

1.4.1 Server

```python
import socket
import select
import sys
from thread import *

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

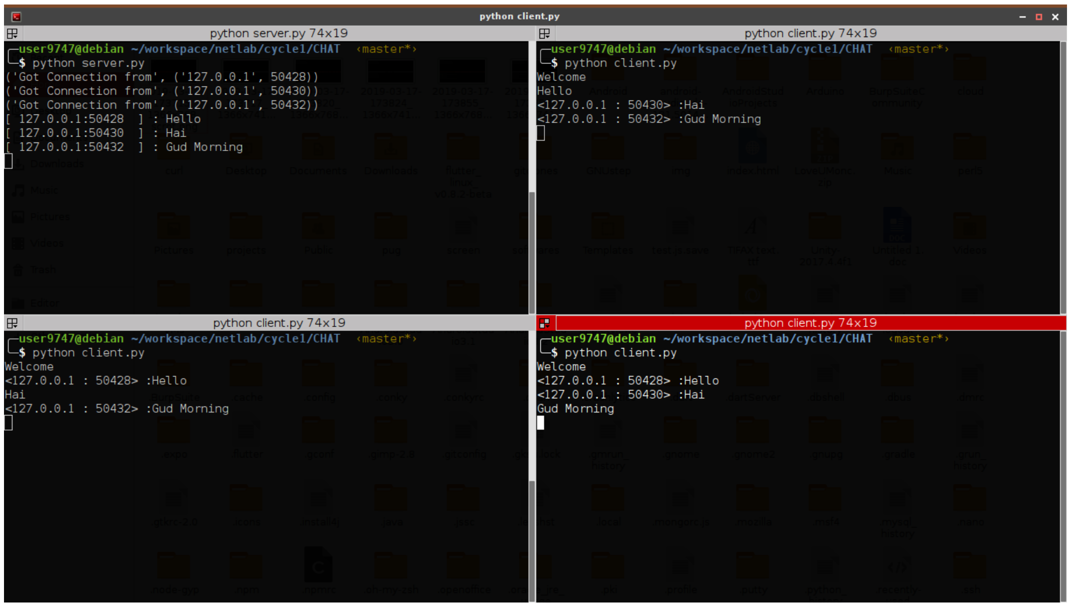
```
port=8000

s.bind(('',port))

s.listen(5)
list_of_clients = []
def clientThread(conn,addr):
    conn.send('Welcome')
    while True:
        message=conn.recv(1024)
        if message:
            print"[ "+ addr[0]+":"+str(addr[1]) +"  ] : "+
                message
            message_to_send = "<" + addr[0] +" : "+ str(addr
                [1]) + "> :" + message
            broadcast(message_to_send,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()
```

1.4.2 Client

```
import socket
import select
import sys
from thread import *

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

```
port = 8000
s.connect((sys.argv[1],port))
def sendMessage(so):
    while True:
        message=raw_input()
        so.send(message)

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


s.close()
```

1.5 Output



```
user9747@debian ~/workspace/netlab/cycle1/CHAT  ‹master*›
$ python server.py
('Got Connection from', ('127.0.0.1', 50428))
('Got Connection from', ('127.0.0.1', 50430))
('Got Connection from', ('127.0.0.1', 50432))
[ 127.0.0.1:50428  ] : Hello
[ 127.0.0.1:50430  ] : Hai
[ 127.0.0.1:50432  ] : Gud Morning
```

```
user9747@debian ~/workspace/netlab/cycle1/CHAT  ‹master*›
$ python client.py
Welcome
Hello
<127.0.0.1 : 50430> :Hai
<127.0.0.1 : 50432> :Gud Morning
```

```
user9747@debian ~/workspace/netlab/cycle1/CHAT  ‹master*›
$ python client.py
Welcome
<127.0.0.1 : 50428> :Hello
Hai
<127.0.0.1 : 50432> :Gud Morning
```

```
user9747@debian ~/workspace/netlab/cycle1/CHAT  ‹master*›
$ python client.py
Welcome
<127.0.0.1 : 50428> :Hello
<127.0.0.1 : 50430> :Hai
Gud Morning
```

1.6 Result

Implemented TCP Multi Client Chat Server on Python 2.7.13 and executed on Debian 4.9 Kernel 4.9 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.