CSC3002F Networks Assignment 2018

Socket Programming Project: Report

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

Overview:

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.

Two network constraints:

1. Speed of communication (e.g real-time interaction):

Protocol implementation:

The client-server based chat application uses *multi-threading* to enable clients to exchange messages in real-time.

ChatClient.java:

This is the multi-threaded chat client which uses two threads. Two threads are used in the implementation of the chat client as the chat client must communicate with the chat server and at the same time has to read messages from the standard input (user) to be sent to the chat server.

One thread (thread 1):

- captures data from the standard input (user)
- sends text messages to chat server

The other thread (thread 2):

- 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. Threads are also used in the implementation of the chat server.

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 to all clients present in the chat room, except a particular client
- transfers (private) text messages to a particular client present in the chat room
- transfers (public) text messages to all clients present in the chat room
- notifies the client that the blocked, private or public text message has been sent successfully
- notifies all the clients present in the chat room that a client has left

Protocol design:

The protocol design has to include specification for pattern of communication and data transmission, message structure, as well communications rules.

The pattern of communication:

The pattern of communication is client-server-based. The application allows clients to send public, private and blocked text messages. Public text messages can be sent to all clients present in the chat room. Private text messages can be sent to a particular client present in the chat room. Blocked text messages can be sent to all clients present in the chat room, except a particular client. Messages sent and received in the application is in text format.

Specification of protocol messages involves defining the types and structure of messages. Three types of messages can be defined; commands, data transfer, and control.

The three message types in the application (blocked, private or public) are data transfer messages.

Data transfer messages:

Message format / structure:

For sending public messages:

<your text message>

For sending private messages:

@<TargetClient>: <your text message>

For sending blocked messages:

#<TargetClient>: <your text message>

Command messages:

Message format / structure:

For quitting the application:

QUIT

2. Limited bandwidth

Protocol implementation:

Since one communication constraint in many developing countries is bandwidth, for media file transmissions, the recipient has the option of accepting or declining the receiving of the media file. The client-server based chat application uses a *file transfer protocol* to enable clients to exchange media file messages in real-time.

ChatClient.java:

This is the multi-threaded chat client which uses the existing two threads to implement / incorporate file transfer functionality for media file transmission.

One thread (thread 1):

- 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 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)

ChatServer.java:

Threads are also used here in the implementation of the file transfer functionality for media file transmission.

The clientThread class:

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

This thread:

- transfers (blocked) media files to all clients present in the chat room, except a particular client
- transfers (private) media files to a particular client present in the chat room
- transfers (public) media files to all clients present in the chat room

Protocol design:

The protocol design has to include specification for pattern of communication and data transmission, message structure, as well communications rules.

The pattern of communication:

The pattern of communication is client-server-based. The application allows clients to send public, private and blocked media file messages. Public media file messages can be sent to all clients present in the chat room. Private media file messages can be sent to a particular client present in the chat room. Blocked media file messages can be sent to all clients present in the chat room, except a particular client.

Specification of protocol messages involves defining the types and structure of messages. Three types of messages can be defined; commands, data transfer, and control.

The three message types in the application (blocked, private or public) for media files are data transfer messages.

Data transfer messages:

Message format / structure:

For sending public messages:

SEND_MEDIA <your media file>

For sending private messages:

@<TargetClient>: SEND_MEDIA <your media file>

For sending blocked messages:

#<TargetClient>: SEND_MEDIA <your media file>

Control messages:

Message format / structure:

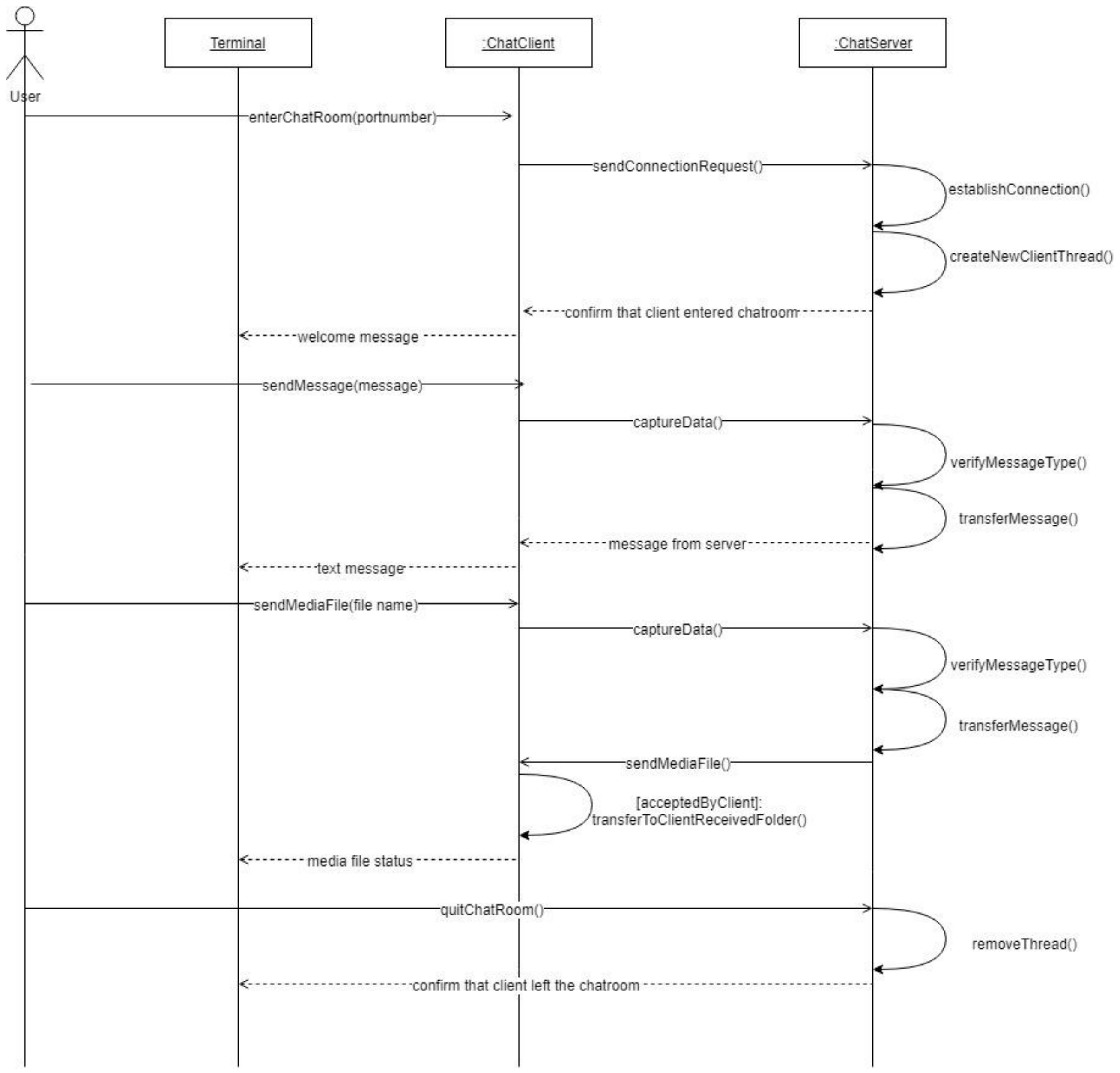
For accepting the media file:

YES

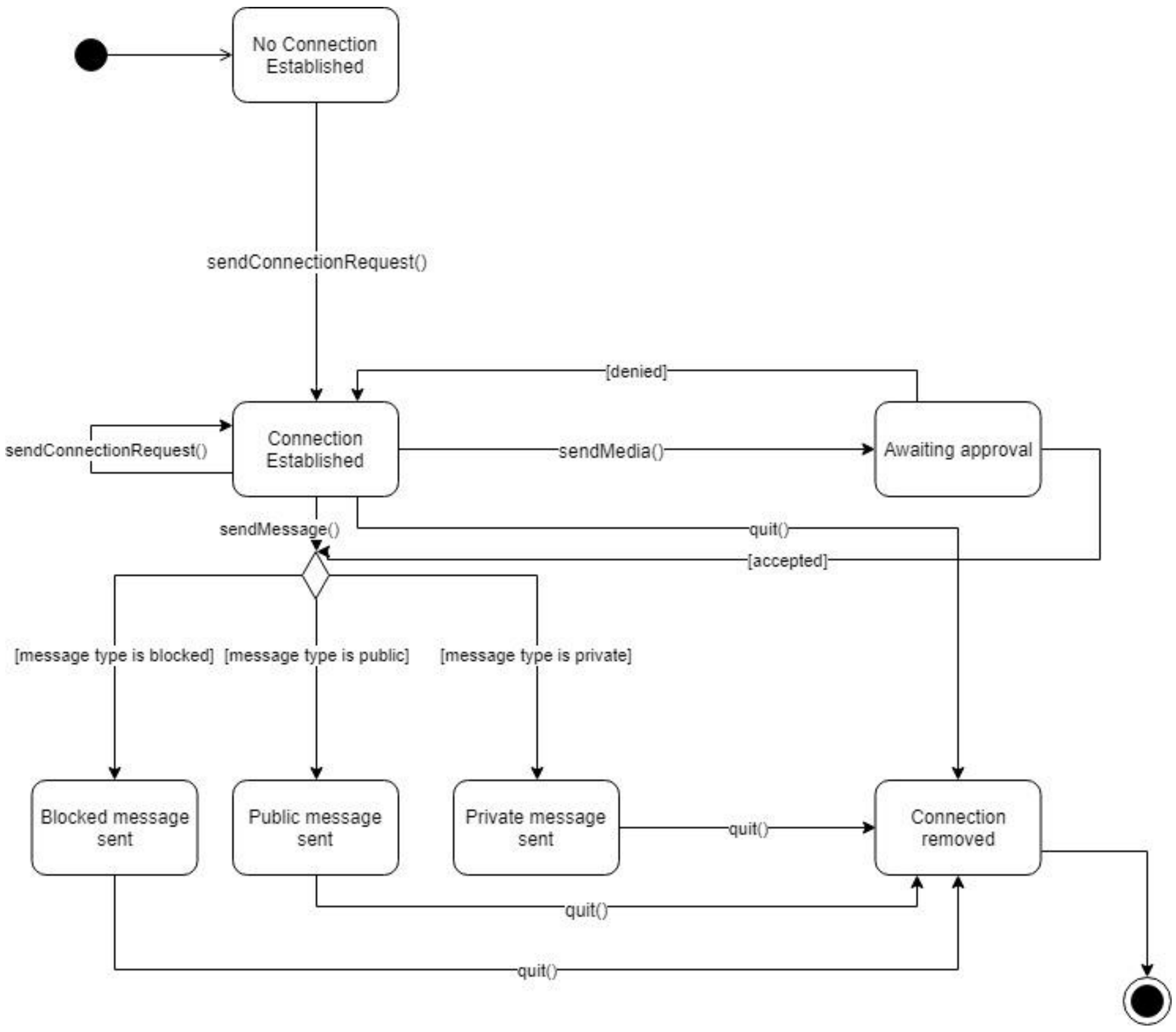
For declining the media file:

NO

Sequence diagram:



State Diagram:


No Connection Established
sendConnectionRequest()
[denied]
sendConnectionRequest()
Connection Established
sendMedia()
Awaiting approval
sendMessage()
quit()
[accepted]
[message type is blocked]
[message type is public]
[message type is private]
Blocked message sent
Public message sent
Private message sent
quit()
Connection removed
quit()
quit()