Chat Server Milestone 3

November 30th, 2012

Media Engineering and Technology, Computer Science Department
Team Members: Mohammad Khalid Ibrahim Abdulkhaliq, 22-2768, T-07
Mohammad Abdelfattah, 22-3781, T-07
Omar Mahmoud, 22-1534, T-07

Contents according to their order:

- **Classes & Description:**

1- Main
2- Server
3- ThreadSocket
4- Message
5- Chatroom
6- Client

Classes & Description:

1- Main

Main is a class whose main purpose is to initialize a server that listens to connections at all times.

2- Server

Server is one end of the server socket connection to which clients can connect. It has a name to identify it, a flag to identify its state(running or not) and hash tables for (threadsockets, chatrooms, TCP transferred files, UDP transferred files) respectively. It extends **ServerGUI** which has the respective gui of the server.

listen(): method which sets it running and sets the threadsocket into a state where it accepts incoming connections. Other methods for the server are

getNAME() setName(String) which gets the name and sets the name of the server respectively.
removeClient(ThreadSocket): removes a certain client from the hashtable of the threadsocket.
getCLIENTS() setCLIENTS(): gets and sets the hashtable of the threadsocket respectively.
getClientNames(): gets a list of the clients in the threadsocket hashtable.
getChatRoomList(): gets a list of all the rooms in the chatroom hashtable.
close(): closes the server socket if the server is running.
checkName(String): checks if the name of the client exists in the threadsocket hashtable.
getROOMS() setROOMS(Hashtable): gets & sets the chatrooms connected to the server respectively.
getByName(String): gets a specific client from the threadsocket hashtable.
getTCP_FILES() setTCP_FILES(hashtable): gets and sets the TCP files in the hashtable respectively.
getUDO_FILES() setUDP_FILES(hashtable): gets & sets the UDP files in the hashtable respectively.

3- ThreadSocket

ThreadSocket is a thread class that constructs the socket connection between the server and the client when it is run, handles the messages transmitted over the connection and makes sure the socket connection is terminated when the client is disconnected. Its main method is

run(): initialize a message to be transmitted over the connection and handles it.
write(): writes an Object Output Stream into a message.
close(): terminates the socket connection by closing the sockets at the endpoints.
isClosed(): checks if the server socket connection has been closed.
getLobby() setLobby(Server): gets & sets the server to be connected to respectively.
getCLIENT_NAME() setCLIENT_NAME(String): gets & sets the client connected to the server res.
isAdmin() setAdmin(boolean): checks if the client connected is the admin & sets him to admin respect.
removeChatRooms(String): removes the client from all the chat rooms to which he is connected.
handleMessage(Message) : Main method for handling messages received from client

- **Message Type 0** name message checks if name already in client list hashtable on server if exists
- Send error message to client requesting change of name.
- **Message Type 1** normal lobby message once it reaches a thread socket it is broadcasted to all clients
- Currently on server by calling **send(Message**) which in short converts the hashtable of clients to an **object []** then iterates over each threadSocket and calls it's respective **write** method on the same message.
- **Message Type 2** special request message for lobby action i.e not related to chat rooms eg. Get Client List
- ,getChatroom list, create public/private chatroom,etc.

- **Message Type 3** chatRoom message here the **M.getDst()** gives the name of chatroom **hence** message is routed to only members of the particular chatroom.
- **Message Type 4** ChatRoom special request message here it is concerned with special requests regarding a particular chatroom eg. Invite,kick-off,etc. it in general checks if user of action has permission to use it then acts accordingly.
- **Message Type 5** Error message sent to clients with info on error.
- **Message Type 6** Messages related to upload , download requests may contain the actual files.

4- Message

Message is a class that defines the object being sent over the server socket connection. Its methods are
getTYPE() setTYPE(int): gets & sets the type of the message respectively.
getMSG() setMSG(String): gets & sets the content sent in the message respectively.
getSRC() setSRC(String): gets & sets the source of the message respectively.
getDST() setDST(String): gets & sets the destination of the message respectively.
getInfo() setInfo(String): gets & sets the information attached with the message.
getOMSG() setOMSG(): gets & sets the object(files in this case) being transmitted respectively.

5- **ChatRoom**

Chatroom is a collection of clients connected together by an arraylist, it has a flag that identifies it as private or not (as in connected to by a request or not), it has a main client who is the administrator of the chat room and has a name to identify it. Its methods are
getADMIN() setADMIN(ThreadSocket): gets & sets the admin of the chat room respectively.
getMEMBERS() setMEMBERS(ArrayList): gets & sets the clients connected over the chat room res.
addMember(ThreadSocket): adds another client to the chat room.
removeMember(): removes a client from the chat room.
search(String): returns a client if he is found in this chat room.
getMemberList(): returns a list of all the names of the clients in this chat room.
getName() setName(String): gets & sets the name of this chat room respectively.
Send(Message) Iterates over members of the chatroom and sends messages through every threadSocket.

