Documentation

Computer Network Lab Assignment 2

Socket Programming Basics

Implementation of a simple text exchange program like a messenger

Group:

Rishu Garg (CS11B030)

Vikash Kumar (CS11B038)

Design:

1. One computer will work as a server. Two or more computers will work as clients.

2. The software is a command line application.

*3. In Both server and clients, the exchanged messages shown in **chronological** order with a nickname of sender (not the IP address).*

*4. The protocol used to exchange messages is **TCP**.*

Implementation:

The chat application we made can be used for one-to-one as well as one- to-many or many-to-many communication. So this means that multiple users can connect to the chat server and send their messages to a specific person or to a group of person.

It has two sides, server side and client side.

Server side:

Server will accept multiple incoming connections for client and Read incoming messages from each client and send them to all or to

specific connected client as required by sender. The server handles multiple chat clients which is implemented using multithreading. If any of the Client Socket is readable then it means that one of the chat Client has send a message.

The clients_conn will be an dictionary consisting of all nickname as key and socket descriptors as value corresponding to that key . Clients is a dictionary where address is key and nickname is the value. So if the server socket is readable, that means a new connection has come and it will accept the new connection if the limit is not exceeding. If any of the Client Socket is readable, the server would read the message and extract the receivers nickname and then send the message to them. If the send function fails to send message to any of the client, the client is assumed to be disconnected and the connection is closed and the entry pertaining to the socket is removed from the connection list.

Client Side:

Client Listen for incoming messages from the server. Check user input. If the user types in a message then send it to the server. The client has to actually listen for server message and user input at the same time. To do this, we use the select function. The select function can monitor multiple sockets or file descriptors for some "interesting activity" which is this case is readable. When a message comes from the server on the connected socket, it is readable and when the user types a message and hits enter, the stdin stream is readable.

So the select function has to monitor 2 streams. First is the socket that is connected to the remote webserver, and second is stdin or terminal input stream. The select function blocks till something happens. So after calling select, it will return only when either the server socket receives a message or the user enters a message. If nothing happens it keeps on waiting.

We simply create an array of the stdin file descriptor that is available from the sys module, and the server socket ***s****. Then we call the select function passing it the list. The select function returns a list of arrays*

that are readable, writable or had an error. The readable sockets will be again a list of sockets that is readable.
So in this case, the read_sockets array will contain either the server socket, or stdin or both. Then the next task is to do relevant processing based on which socket is readable. If the server socket is readable, it means that the server has send a message on that socket and so it should be printed. If stdin is readable, it means that the user typed a message and hit enter key, so that message should be read and send to server as a chat message.

Extra Feature:

