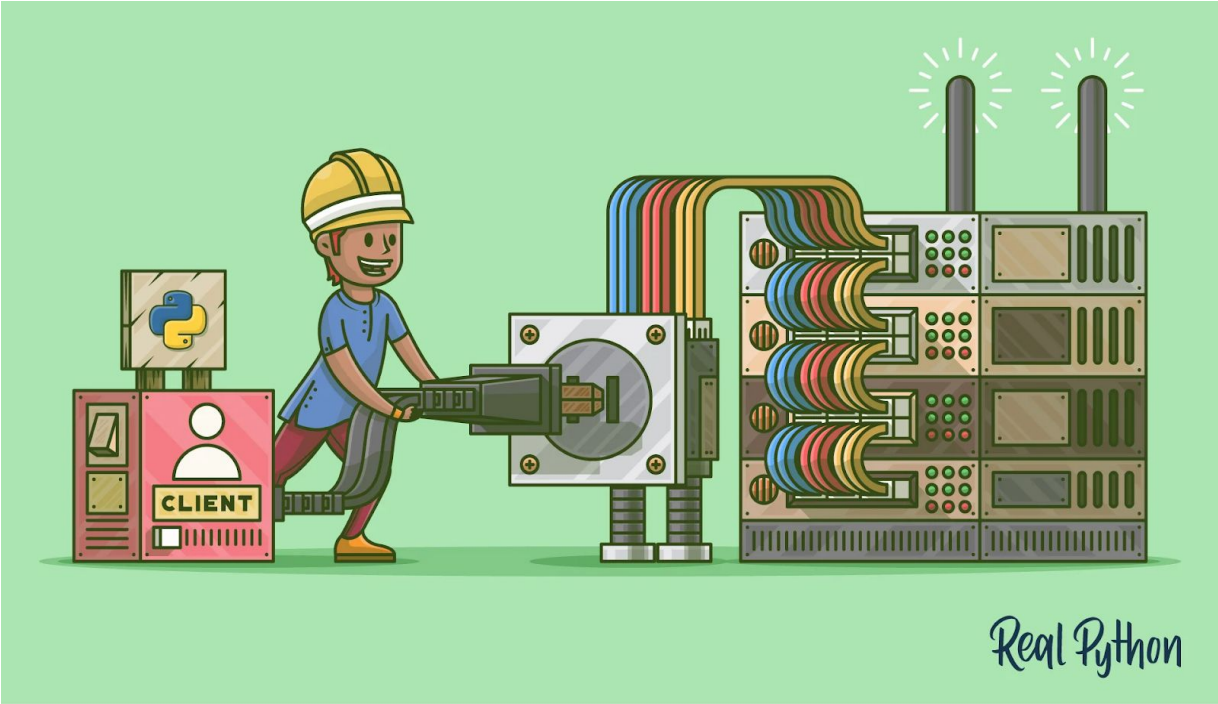
OS PROJECT

CHAT ROOM



GROUP MEMBERS:

Kavya Vemuri	**CED18I027**
Jaya Sathwika	**COE18B019**
Dara Sanjana	**CED18I015**
Sai Dheekshitha	**CED18I026**
Sri Silpa Roy	**CED18I013**
Medi Akarsha	**CED18I033**

