****

|  |
| --- |
|  |
|
|
|

**IT-308**

**Operating Systems**

**Multi-threaded Client/Server Chat Application**

**Team-Members**

**Krishanu Konar-201401127**

**Kunal Suthar-201401131**

**Socket in Networking:**

A network socket is an endpoint of an inter-process communication flow across a computer network. Sockets may communicate within a process, between processes on the same machine, or between processes on different machines. Today, most communication between computers is based on the internet protocols; therefore most network sockets are internet sockets. To create a connection between machines, Python programs import the socket module, create a socket object, and call the object’s methods to establish connections and send and receive data. Sockets are the endpoints of a bi-directional communication channel.

**Network Connection Mechanism:**

A network connection is initiated by a client program when it creates a socket for the communication with the server. To create the socket in Python, the python socket library is used for establishing a TCP Connection. After that, we use the connect(host,port) function of that socket object to connect to the server. At this stage, the server must be started on the machine having the specified address and listening for connections on a specific port number. The server uses a specific port dedicated only to listening for connection requests from clients. So, its specific port is dedicated only to listening for new connection requests. The server side socket associated with specific port is called server socket. Here after defining a TCP connection, we need to bind in the host IP and the port number, to define the socket on the server side, so that multiple clients could connect to it. When a connection request arrives on this socket from the client side, the client and the server establish a connection. A new thread is started by the server to manage this client’s requests.

Thus, to summarize the above description sequentially, the connection is established as follows:

1. *When the server receives a connection request on its specific server port, it creates a new thread for each client’s request and binds a port number to it.*
2. *It sends the new client’s information to all existing connected clients.*
3. *The server goes on now by listening on the port:*
   1. *It waits for any new incoming connection requests on its specific port.*
   2. *It reads and writes messages on established connection with the accepted client. The server communicates with the client by reading from and writing to the port. The message incoming from the client is used to find out the destination client(s) and then the message is forwarded to the requested client. Also, if other connection requests arrive, the server accepts them and processes them through multithreading and creates a separate thread for every new connection. Thus, at any instant, the server must be able to communicate simultaneously with many clients and to wait on the same time for incoming requests on its specific server port. The communication with each client is done via the client and server sockets. We’ve used stream communication protocol. The stream communication protocol is known as TCP (transfer control protocol). TCP is a connection -oriented protocol. In order to communicate over the TCP, a connection must first be established between the two sockets. While one of the sockets listens for a connection request (server), the other asks for a connection (client). Once the two sockets are connected, they can be used to transmit and/or to receive data. When we say "two sockets are connected" we mean the fact that the server accepted a connection.*

**Implementation:**

The chat application can be used for one-to-one as well as one-to-many communication. This means that multiple users can connect to the chat server and send their messages to a specific person or to a group of persons, or everyone in the chatroom.

It has two sides, **client** side and **server** side*.*

**The Chat Client**

Upon successful connection, client listens for incoming messages from the server. It checks user input. If the user types in a message, then directly send it to the server. The client has to actually listen for the 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, stdin is readable and when the user types a message and hits enter, the stdout stream is readable. So the select function has to monitor 2 streams. The first one 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 client socket receives a message or the user sends a message. If nothing happens, it keeps on waiting. We simply create an array of the stdin file descriptors that is available from the sys module, and the server sockets. Then we call the select function passing it the list. The select function returns a list of sockets that are readable, writable or had an error. So, 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. The code below is the multi-threaded chat client:

* **Client**

**import socket**

**import select**

**import sys**

**host, port = "", 0**

**def connection\_address():**

**# Get Connection address; Client connects to this server and can contact everyone connected to this server**

**global host**

**global port**

**host = raw\_input("Enter Server's IP: ")**

**port = int(raw\_input("Enter Connecting Port: "))**

**def prompt():**

**sys.stdout.write(">>")**

**sys.stdout.flush()**

**def main():**

**#creating a tcp connection**

**soc = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)**

**soc.settimeout(100)**

**soc.connect((host,port)) #connecting to server**

**print '\n\t\t\t-------------------'**

**print "\t\t\tWelcome to Chatbox!"**

**print '\t\t\t-------------------\n'**

**print " Please Use the Following Message format: \n" #specifing the format of message**

**print ' @person\_nick: "Your message"'**

**print "\n Press 'Ctrl+C' to disconnect from the server"**

**print " ---------------------------------------------\n"**

**print soc.recv(1024),**

**nick\_name = raw\_input()**

**soc.send(nick\_name) #sending nickname to server**

**prompt() #displaying the prompt**

**while 1:**

**sock\_list = [sys.stdin, soc] #creating a list of socket and std prompt**

**try:**

**read\_sockets, write\_sockets, err\_sockets = select.select(sock\_list,[],[])**

**except:**

**print "Successfully disconnected from chat server"**

**sys.exit()**

**for sock in read\_sockets:**

**if sock == soc:**

**data = sock.recv(1024) #receiving message from the server**

**if data == -1: print 'lol'**

**if not data:**

**print "\n\nServer was shut down!!!\nExitting......"**

**sys.exit()**

**else:**

**sys.stdout.write(data)**

**#displaying the incoming message on the screen**

**prompt()**

**else:**

**message = sys.stdin.readline()**

**#reading the user prompt and sending the message to server**

**if message == "\n": pass**

**elif "@" not in message or ":" not in message:**

**print "Invalid message! Use Given Format (@person\_nick:**

**'Your message')"**

**else:**

**message = "@"+nick\_name+" " + message**

**soc.send(message)**

**prompt()**

**if \_\_name\_\_ == '\_\_main\_\_':**

**connection\_address()**

**main()**

**The chat Server**

Server will accept multiple incoming connections from the client and read incoming messages from each client and send them to all or to specific connected client as required by the 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 sent a message. The clients\_conn will be an dictionary consisting of all nicknames as key and socket descriptors as value corresponding to that key. Clients is another 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 Sockets are readable, the server would read the message and extract the receiver’s nickname, and then send the message to them. If the send function detects a client who is not present in the connected clients list, it does not send message to any of the clients, and sends a message back to the sender stating the requested person is offline. If/When a client decides to leave the chatroom, it closes its client app and server detects this change. The connection is closed, the entry pertaining to the socket is removed from the connection list and a message is sent to all the connected clients regarding the unavailability of the recently disconnected client.

* **Server**

**import socket**

**import thread**

**import string**

**##########################################################################################**

**# Below function alert all other connected clients if a new connects to the server.**

**# It takes nickname of the connected client and according sends the message to all other # connected clients that a new client is available**

**#########################################################################################**

**def send\_message\_all(nickname):**

**for soc in clients\_conn:**

**if soc!=nickname:**

**clients\_conn[soc].send("\r\t\t"+nickname.upper()+" is online now\n")**

**for z in clients\_conn:**

**if z!=nickname:**

**clients\_conn[nickname].send("\r\t\t"+z.upper()+" is online now\n")**

**###############################################################################################**

**# Below fucntion takes the message the send to by the client to the server which is to be # forwarded to specific clients**

**# and this fuction extracts the receiver and convert it into a list and send it back along with # the actual message**

**###############################################################################################**

**def getreceiver(data):**

**raw\_text = string.split(data,":") # @user(s) : message**

**if len(raw\_text)<2:**

**print 'exception occured'**

**return (-1,-1)**

**users = string.split(raw\_text[0]," ")**

**for i in range(len(users)):**

**users[i]=users[i].replace("@","")**

**users[i]=users[i].lower()**

**return (users,raw\_text[1])**

**def get\_users():**

**print "Current Online Users: "**

**for c in clients:**

**print clients[c],**

**##############################################################################################**

**# This function is one on which thread runs. It takes of sending and receiving of messages of # client and forwarding it**

