CSC3002F Networks Assignment 2018

Socket Programming Project: Report

Kyle(dplkyl002) Diya(sbrdiy001) Suzan(mbssuz001)

Implementation:

The multi-threaded client-server based chat application with media file transmission using TCP sockets, uses two java files (ChatServer.java and ChatClient.java). The application allows group and 1:1 private chat in real-time. The chat server accepts connection requests from the clients and delivers all messages from each client to other clients. The chat client's and server's interface uses the Ubuntu Linux terminal.

ChatClient.java:

This is the multi-threaded chat client which uses two threads.

One thread (thread 1):

- captures data from the standard input (user)
- checks if there is a media file to be transferred (for blocked, private and public messages)
- processes media file to be sent (uses byte array)
- sends text message / media file to chat server

The other thread (thread 2):

- checks for / creates a received folder for each client for media file transfer
- receives media file from chat server
- transfers media file to the particular client's received folder
- the client has the option of receiving or declining the media file (incorporates bandwidth constraint - media file transmission)
- receives text messages from the server and prints it to the screen (standard output)
- terminates the client on quit

ChatServer.java:

This is the multi-threaded chat server containing two classes.

The ChatServer class:

- is responsible for the overall control and coordination of communication between clients
- displays a log of each client's actions in the chatroom
- accepts connection requests from clients
- creates a new client thread every time a connection is established (i.e. a new connection request from a client is accepted)

- uses a separate thread for each client
- shows number of clients connected to the chat server

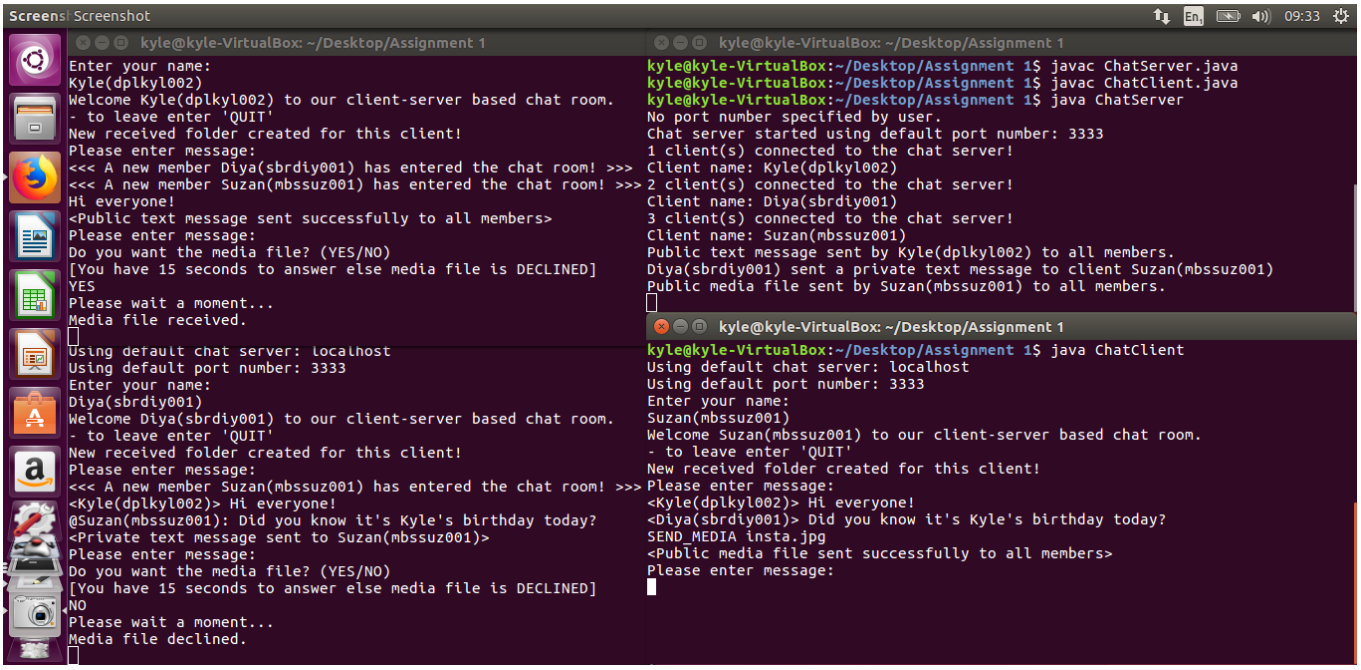
The clientThread class:

- thread for a particular client (handles individual clients in their respective threads)