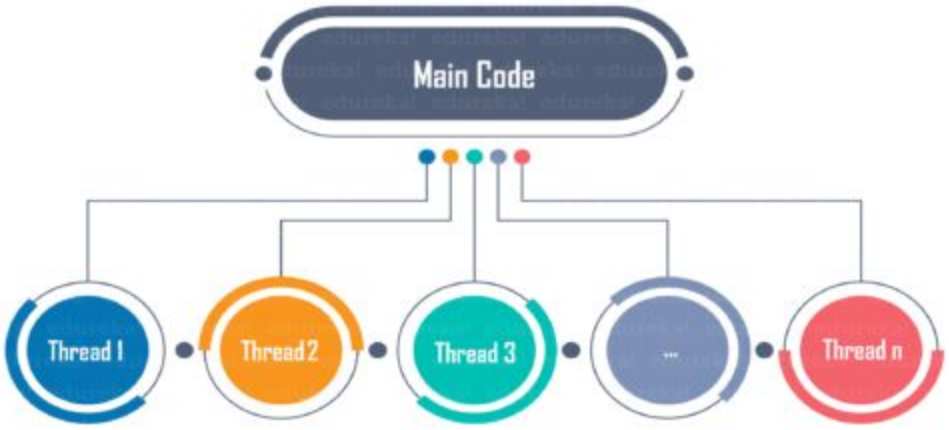
Socket programming

Sockets can be thought of as endpoints in a communication channel that is bi-directional and establishes communication between a server and one or more clients. Here, we set up a socket on each end and allow a client to interact with other clients via the server. The socket on the server side associates itself with some hardware port on the server-side. Any client that has a socket associated with the same port can communicate with the server socket.

Multi-Threading

A thread is a sub-process that runs a set of commands individually or any other thread. So, every time a user connects to the server, a separate thread is created for that user, and communication from the server to the client takes place along individual threads based on socket objects created for the sake of the identity of each client.

We will require two scripts to establish this chat room. One to keep the serving running, and another that every client should run in order to connect to the server.



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 (windows users might have to make an exception for the specified port number in their firewall settings, or can rather use a port that is already open). 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 encountered an error while trying to receive a message from a particular thread, it will exit that thread.

Technologies used:

- ➔ Multithreading
- ➔ Sockets
- ➔ Python GUI

Usage:

➔ Here, we have a public host as the server, which connects clients from different areas and establishes the chat room between them.(Remote Connection)

➔ Run the server, in the public host available to you(loopback ip can also be used) and set the IP address of the public host in server particulars of client code, and the members willing to chat should run the client code.

➔ All clients get connected to the host. Every client is asked to choose the name for their chat room which is visible as the title for their chatroom (It is chosen individually).

➔ Every client trying to connect to the server with a specific port enters into a specific chatroom. Trying with different servers and ports leads to different chat rooms.

CODE:

SERVER SIDE:

```
import re
import os
import json
import threading
import time
import socket
from threading import Thread
```

```
import pytz
from datetime import datetime
from json import dumps

class Helper:
    # encode the request and send it.
    def encodehttprequest(self, messsage, timestamp):
        body = {'message': messsage, 'time': timestamp}
        body_str = dumps(body)
        method = "POST"
        contentType = "application/json; " + '\n' + "Accept-charset = UTF-8"
        userAgent = "Chat Room"
        host = '127.0.0.1'
        contentlength = len(body_str)
        tz_NY = pytz.timezone('Asia/Kolkata')
        datetime_NY = datetime.now(tz_NY)
        date=datetime_NY.strftime("%H:%M")
        titleheader = 'HTTP/1.1 200 OK\r\n'

        headers = method + ' ' + titleheader + "Host: " + host + '\r\n' +
"Content-Length: " + str(contentlength) + "\r\n" + "User-Agent: " + \
                  userAgent + "\r\n" + "Content-Type: " + contentType + '\r\n' +
'Date: ' + date

        encodedMessage = headers + "\r\n\r\n" + body_str
        return encodedMessage

def accept_connections():
    while True:
        client, client_address = server_socket.accept()
        # print("%s:%s has connected." % client_address)
        # it encodes the message from client and sent it to server.
        encoded_message = Helper().encodehttprequest(
            messsage="Greetings from the server! Now type your name and
press enter!", timestamp=time.time())
        client.send(encoded_message.encode())
```

```
        addresses[client] = client_address
        Thread(target=handle_client, args=(client,)).start()


def handle_client(client):  # Takes client socket as argument.
    try:
        while 1:
            # server receives the message every time client sends it.
            name = client.recv(buffer_size).decode("utf8")
            msg_after_split = name.split('\r\n\r\n')
            body = json.loads(msg_after_split[1])
            name = body['message']
            # check the bad name here.
            if re.match('^[a-z A-Z]+$', name):
                break
            else:
                error_message = Helper().encodehttprequest(messsage='Only
alphabets are allowed, Numbers or special characters are not allowed.',
timestamp=time.time())
                client.send(error_message.encode())
        print(name, ':handles by', threading.current_thread())
        welcome = 'Welcome %s! If you ever want to quit, type quit to exit.'
% name
        # Use to generate response in HTTP format.
        print(Helper().encodehttprequest(messsage=welcome,
timestamp=time.time()))
        encoded_message =
Helper().encodehttprequest(messsage=welcome, timestamp=time.time())
        client.send(encoded_message.encode())
        msg = "%s has joined the chat!" % name
        broadcast(msg, name)
        clients[client] = name
        while True:
            # used to receive message from client.
            msg = client.recv(buffer_size).decode()
            msg_after_split = msg.split('\r\n\r\n')
```

```
            body = json.loads(msg_after_split[1])
            #print("body receive at the server is: ", body)
            msg = body['message']
            if msg != "quit":
                broadcast(msg, name, name + ": ")
            else:
                encoded_message =
Helper().encodehttprequest(messsage='quit', timestamp=time.time())
                client.send(encoded_message.encode())
                client.close()
                del clients[client]
                broadcast("%s has left the chat." % name,name)
                break

    except OSError:
        pass
        # if someone left the chat it goes here.
        closed_connection_msg = ' has closed connection forcefully.'
        print(name, closed_connection_msg)
        file = open('log.txt', '+a')
        file.write(json.dumps({'name': name, 'message':
closed_connection_msg}))
        file.write('\n')
        file.close()


def broadcast(msg, name, prefix=""):  # prefix is for name identification.
    #Broadcasts a message to all the clients.
    # write all the log to log.txt file.
    file = open('log.txt', '+a')
    file.write(json.dumps({'client_name': name, 'message': msg}))
    file.write('\n')
    file.close()
    for sock in clients:
        encoded_message = Helper().encodehttprequest(messsage=prefix
+ ' ({time}) - ' + msg, timestamp=time.time())
```

```
        #print(clients)
        #print("\n\nServer Broadcasted: ", encoded_message)
        sock.send(encoded_message.encode())


clients = {}
addresses = {}
# give host address of the server
host = '127.0.0.1'
# give port number of the server
port = 9008
# assign buffer size to store the data
buffer_size = 1024

# create socket to listen to client.
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# bind host and port

server_socket.bind((host, port))
if __name__ == "__main__":
    server_socket.listen()
    print("Server is running.\nWaiting for connection...\n")
    restart_file = open('log.txt', 'r')
    ACCEPT_THREAD = Thread(target=accept_connections)
    ACCEPT_THREAD.start()
    print('Thread with ID {} has been created.'.format(os.getpid()))
    ACCEPT_THREAD.join()
    server_socket.close()
```