**# to other clients mentioned in the user's message**

**##############################################################################################**

**def clientthread(conn):**

**# print "connected by",addr**

**conn.send("Specify a nick name:") #asking the user his nickname**

**nick = conn.recv(1024)**

**nick = nick.lower()**

**clients[str(addr)]=nick #adding an entry to store the nickname**

**clients\_conn[nick]=conn #adding an entry to store the connection**

**#corresponding to nickname**

**print nick, "," , addr, "joined....\n"**

**# print clients**

**# print clients\_conn**

**send\_message\_all(nick) #alerting all users about the new connected client**

**while 1:**

**try:**

**data=conn.recv(1024) #receving message**

**a,b=getreceiver(data) #extracting the receivers from the message**

**if a==-1:**

**clients\_conn[nick].send("\r\nInvalid message\n")**

**data=conn.recv(1024)**

**a,b=getreceiver(data)**

**if a==-1:**

**clients\_conn[nick].send("\r\nInvalid message!!!**

**Terminating.....\n")**

**clients\_conn.pop(nick)**

**# notifying other clients**

**clients.pop(str(addr))**

**conn.close()**

**return**

**except Exception:**

**print nick, "went offline.\n"**

**for v in clients\_conn:**

**try:**

**clients\_conn[v].send("\r\t\t"+nick+" is offline**

**now\n")**

**#handling the case when a user disconnects and**

**except:**

**pass**

**try:**

**clients\_conn.pop(nick)**

**# notifying other clients**

**clients.pop(str(addr))**

**conn.close()**

**return**

**except:**

**pass**

**return**

**if a[1] == 'all':**

**for cl in clients\_conn:**