6- Client

Client is the other end of the server socket connection, it has a name and ID to identify the client, a server socket connection to which the client it connected, a UDP connection, a port no. to be able to connect to the server using the server's IP address, it has an object output stream and an object input stream for the transmission of data, it has a flag called connected that shows whether the client is connected to the server or not, it has a room count that shows the no. of rooms in which the client is a member, it can send files using file path and file name from the JFileChooser and can determine the send time and receive time to and from the server. It has the following methods
getNAME() setNAME(Stirng): get & set the name of the client respectively.
getCLIENT_ID() setCLIENT_ID(int): get & set the ID of the client respectively.
read(): read the message sent over the server socket connection.
write(): write a message to be sent over the server socket connection.
close(): disconnect the client from the server and terminates the socket connection.
makeConnection():
actionPerformed():

getPORT_NO() setPORT_NO(int): get & set the port no. through which the server socket connection is made respectively.
getSERVER_IP() setSERVER_IP(String): get & set the server IP address to which the server socket connection is made.
run():pretty much handles the messages sent from server and interprets them in through the eyes of the client but the messages handled are the same with the addition of error messages which are just printed
copyFile():
search(String): returns the room in which the client is a member.

7- UDPThread

The Ultimate Hack in this code in my opinion the Udp Thread is special that it can act as a Udp server For both threadSocket and Client without divulging private information i.e the object polymorphs according the constructor that is used as indicated by char mode (client or Server) and handles the file differently depending on where it is used hence if it is a server UDP thread will receive the packet and store it in the Server or if it is a client UDP thread will prompt the user to save the file in his directory and print the time taken to download the file from server. Neat!!.

PROTOCOL :

Here not all client actions are explained in particular simple ones such as getting client list or chat room Lists.

UPLOAD->TCP :

1.Choose file on pc and convert it to **File** after getting receiver name send file using overloaded write method that takes object as a parameter plus other parameters such as src,dst,etc. using object output stream.
2.When file reaches server/threadSocket it places the file in a hashtable of files giving it the receiver name as key.
3.Server sends the file upload request to receiving client informing client of sender name and file name.
4.If receiver chooses to accept file, Server retrieves the file using accept message src string and sends the file same way as 1.
5.Once file reaches rClient user can choose a directory to save the file and then the file is copied from java temp memory to that directory using **copyFile**

UPLOAD->UDP:

Due to complete reliance on TCP protocol switching to UDP protocol proved to be a bit hard, Therefore I had to intergrate certain aspects of above mentioned protocol. In short :

After choosing a file and message is sent to server telling it to prepare to receive a datagram packet containg a byte[] of some size, size is sent along with message after server prepares a dedicated UDP thread to receive file it pings the client with an empty type 6 message telling it it is okay to start sending At this point the file is encoded to a byte [] encapsulated in a datagram packet and sent to server.

When packet is recived at server like TCP(step2) it is placed in a hashtable of byte[] then TCP(step3)

If rClient chooses to accept it first sends an accept message then the server sends a special message type 6 with size of file and then client prepares a polymorphed UDPThread and sends a ping to server telling it is okay to send. Then after TCP(step5) instead of copying files the packet data is decoded to a file in user specified directory.

CREATECHATROOM:
Very small differences between creation of public and private chatrooms hence I will explain only one

1.Create Chat Room request sent from client with unique chat room name attached as info.
2.Server processes request by creating a new C**hat Room instance** with creator as admin and only member and places in chat Room hash table with the aforementioned key.
3.Server sends the name of the chat Room to creator and client commences to create a new Instance of **room** and adds it to GUI tabs.

NOTE : What is Room ?

ROOM :
Room is the interpretation of a chat Room abstract object on client side it is represented as a JPanel With **Client** User as an attribute hence it is a helper class to Client giving users the ability to invite, k ick-off ,mute,etc. to admins

END OF NOTE

JOIN :

Somewhat Similar to invite in Concept hence I will only explain Join simply when join is pressed user Sends a join request holding name of chatroom he would like to join as info and then server adds this client to the chatroom member list and replies back with a join accept message holding chatroom name client commences in creating a room tab with same name and hence from here on out any message sent with the chatroom name will reach this new client's tab.

MUTE : if the admin chooses a person to mute if the person is already muted a variable in the user.roomlist is set back to unmute and if the user was unmute the variable will be set to mute.

Observatory Analysis and Bugs :

Here in our implementation every things is done through dedicated message object and Object Streams which in a sense provided a Modular design especially in extending the application or even the user interface simply by providing in the engine a new message Type or a new message class.

Furthremore, Our handling of errors is also done through messages which improves on overall user experience.

Disadvantages : Unreliable protection of private chatrooms as if a user somehow guesses the chatroom name which should not be too hard considering the naming scheme of the rooms can gain quick entry to a private chat room without being invited.

Does not reliably handle all user invalid actions such as inviting himself to a chatroom or sending a file to himself,etc i.e all himself actions !.

Closing of server gui may sometimes, very rarely lead to socket exception error.

Also, File time of send and receive calculations are inconsistent i.e done differently at different parts.

Lastly, when a user downloads a file he has to choose any file within a directory to save his file in that directory due to problems with making JFileChooser choose only directories hence a problem may occur if the directory is empty hence user has to create any file then save using our application.

REFRENCES:

projectMilestone 1 references.
(just the GUI look not the actual GUI code)

Miscellaneous information gathered from net all code pretty much written from scratch except the copy file part in client save file to dir but the methods user were self explanatory and there are different schemes I personally used in the code to implement File to file conversions and copying.