Project 2 – A Simple Chat Client and Chat Server

CS 555 Fall 2013

Due Date: **Oct 31, 2013, 7:20 pm**

1 Description

A *chat server* is an application program that enables conversation between multiple *chat clients*. It maintains a set of chat *rooms/groups*. Chat clients connect to the chat server and make a request to join the group. The chat server may reject the request if the number of clients already in the group exceed the maximum allowed in that group. If the request is accepted, that client is made a member of the group.

All the clients that are members of a group receive all the messages transmitted by any member of the group. This group communication is facilitated by the chat server. A chat client reads the text typed by the user and sends it to the chat server. It is the responsibility of the chat server to relay that text to all the other members of the corresponding group. Chat clients receive the text relayed by the server and display it on the screen.

2 Implementation

You need to write two programs: *chatclient* and *chatserver*. The BSD sockets are used to communicate between these two programs. The functionality of each of these programs is described below.

2.1 Chat Server

Here are some of the actions that need to be performed by a chat server.

- Initialize a table from a configuration file that contains the list of chat groups and their maximum capacity.
- Create a socket and bind it to a port. This is the port that is used by the clients to connect to the server.
- The address of a server should be advertised such that clients know its whereabouts. This is accomplished by having the server create a symbolic link ".chatport" (in user's home directory) that contains the IP address of the machine on which the server is running and the port number assigned to it.
- Now server's job is to wait for requests from clients and service them. These requests can be of three types: *connect* requests from new clients, *action* requests on established connections from existing clients and *disconnect* requests from existing clients.

connect Server accepts the connection and adds it to the list of established connections on which it has to listen for action requests.

disconnect Server closes the connection and removes it from the list of established connections.

action Any message on an established connection is essentially an action request. There are four types of action requests.

- `LIST-GROUPS`: Send the list of groups, their current occupancies and maximum capacities to the client.
- `JOIN-GROUP` *group-name member-name* : Check the current occupancy in the requested group. If maximum capacity already reached, reject the request by sending `JOIN-REJECTED` reply to the client. Otherwise, add this client to the specified group and send `JOIN-ACCEPTED` reply to the client.
- `LEAVE-GROUP` : Remove the client from the group it currently belongs to.
- `USER-TEXT` *user-text* : Identify the group to which this client belongs to and the name of the client. Then insert the name of the client into the message such that it reads "`USER-TEXT` *sender-name user-text*" and forward the message to every member of the group except this client.

- The server stays alive and continues to service clients' requests, till it is explicitly killed by the user. Before it exits, it has to perform some clean up tasks such as removing the symbolic link ".chatport".

2.2 Chat Client

Here is the action sequence of a chat client.

1. It has to first locate the server. It reads the symbolic link ".chatport" (in user's home directory) to figure out the whereabouts of the chat server.

2. Create a socket and connect to the server by specifying the server's IP address and the port.

3. Send `LIST-GROUPS` request and receive the list of groups and their maximum capacities and current occupancies. Display the group list and let the user choose a group. Also provide the user with a `QUIT` option to exit the program.

   - If user selects a group, send a `JOIN-GROUP` request along with the user's chosen group and user's name, and wait for the response from the server.
     - If server responds with `JOIN-REJECTED`, go back to step 3.
     - If server responds with `JOIN-ACCEPTED`, start reading the user input from the keyboard and the relayed messages from the server

    - If user types in "/end" in a line by itself, send `LEAVE-GROUP` message to the server and go to step 3.
    - Any other input from user is sent in a a `USER-TEXT` message to the server along with the user's input.
    - Read `USER-TEXT` message from the server and extract the *sender-name* and *user-text*. Display the *sender-name* and the *user-text* on the screen.
- The client simply exits if user chooses `QUIT` option. The descriptors are closed automatically and the server interprets it as a disconnect and removes this connection from the list of established connections.

2.3 Summary of Messages

2.3.1 To Server

- `LIST-GROUPS`
- `JOIN-GROUP` *group-name member-name*
- `LEAVE-GROUP`
- `USER-TEXT` *user-text*

2.3.2 From Server

- `LIST-GROUPS` *group-list*
- `JOIN-REJECTED`
- `JOIN-ACCEPTED`
- `USER-TEXT` *sender-name user-text*