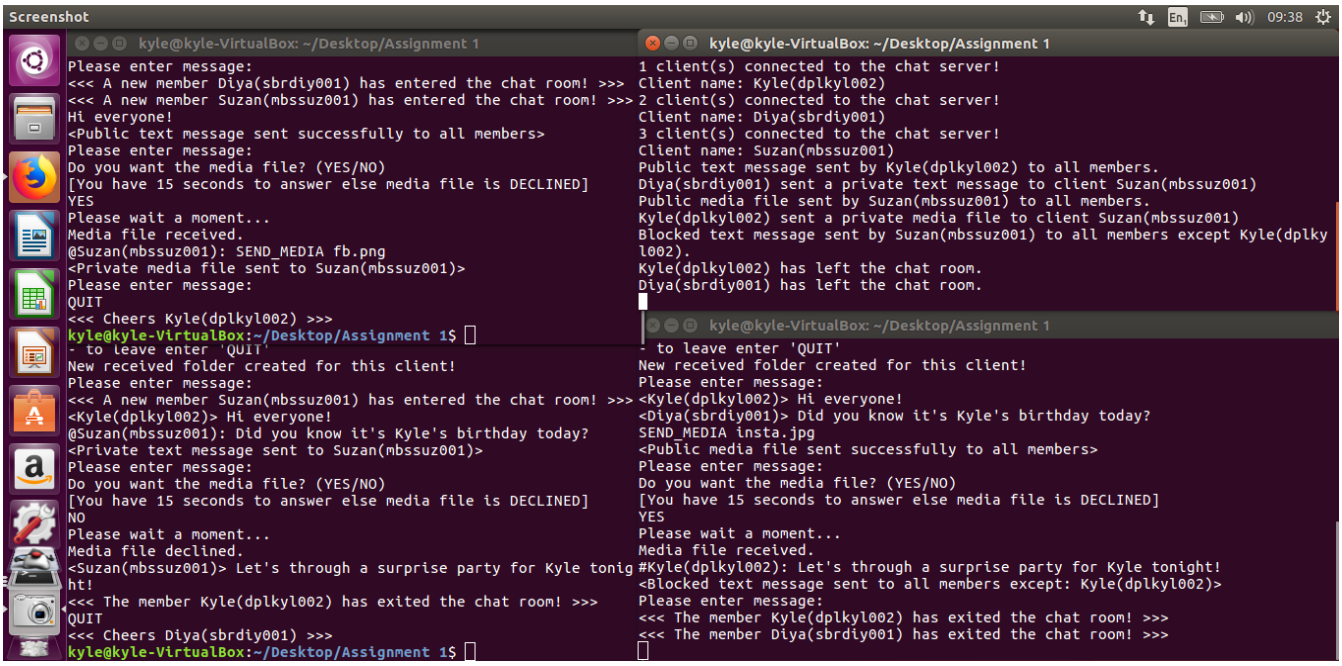
This thread:

- opens the input and the output streams for a specific client
- asks the client to enter their name
- sends a welcome message to the particular client
- notifies all the other clients that a new client has joined the chat room
- receives data from client and checks message type (blocked, private or public)
- transfers (blocked) text messages or media files to all clients present in the chat room, except a particular client
- transfers (private) text messages or media files to a particular client present in the chat room
- transfers (public) text messages or media files to all clients present in the chat room
- notifies the client that the blocked, private or public text message or media file has been sent successfully
- notifies all the clients present in the chat room that a client has left

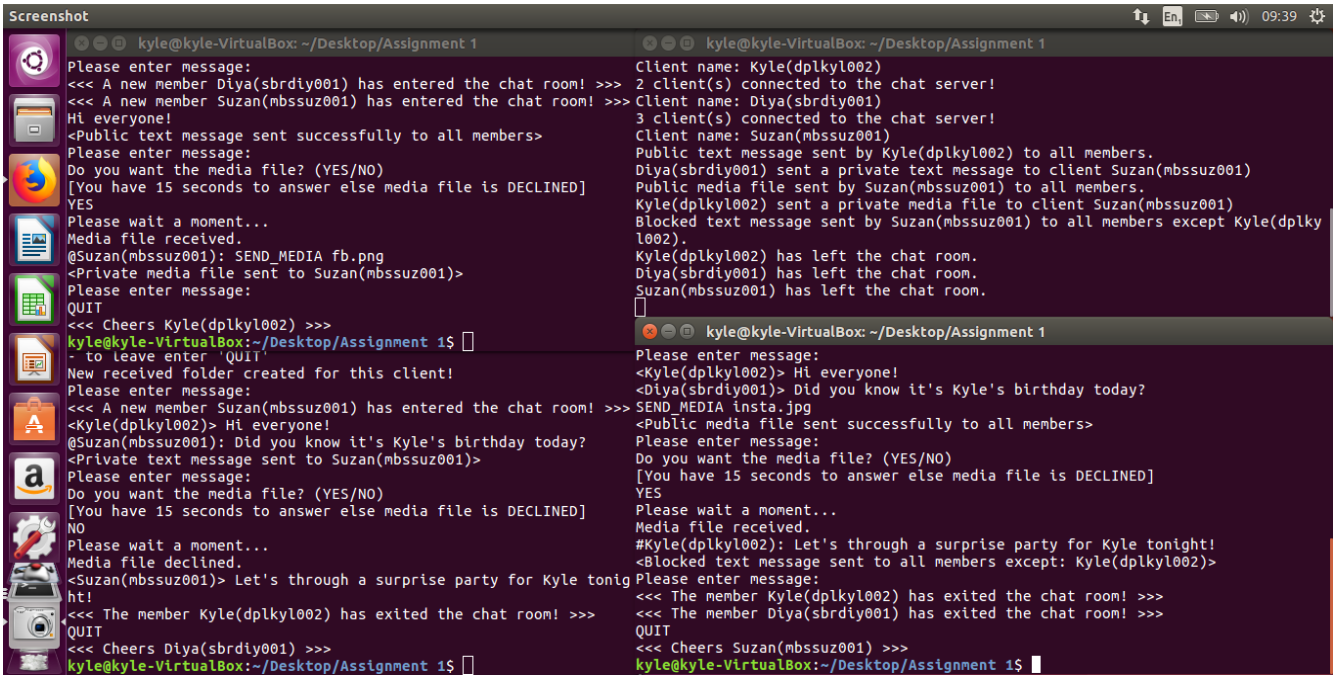
Screenshots of the application:



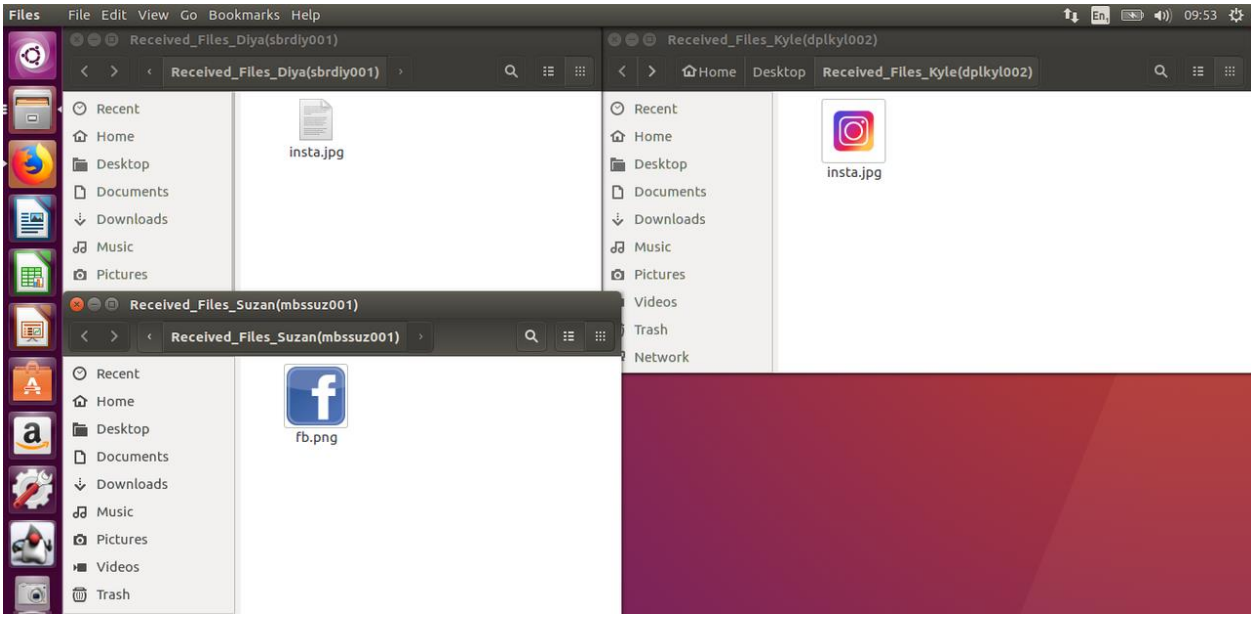
In the screenshot above, three clients (Kyle, Diya, Suzan – with student numbers included for each) joins the chatroom. Kyle sends a public text message and Diya sends a private text message to Suzan. Then Suzan sends a public media file to everyone in the chatroom. Kyle accepts the public media file, while Diya declined to accept the media file (limited bandwidth constraint).



In the screenshot above, Kyle sends a private media file to Suzan. Suzan accepts the media file. Susan then sends a blocked text message that is sent to all members except Kyle. Then Kyle and Diya leave the chatroom and Suzan is notified of their departure.



In the screenshot above, Suzan then leaves the chatroom and receives a message confirming her departure.



The above screenshot shows the media files in each clients' respective received folder as expected. As Diya had declined to receive the "insta.jpg" file, it is still displayed in her folder - but the file is empty (0 bytes).