Project #1B – Chat Server

Due Oct. 20 @ 11:59 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 the chat server for the chat application. The server must be written in C or C++ and must use sockets. It must run on the instructional machines.

Timeline
The source code for the server is due on Oct. 20. The specification of the client and the description of the checkpoint were provided in the previous document.

Section I: Introduction
The chat server is responsible for providing a mechanism for the clients to communicate. This includes allowing clients to login so they can be identified by username and forwarding messages from one client destined to another. This also includes listing the usernames of the clients that are currently logged in to facilitate communication.

Section II: Chat Server Overview
The chat server has to do the following tasks:

1. **Listen** for incoming client connections at a specified port. The server obtains the port number from a command-line argument. For example:

   ```
   ./server 10000
   ```

   Then the server immediately attempts to create the socket using TCP and tries to listen for connections to the specified port number. If it fails, the server prints an error message to the console and returns. To print the error message, the server 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 server is successful, it continues to step 2.

2. **Respond to messages** received by multiple clients. The server must be able to properly respond to the following messages:

   a. **Login Message**
b. List Message
c. Sendto Message
d. Logout Message

The structure of these messages and the actions the server must take when receiving them are provided in the following section. *Additionally, the server must be able to handle multiple clients simultaneously using select().*

### Section III: Responding to Messages

The following is the structure of the messages the server receives from the client and the actions the server must take. B is a shortcut for Byte and XB refers to a variable length field. Assume big endian number representation for these messages.

- **Login message:**
  - **Structure of message**
    - **Field values:**
      - MSG TYPE: Set to 0x01
      - UNAME: Login username
  - **Actions**
    - If UNAME contains whitespace, if it starts with a null byte, or if it is already taken by another user, send failure response.
    - If the client who sent this message is already logged in, also send failure response.
    - Otherwise, associate the username with the connection and send OK response
  - **Structure of response**
    - **Field values:**
      - MSG TYPE: Set to 0x05
      - RES TYPE: Set to 0x01 for OK, 0x02 for Malformed, 0x03 for failure.

- **List Message:**
  - **Structure of message**
    - **Field values:**
      - MSG TYPE: Set to 0x02
  - **Actions**
    - Fill response message with username of every logged in client.
**Structure of response**

- **Field values**
  - MSG TYPE: Set to 0x02
  - UN NUM: Set to number of logged in clients (this value is treated as unsigned)
  - UNAMES: The user names of the clients that are logged in. Each user name is padded to 20 bytes as specified in the previous document.

**Sendto Message:**

- **Structure of message**
  - **Field values:**
    - MSG TYPE: Set to 0x03.
    - UNAME: User name of client to whom to forward the message.
    - T LEN: Number of bytes in the text field (this value is treated as unsigned).
    - TEXT: Message sent by the client. Does not have to be null terminated.

- **Actions**
  - *In the following explanation, this message was received from client A and the target of the message is client B.*
  - If client A is not logged in or cannot find client B (who is specified with UNAME), send failure response to client A.
  - Otherwise, replace the contents of UNAME with client A’s UNAME and send the message to client B. If this fails, transmit a failure response to client A.
  - Otherwise send OK response to client A.

- **Structure of response**
  - **Field values:**
    - MSG TYPE: Set to 0x05
    - RES TYPE: Set to 0x01 for OK, 0x02 for Malformed, 0x03 for failure.

**Logout Message:**

- **Structure of message**
  - **Field values:**
• MSG TYPE: Set to 0x04
  o Actions
    ▪ If client is not logged in, send failure response. Otherwise send OK response.
  o Structure of response
    * Field values:
      • MSG TYPE: Set to 0x05
      • RES TYPE: Set to 0x01 for OK, 0x02 for Malformed, 0x03 for failure.
    • All Other Messages:
      o Actions
        ▪ Send response indicating that we have received malformed packet.
        ▪ Close connection with client.

Section IV: Other Requirements
• Your server should be robust to unexpected messages and should close the connection with the client that caused that message instead of crashing. Other clients should not be affected.
• There should not be any memory leaks
• The server should interact properly with the client binary that was been provided as part of the client test script.

Section V: Grade Breakdown
• 95% - Passing all test-cases that test the functionality of the server (including its ability to avoid crashes).
• 5% - Not having any memory leaks. Valgrind is a good tool for this.