CLIENT SIDE:

```
import json
import sys
```

```
import tkinter
#from PIL import Image, ImageTk
from socket import AF_INET, socket, SOCK_STREAM
from threading import Thread
from tkinter import messagebox
from datetime import datetime
import pytz

class ClientHelper:
    # encode client message and send it.
    def encodemessage(self, request):

        body = {'message': request}
        body_string = json.dumps(body)
        method = 'POST'
        host = '35.246.29.33'
        url = '/chat'
        protocol = 'HTTP/1.1'
        user_agent = 'Chat Room'
        content_type = 'application/json'
        content_length = len(body)
        tz_NY = pytz.timezone('Asia/Kolkata')
        datetime_NY = datetime.now(tz_NY)
        date=datetime_NY.strftime("%H:%M")
        #date = datetime.strftime('%a , %d  %b %Y %H:%M:%S GMT')
        #utc_dt_aware = datetime.datetime.now(datetime.timezone.utc)

        header = method + url + protocol + "\r\n" + 'Host: ' + host + "\r\n" +
'User-Agent: ' + user_agent + "\r\n" \
                + 'Content-Type: ' + content_type + "\r\n" + 'Content-Length: ' +
str(content_length) + "\r\n" + \
                'Date: ' + date

        encodedMessage = header + "\r\n\r\n" + body_string

        return encodedMessage
```

