CSc 360: Operating Systems (Summer 2006)

Assignment 3: Multi-thread Chat Server

Design due: June 30, 2006
Implementation due: July 13, 2006

1 Introduction

In this assignment, you will design and implement a simple, multi-thread chat server using socket and pthread API. Only system requirements are provided, and you have the freedom to create your design. Be creative, but you should be able to justify your own design.

2 Requirements

2.1 Server requirement

1. Run the chat server: `./chat_server_name [port_number]`
2. The default server port number: `5050`
3. The server supports a maximum of 10 concurrent chat clients.
4. The server creates one thread each to receive all messages from one client. Additional threads can be used when necessary. Each message is at most 1024 bytes.
5. The messages from one client are broadcast to all other clients unless otherwise specified. Every client should receive the broadcast messages of all other clients in the same order.
6. Client-server communication is supported by the `STREAM` socket service using TCP/IP.

2.2 Supported client commands

1. Login: Clients login to the chat server by connecting to the specified server IP address and port number, and clients will be identified by their IP address and port number initially unless otherwise specified. For testing purposes, you can use

   `telnet server_ip server_port`

   to login to the server. If successful, the client will receive a message

   `client_ip:client_port <`

   from the server to prompt input (i.e., so-called "input prompt").

2. Broadcast: A client can type the following command to send a message to all other clients.

   `/say message`

   The requesting client will receive an input prompt, and all other clients will receive a message

   `speaker_ip:speaker_port > message`

   from the server.

3. List: A client can type the following command to get the identity list of all other clients.

   `/who`

   Then, the requesting client will receive a message

   `client_1_ip:client_1_port`

   `client_2_ip:client_2_port`

   ...

   `client_N-1_ip:client_N-1_port`

   and an input prompt from the server, if there are in total N clients currently in the system.

4. Whisper: A client can type the following command to send a message to a particular client.

   `/listener_ip:listener_port message`

   The requesting client will receive an input prompt. The requested client will receive a message

   `speaker_ip:speaker_port >> message`

   from the server.

5. Logout: Clients can logout from the server by discontinuing the connection to the server.

6. Unrecognized commands are ignored by the server.

2.3 Bonus client commands

1. Name: A client can type the following command to set its new nick name.

   `/whoami my_new_nick_name`

   The requesting client will receive an input prompt, possibly with the new nick name.

   Be aware that the nick name should be checked by the server for uniqueness. If successful, the client will be identified by its nick name, rather than `ip:port`, throughout the system afterward, and the server still should be able to handle the whisper command properly.

2. Last: A client can type the following command to get the last message broadcast by the server

   `/last`

   and the following command to get the last n broadcast messages ($1 \leq n \leq 10$).

   `/last n`

   The requesting client will receive the requested message(s), followed by an input prompt.

3 Submission

IMPORTANT: Please include a README file in your submission, indicating your real name and student number. A gzipped tarball named `assign3.tar.gz` of your assignment should be submitted through `http://www.csc.uvic.ca/~submit/index.cgi`.

4 Marking

Design document [2]; Server code [10]; Bonus function [3]

5 Note

This assignment is to be done individually. All submitted work should be yours. If you have used something out there, even a small component in your implementation, you should give credits to references, and we can know your contribution accordingly.