CSC 469 - Assignment 3

Zeeshan Qureshi Abhinav Gupta

5 Apr 2013

Features

We've implemented all of the requirements of the assignment:

- Server join and quit
- Chat room listing, switching and creation
- Location server support
- Failure detection and fault tolerance

Overview

We proceeded with the given project structure and refactored some methods to be more general. When the client initially starts up, it sets up a TCP connection to the location server and gets the list of chat servers. It then creates the receiver which binds to an available UDP port and returns the port number to the client. The client then sends the port to the server in the registration request and if it gets a success message then proceeds to user input. If there is any name lookup, network error or registration fails then the client quits with a message.

Now, whenever the user enters a command, the client parses it and either sends a command to the server or a chat message. On receiving a quit command the client sends a quit request to the server and then to the receiver, it waits for the receiver to quit and then closes all open sockets and quits.

The receiver displays all messages in its window as soon as it receives them on its socket. It also listens for messages from the client on its message queue and if there's a quit message, then it unbinds from port and quits.

NOTE: Printing the code and submitting it on paper seemed quite wasteful so we decided not to do it. If you really want it then we can submit it later.

Fault Tolerance

We implemented failure detection by periodically sending a heartbeat to the server. Instead of using threads for this, we silently timeout on user input, and send a heartbeat and then listen for input again. Since the terminal input is line buffered, the user won't notice this. This removes the extra complexity and overhead of maintaining threads.

If there is an error sending a heartbeat to the server, the client goes through the initialization process all over again until it establishes a new connection. If it can't establish a connection it keeps on retrying indefinitely with a 5 second wait between each retry.

Since we send a heartbeat every 5 seconds and the payload is very small, this solution is scalable to a very large number of clients without stressing the server.

Program Design

We created a *send_control_message* function that takes the data (byte array) that the user wants to send and packages it into a packet adding the control header with the correct length and member id. It then sets up a TCP connection to the server and sends the packet and reads the reply. It reports any error back to the calling function or writes the reply to an output buffer if successful.

We also created a *handle_room_request* function that generalizes control requests for a room and takes in function pointers to execute on success or failure. This removes a lot of code duplication from the request handling methods and makes the code cleaner and easier to maintain.