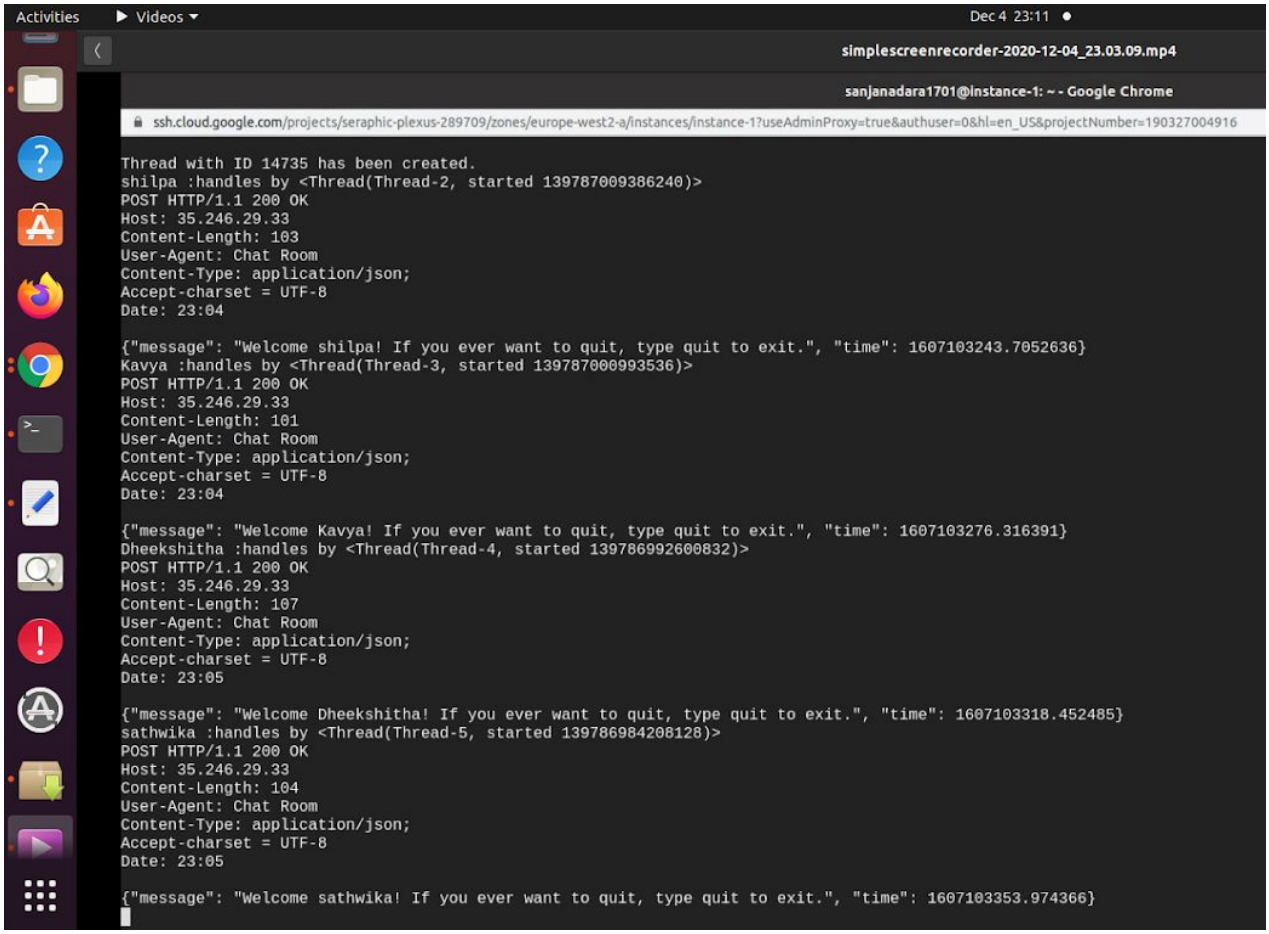
```
def receive():
    _last_time = 0
    while True:
        try:
            # receive message from server.
            msg = client_socket.recv(buffer_size).decode("utf8")
            # split the header and body.
            msg_after_split = msg.split('\r\n\r\n')
            body = json.loads(msg_after_split[1])
            if _last_time == 0:
                _last_time = body['time']
            time_passed = body['time'] - _last_time
            m, s = divmod(time_passed, 60)
            h, m = divmod(m, 60)
            tz_NY = pytz.timezone('Asia/Kolkata')
            datetime_NY = datetime.now(tz_NY)
            date=datetime_NY.strftime("%H:%M")
            body['message'].format(time=date)
            msg_list.insert(tkinter.END, body['message'].format(time=date))
            _last_time = body['time']
        except OSError:  # Possibly client has left the chat.
            sys.exit()


def send(event=None):
    msg = my_msg.get()
    my_msg.set("")  # Clears input field
    encoded_message = ClientHelper().encodemessage(msg)
    client_socket.send(encoded_message.encode())
    if msg == "quit":
        client_socket.close()
        top.destroy()
        top.quit()
```

```
def closing():
    # ask to close the application.
    if tkinter.messagebox.askokcancel("Quit", "Do you really wish to
quit?"):
        top.destroy()
        encoded_message = ClientHelper().encodemessage('has left the
chat.')
        client_socket.send(encoded_message.encode())
        client_socket.close()
    else:
        pass

print("Enter a title for your room: ");
titlename = input()
top = tkinter.Tk()

top.title(titlename)

messages_frame = tkinter.Frame(top)
my_msg = tkinter.StringVar()  # For the messages to be sent.
my_msg.set("")
scrollbar = tkinter.Scrollbar(messages_frame)  # To navigate through past
messages.

msg_list = tkinter.Listbox(messages_frame, height=15,
width=80,bg="black",fg="white",yscrollcommand=scrollbar.set)
scrollbar.pack(side=tkinter.RIGHT, fill=tkinter.Y)
msg_list.pack(side=tkinter.LEFT, fill=tkinter.BOTH)
msg_list.pack()
messages_frame.pack()

entry_field = tkinter.Entry(top, textvariable=my_msg)
entry_field.bind("<Return>", send)
entry_field.pack()
send_button = tkinter.Button(top, text="Send", command=send)
send_button.pack()
```

```
top.protocol("WM_DELETE_WINDOW", closing)

# Socket and port name is defined and thread starts from here.
host = '35.246.29.33'
port = 9002
address = (host, port)
buffer_size = 1024
client_socket = socket(AF_INET, SOCK_STREAM)
client_socket.connect(address)
receive_thread = Thread(target=receive)
receive_thread.start()
tkinter.mainloop()
```