**if cl != nick:**

**clients\_conn[cl].send("\rMessage from "+nick+":**

**"+b+"\n")**

**else:**

**for f in range(1,len(a)):**

**try:**

**clients\_conn[a[f]].send("\rMessage from "+nick+":**

**"+b+"\n")**

**#sending incomming message to mentionded receivers**

**except:**

**string = "\t\t\t" + str(a[f]) + " is offline.\n"**

**clients\_conn[nick].send(string)**

**host = '10.100.88.32' #declaration of port and hostname**

**port = 12341**

**s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) #creating a TCP socket**

**s.bind((host, port)) #binding the hostname and port to socket created**

**s.listen(10) #listening to at max 10 connection including server**

**clients={} # {add:nickname}**

**clients\_conn={} # {nickname:conn}**

**print "Server started!\nIP: %s\nPort: %s\n\nWaiting for clients.....\n" % (host,port)**

**i=1**

**while i<10:**

**conn, addr = s.accept() #accepting new connection**

**i+=1**

**try:**

**#creating a new thread for the new client**

**thread.start\_new\_thread(clientthread,(conn,))**

**except Exception:**

**print conn.getpeername(), "left."**

**for v in clients\_conn: #handling if the thread fails**

**if v!=clients[str(conn.getpeername())]:**

**clients\_conn[v].send(clients[str(conn.getpeername())]+" is offline**

**now")**

**clients\_conn.pop(clients[str(conn.getpeername())])**

**clients.pop(str(conn.getpeername()))**

**conn.close()**

**i-=1**

**s.close() #closing the connection**

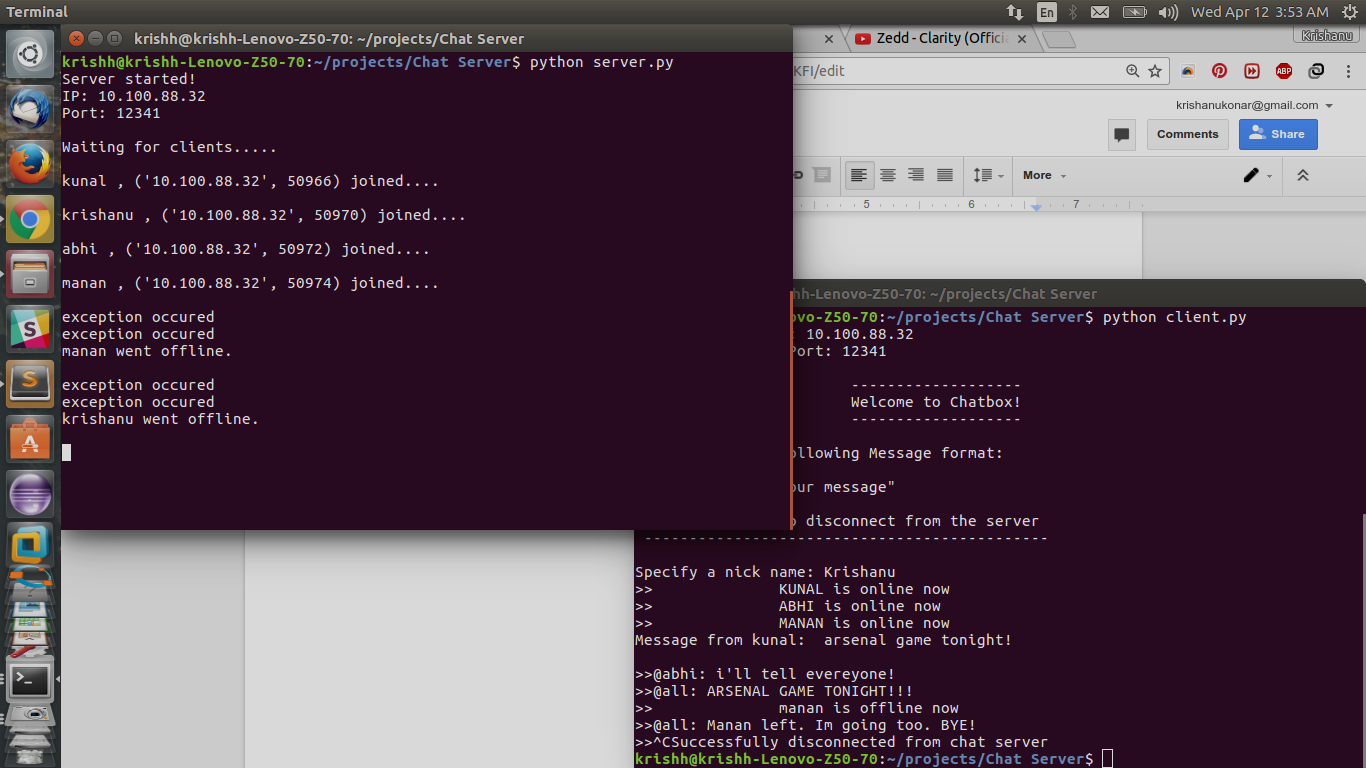
**Tested Environment:**

* Ubuntu 15.10, 16.04 LTS
* Mint 18 Cinnamon
* Fedora 13 (Constantine)