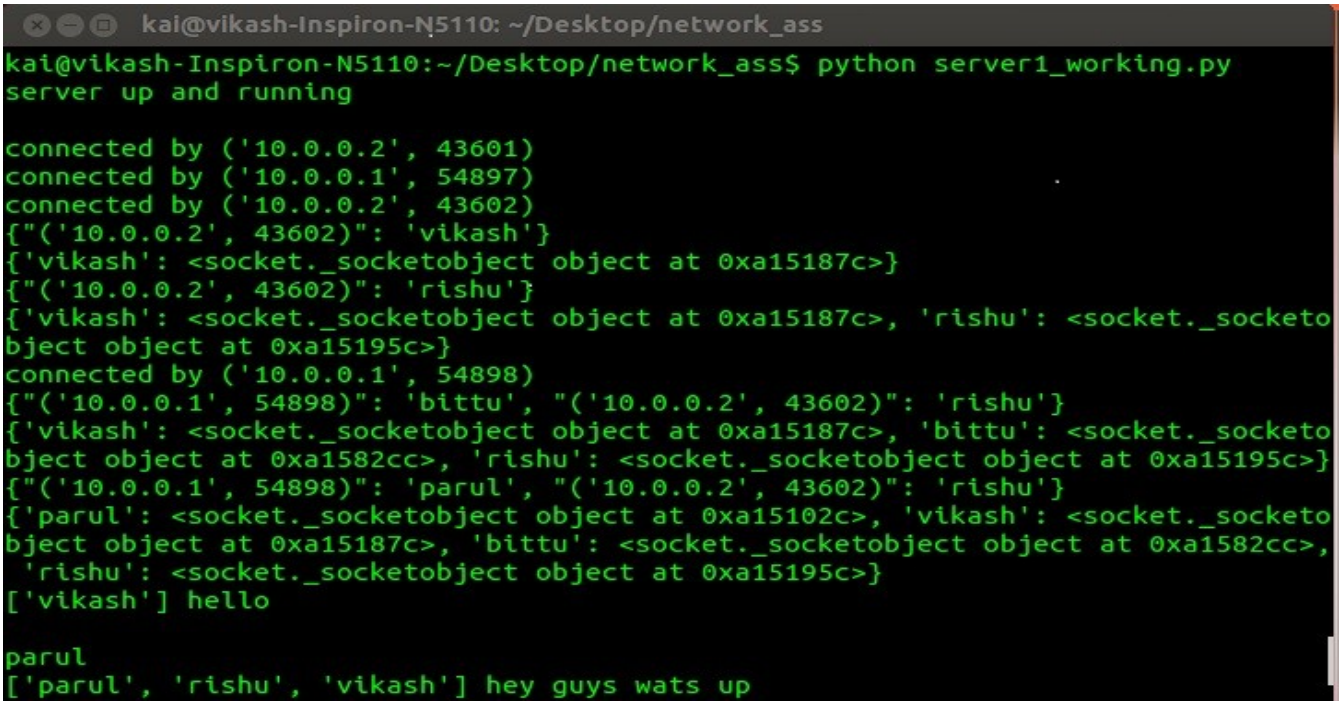
Here we did multiplexing. Server will receive the message from client and send them according to our requirements, means we can do private chat by just typing the name of the person to whom we want to send the message. Also we can send the message to all the clients that are connected. This concept is called as grouping.

Tested Environment:

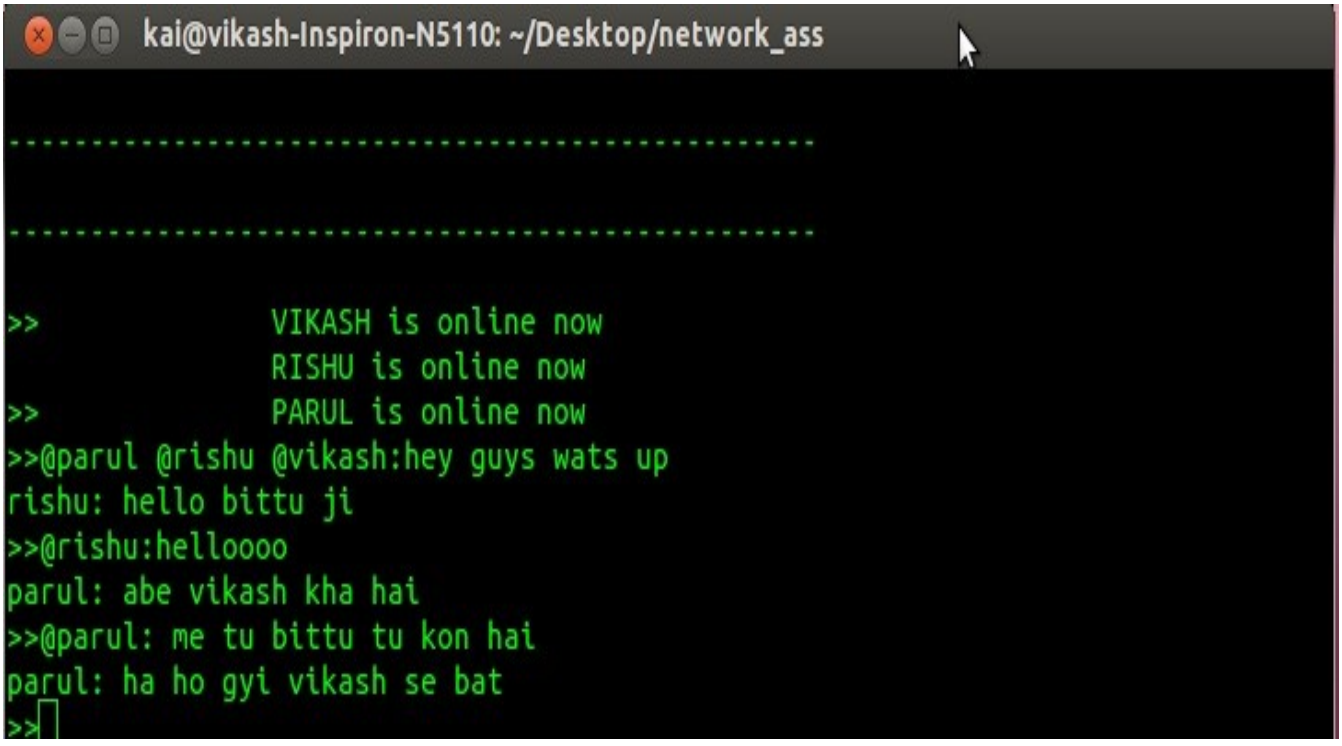
We have implements this on Linux (Ubuntu) Platform.