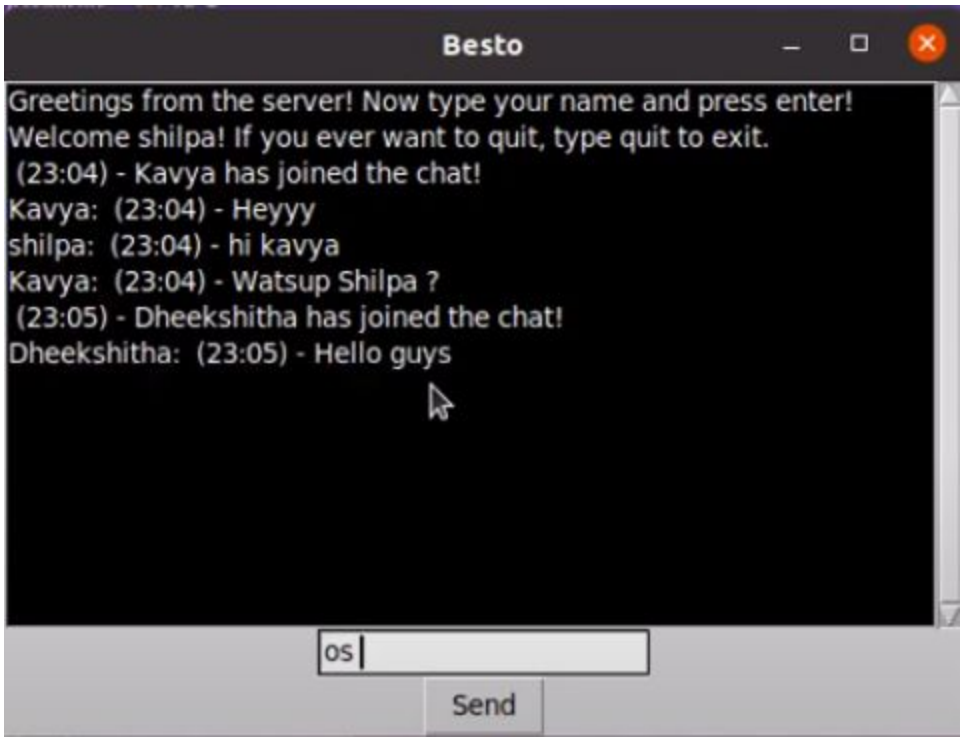
OUTPUT:

Server :