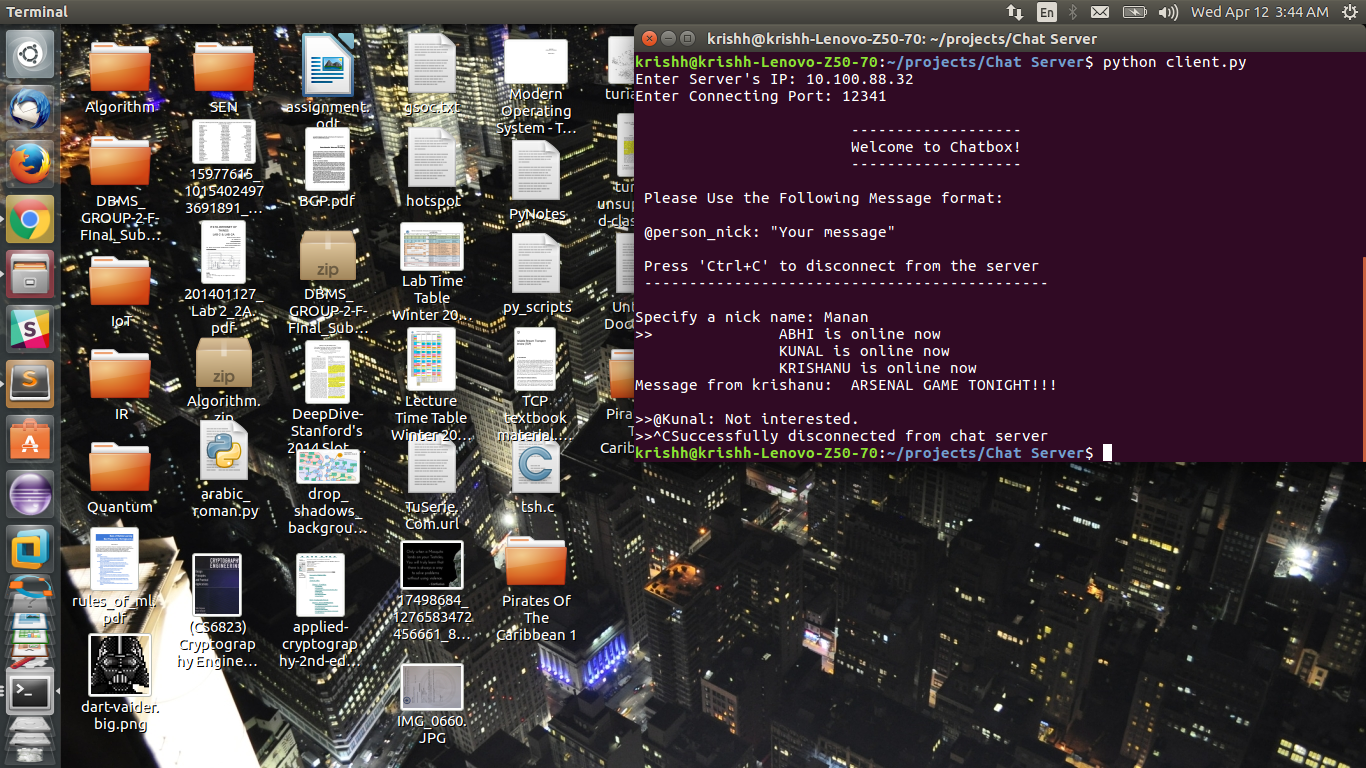
**Chat Example:**

We take 4 clients in a Group chat. A short sample conversation is emulated using 4 clients. Messages sent are varied, including one-to-one messaging and broadcasting. Sending messages to multiple clients is also supported.