Screen Shots

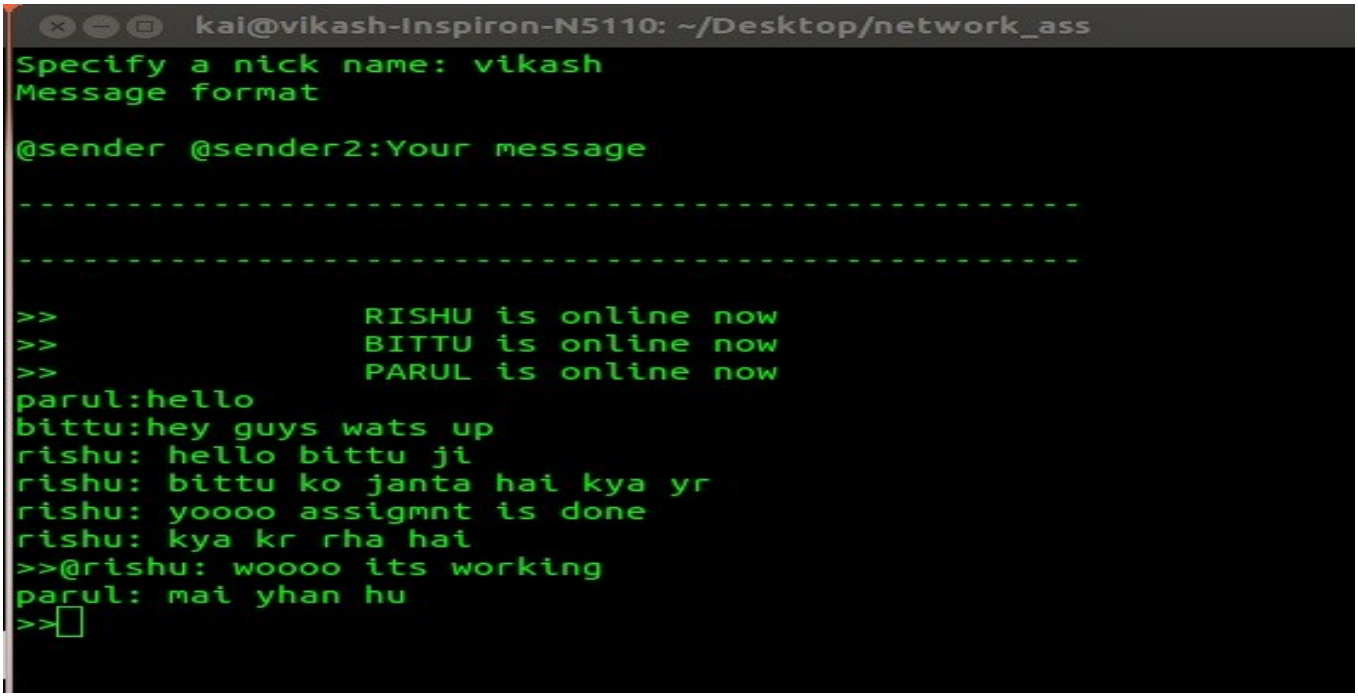