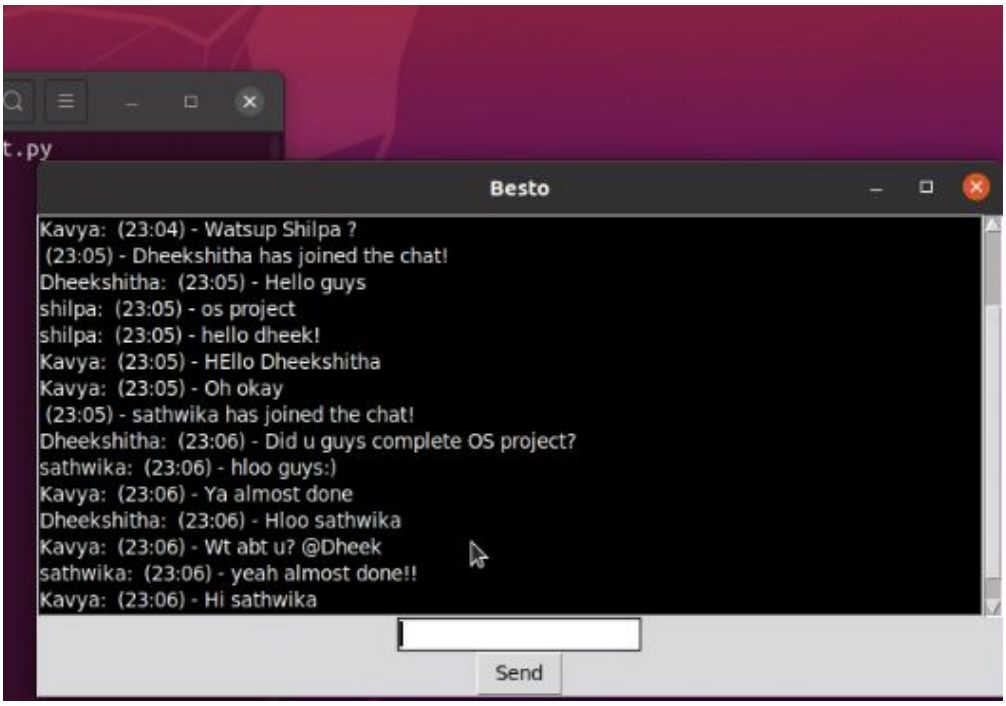


Clients:

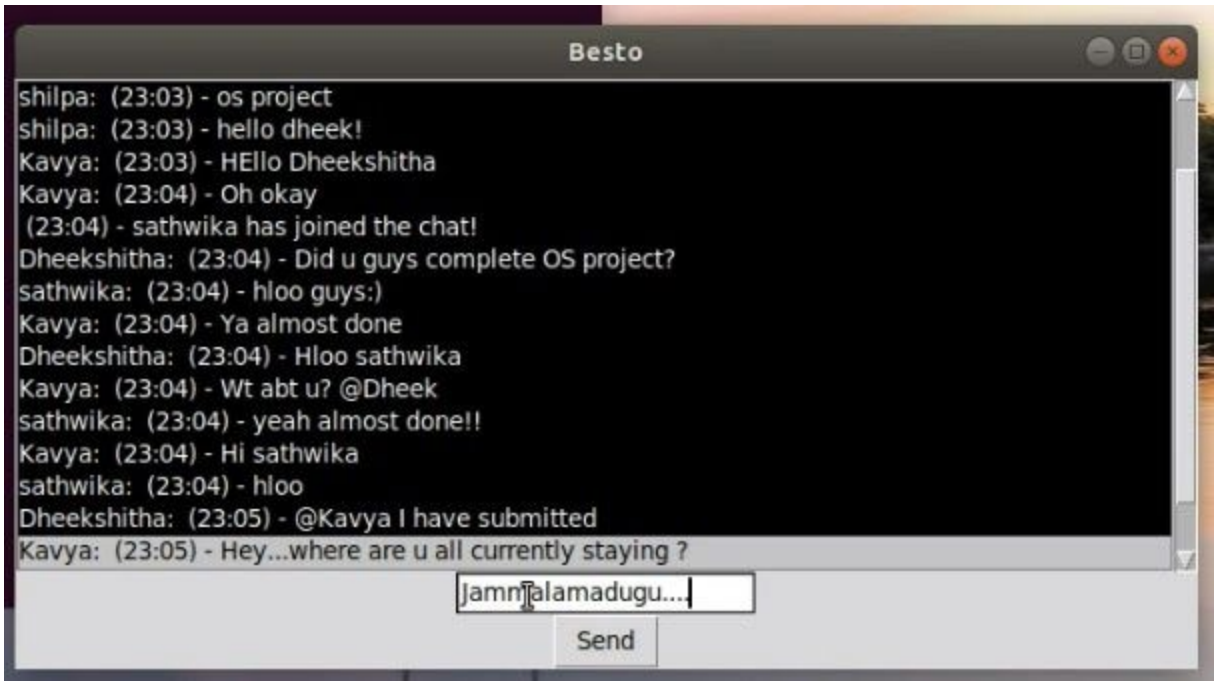
Silpa(Client-1):