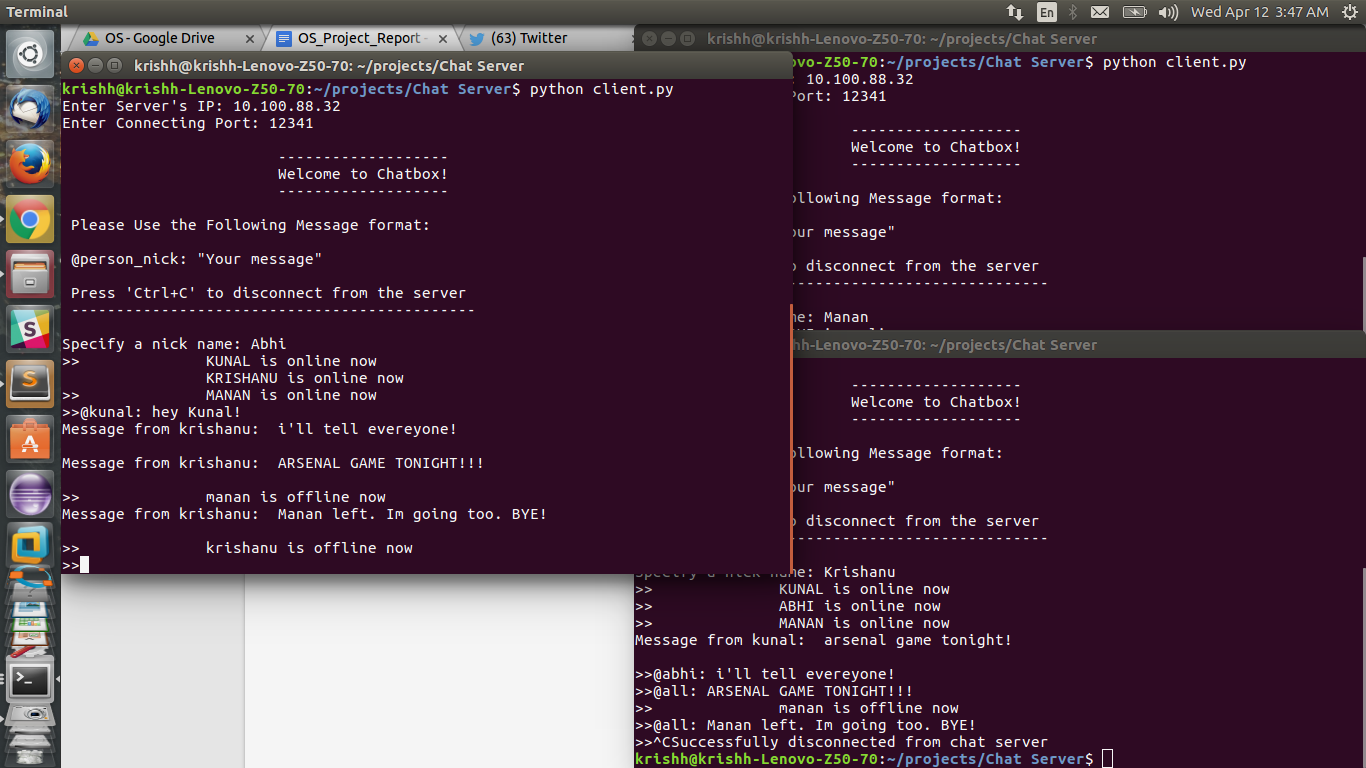
**Server**

****

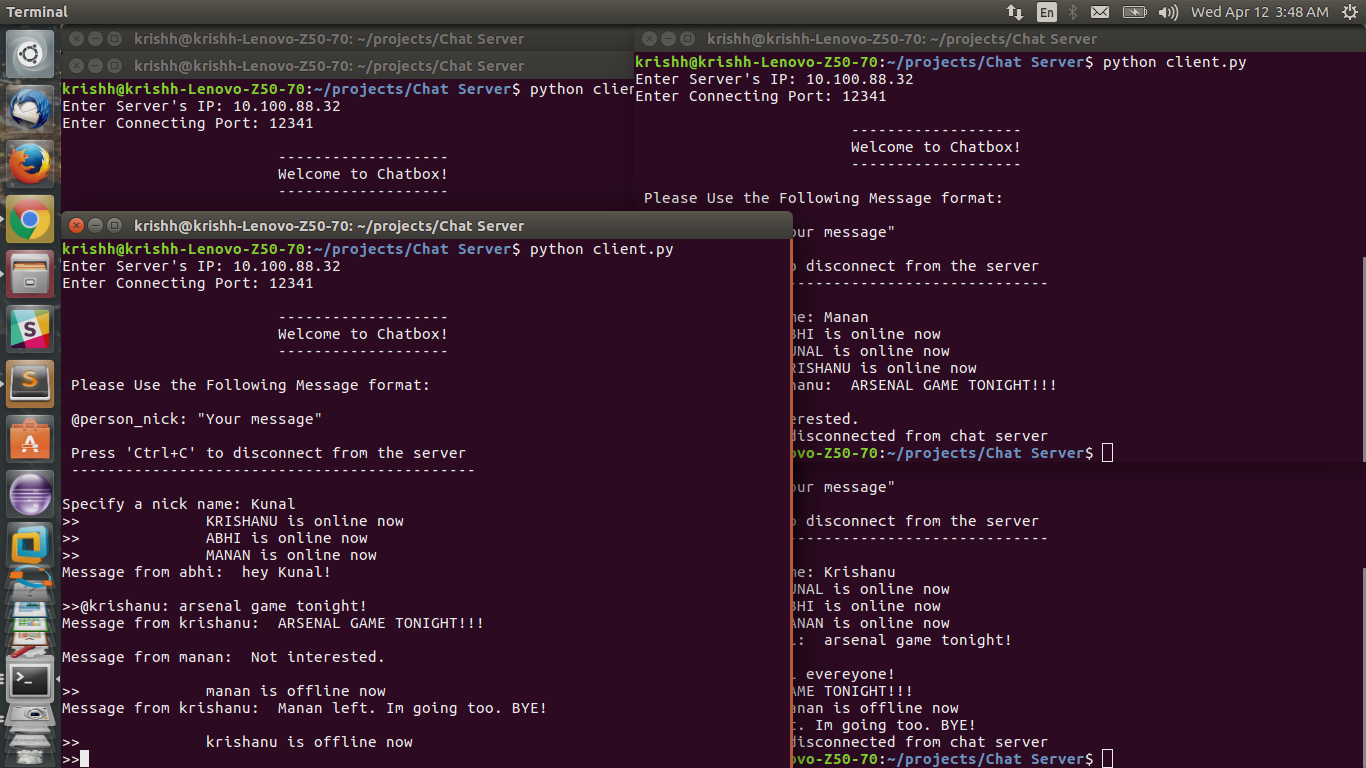
**Client 1 (Manan):**

****