Server



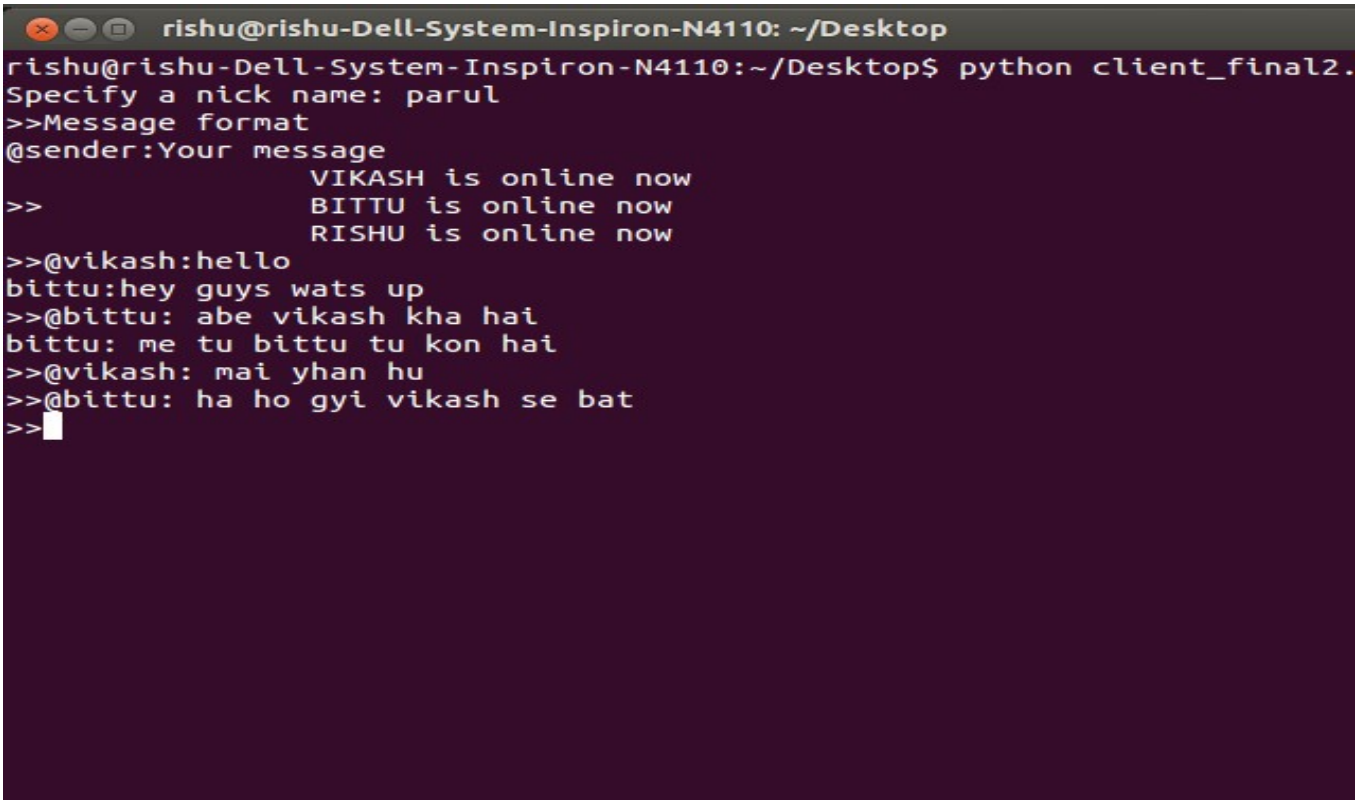
Client 1:



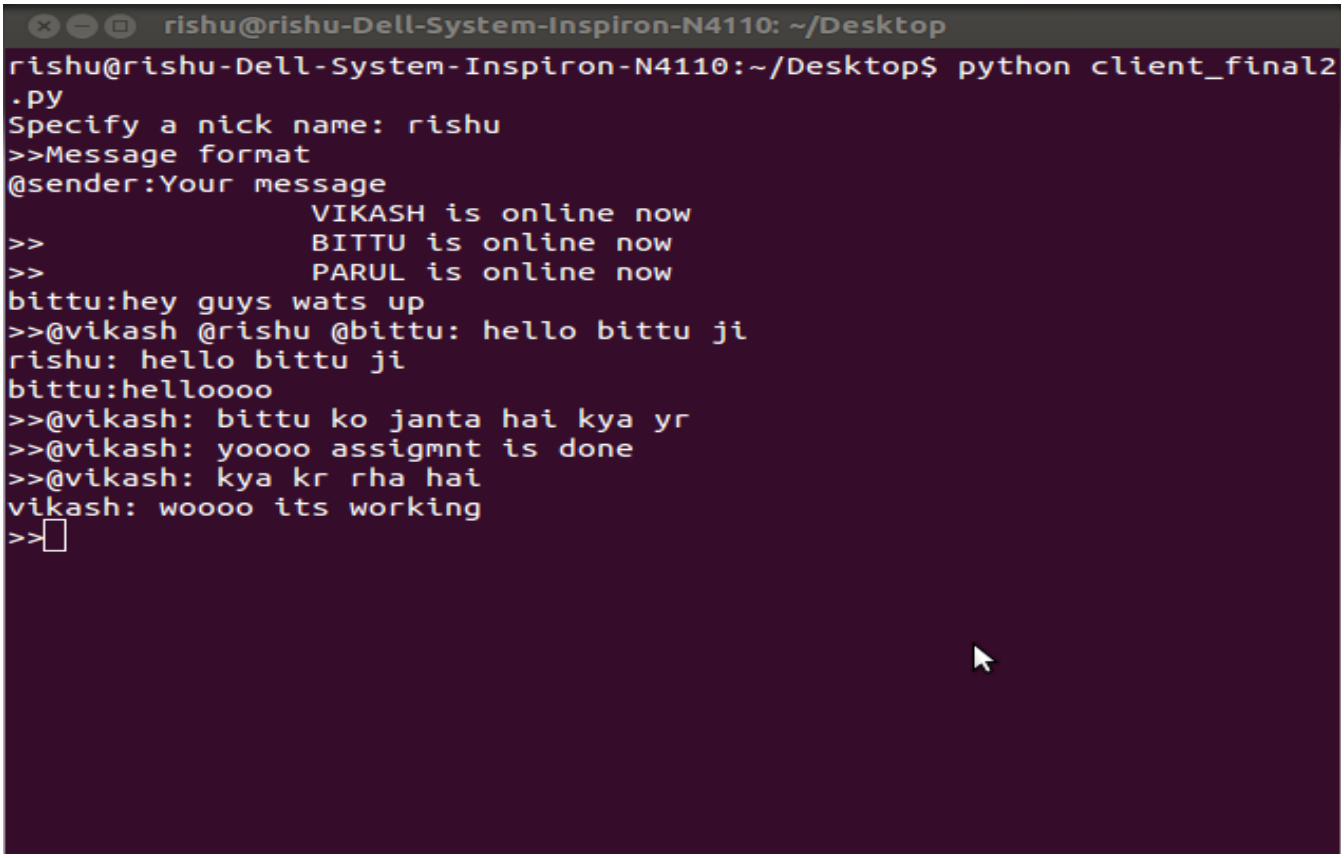
Client 2:


kai@vikash-Inspiron-N5110: ~/Desktop/network_ass
Specify a nick name: vikash
Message format
@sender @sender2:Your message
>> RISHU is online now
>> BITTU is online now
>> PARUL is online now
parul:hello
bittu:hey guys wats up
rishu: hello bittu ji
rishu: bittu ko janta hai kya yr
rishu: yoooo assigmnt is done
rishu: kya kr rha hai
>>@rishu: woooo its working
parul: mai yhan hu
>>

Client 3:


rishu@rishu-Dell-System-Inspiron-N4110: ~/Desktop
rishu@rishu-Dell-System-Inspiron-N4110:~/Desktop$ python client_final2.
Specify a nick name: parul
>>Message format
@sender:Your message
VIKASH is online now
>> BITTU is online now
RISHU is online now
>>@vikash:hello
bittu:hey guys wats up
>>@bittu: abe vikash kha hai
bittu: me tu bittu tu kon hai
>>@vikash: mai yhan hu
>>@bittu: ha ho gyi vikash se bat
>>

Client 4:



References:

Computer Networks A System Approach (Larry L Peterson and Bruce S. Davie)

http://neerajkhandelwal.wordpress.com/2012/02/16/socket-programming-handling-multiclients/

http://docs.python.org/2/library/select.html

http://neerajkhandelwal.wordpress.com/2012/02/11/socket-programing-python/

http://docs.python.org/3/library/socketserver.html