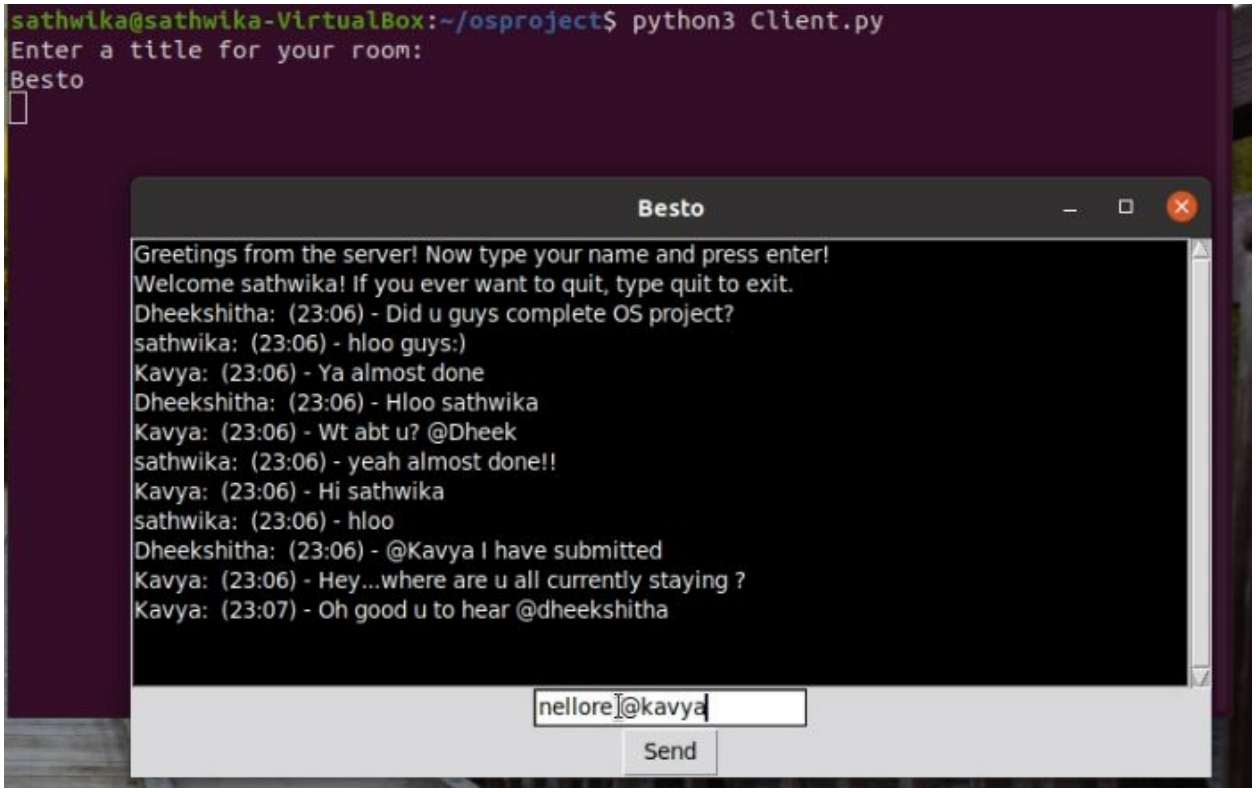
Kavya(Client-2):



Sai Dheekshitha(Client-3):



Sathwika(Client-4):



----------THE END----------