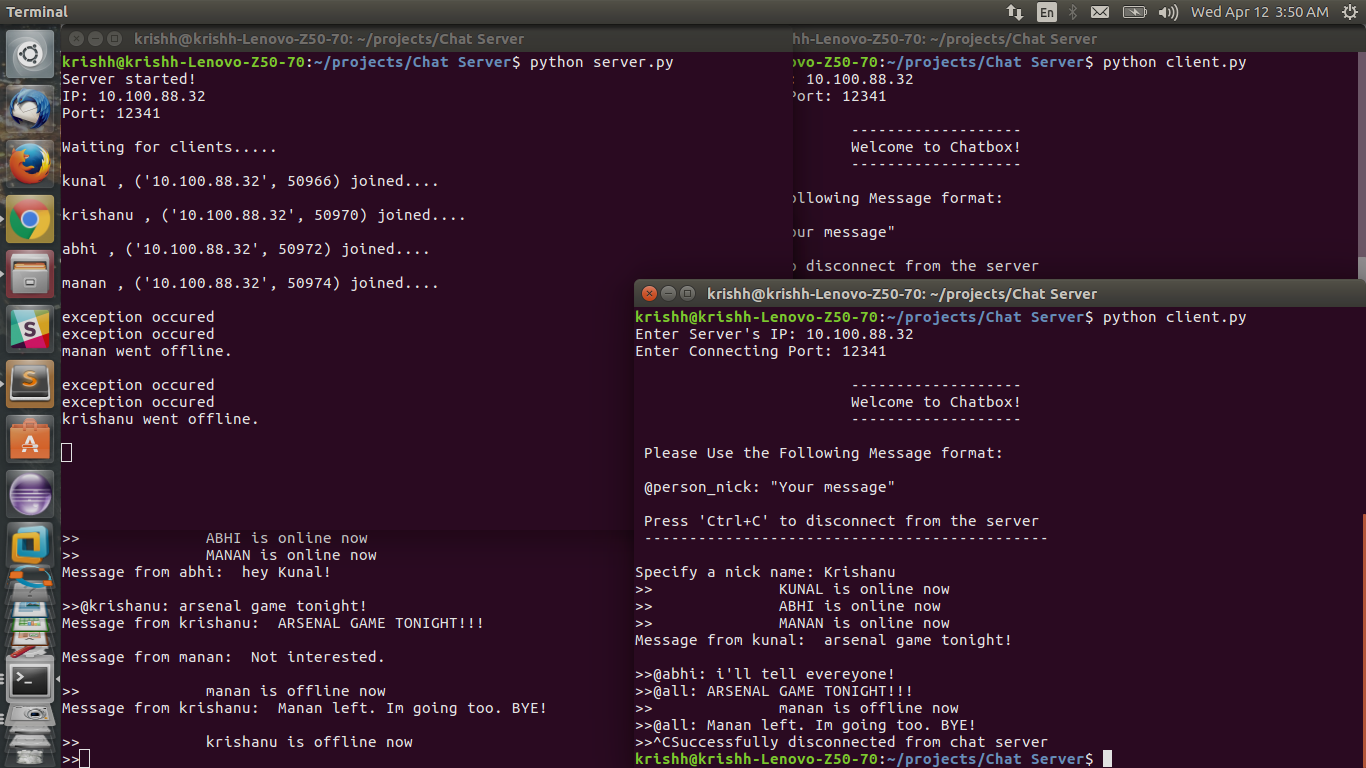
**Client 2 (Abhi):**

****

**Client 3 (Kunal):**

****

**Client 4 (Krishanu):**

****