Project #1A – Chat Client

Due Oct 22 @ 3:50 pm

EE122: Introduction to Communication Networks (Fall 2008)

Department of Electrical Engineering and Computer Sciences
College of Engineering
University of California, Berkeley

Project Goal

Create a Chat application consisting of a *chat client* and a *chat server*. The client and server must be written in C or C++ and must use sockets.

Project Timeline

Project 1 consists of two parts.

1. The first part (part A) is to implement the chat client. It is due on **Oct. 6**. We will provide you a server that can be reached at: IP 128.32.48.187, port 10000. You can use this server to test your client. In addition, we will provide you the binary for the server that you can execute on instructional UNIX machines. *Your overarching goal for this part is to make your client work with our server!*
2. The entire project is due on **Oct. 20**. It includes both the client and server code. The specification for the server will be provided soon.

Section I: Introduction

The chat application allows multiple chat clients to connect to a chat server. Connected clients can: list the members already logged in, log in with a username, exchange messages with other logged in users, and log out. The server has to accept and maintain connections to all the clients and relay chat messages between them.

Section II: Chat Client Overview

The chat client application has to do the following tasks:

1. **Connect** to a chat server. The client obtains the server IP address and the port it is listening on from command-line arguments. For example:

   >./client 128.32.48.187 10000

   Then, the client immediately attempts to connect to the server socket using TCP. If it fails, the client application prints an error message to the console and returns. To print the error message, the client must call function *perror* (defined in stdio.h) with the name of the socket function that failed in lowercase. For example:

perror("socket");

If, on the other hand, the connection succeeds, the client continues to step 2.

2. **Accept console text commands** from the user. The client must be able to handle the following commands. Parameters are enclosed in <>:
    a. **login <username>**
    b. **list**
    c. **sendto <username> <message>**
    d. **logout**
    e. **exit**

    Each line must begin with a command. The parameters (if any) are separated by a space. The commands and parameters are described in more detail below. In order to implement these commands correctly, the client will have to send and receive the following messages to/from the server:
    a. **Send messages** to the server. The client must be able to send the following messages to the server. These messages are described in detail in section 4 of the document:
        i. **Login Message**
        ii. **List Message**
        iii. **Sendto Message**
        iv. **Logout Message**
    b. **Accept incoming messages** from the server. The client must be able to properly react to the following messages. These messages are shown in section 5 of the document:
        i. **List Message**
        ii. **Response Message**
        iii. **Sendto Message**

Section III: Console Commands

We now present the console commands that a client must handle. *Note: whenever a client prints an error message, it should not perform any other action.*

- **login**: sends a login message to register with the server using the provided username.
    - **Parameter(s):** a string to be used as the username.
    - **Rules:**
        - The string may not contain any white space. Screen names are treated as case sensitive.
        - Any extra parameters after the username must cause the client to print an error. The error must also be printed if the username is more than 20 characters in length or if the username is not present. The following is the error:
          Bad username
    - **Examples:**
        - The following is an example of a valid command:
          login John

        - The following is an example of an invalid command:
          login John Doe

- **list:** sends a list message requesting the list of users already logged in.
    - **Parameter(s):** none
    - **Rules:**
        - Any extra parameters must cause the client to print the following error:
          Invalid command
    - **Examples:**
        - The following is an example of a valid command:
          list
        - The following is an example of an invalid command:
          list bob
- **sendto:** sends a sendto message to the server to communicate with another client.
    - **Parameters:** The user name of the destination client and the text to send.
    - **Rules:**
        - If either or both of the parameters are missing or if the username contains more than 20 characters, the following error message should be printed to the console:
          Bad username
        - If there are more than 65535 characters of text, the following error message should be printed to the console:
          Bad message
    - **Examples:**
        - The following is an example of a valid command:
          sendto John Hello, how are you?
        - The following is an example of an invalid command:
          sendto JohnHello
- **logout**: sends a logout message to the server.
    - **Parameters**: none.
    - **Rules:**
        - Any extra parameters must cause the client to print the error:
          Invalid command
        - The client should not close the connection with the server when executing this command. After executing this command, the client should still be able to execute the list and login commands.
    - **Examples:**
        - The following is an example of a valid command:
          logout
        - The following is an example of an invalid command:
          logout Joe
- **exit:** closes the client socket and exits the program.