**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:
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.
  o **Parameter(s)**: a string to be used as the username.
  o **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*
  o **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.
- **Parameters**: Takes no parameters
- **Rules**:
  - Any extra parameters must cause the client to print the error: *Invalid command*
- **Examples**:
  - The following is an example of a valid command: `exit`
  - The following is an example of an invalid command: `exit client`

*Protocol Overview*: For each message that the client sends to the server the server will send a message back. For a successful List Message the server will return a List Message back (see the messages specification for details). For all other messages or an unsuccessful List Message, the server will respond with a Response Message. The client never sends messages back in response to messages received from the server.

**Section IV: Messages Sent from Client to Server**

The following is the structure for messages the client sends to the server. In all of these diagrams, B is a shortcut for Byte and XB refers to a variable length field. *Also, the client must expect a response from the server whenever it sends a message and must block user input until it receives that message.*

- **Login message**:  
  
  ![Login message diagram](image)

  - **Field values**:
    - MSG TYPE: Set to 0x01.
    - UNAME: Login username.
  - **Rules**:
    - UNAME must be padded to twenty bytes (i.e. if Bob is the user name, it must be followed by 17 null bytes).
    - UNAME does not have to be null-terminated (i.e. can contain a user name with 20 characters).
    - UNAME cannot contain any white space.

- **List message**:  
  
  ![List message diagram](image)

  - **Field values**:
    - MSG TYPE: Set to 0x02.

- **Sendto message**:  
  
  ![Sendto message diagram](image)
Field values:
- MSG TYPE: Set to 0x03.
- UNAME: User name of client to whom we are sending the message (same rules as for the login messages apply).
- T LEN: Number of bytes in the text field (this value is treated as unsigned).
- TEXT: Message we are sending to the client. Does not have to be null terminated.

Logout message:
Field values:
- MSG TYPE: Set to 0x04

Section V: Messages Sent from Server to Client
The following is the structure for messages the client receives from the server. Again, in all of these diagrams, B is a shortcut for Byte and XB refers to a variable length field.

List message: sent in response to successful list messages received from the client.

Field values:
- MSG TYPE: Set to 0x02.
- UN NUM: Number of user names in this message (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 earlier in the document.

Rules:
- The list should be printed one user name after another, ending with the text “<end list>”. For example:

  Bob
  Joe
  <end list>

Response message: sent in response to all messages received from the client, except where list messages are appropriate.

Field values:
- MSG TYPE: Set to 0x05
- RES TYPE: Set to 0x01 for OK, 0x02 for malformed packet, 0x03 for failure. OK means the command was successful. An example of a malformed packet is one that does not have a valid MSG TYPE. An example of failure is when a client tries to log in with a taken username.

Rules:
If the server finds that a list message from the client has a problem (i.e. the server cannot provide the list of logged in users), it will send a response message, NOT a list message.

- If result is ok, the client must print the following message:
  *Success*
- If result is failure, the client must print the following message:
  *Failure*
- If the result is malformed packet, there is something wrong with your code.

- **Sendto message:** sent whenever another client wishes to communicate with us.

<table>
<thead>
<tr>
<th>Field values:</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG TYPE: Set to 0x03.</td>
</tr>
<tr>
<td>UNAME: User name of client who sent us the message (same rules as for the login messages apply).</td>
</tr>
<tr>
<td>T LEN: Number of bytes in the text field (this value is treated as unsigned).</td>
</tr>
<tr>
<td>TEXT: Message sent by the client. Does not have to be null terminated.</td>
</tr>
</tbody>
</table>

- **Rules:**
  - Client should be able to receive this message whenever it is logged in.
  - Cannot assume that the values in the UNAME or the TEXT fields are null-terminated.
  - Whenever this message is received, the client application has to print the message in the form “<username>:<text message>” to the console. For example if Bob sends “Hi how are you?” to John, then John’s client should print out:
    *Bob:Hi How are you?*

**Section VI: Useful Links**

- **Class Slides Links on Sockets:** [http://inst.eecs.berkeley.edu/~ee122/fa08/notes/03-Socketsx2.pdf](http://inst.eecs.berkeley.edu/~ee122/fa08/notes/03-Socketsx2.pdf)
- **Other Reference Links on Sockets:** [http://www.stanford.edu/class/cs244a/socket-links.html](http://www.stanford.edu/class/cs244a/socket-links.html)
- **UNIX Manual pages (man):** On UNIX type “man socket”, “man bind”, “man listen”, “man connect”, “man accept”, “man send”, “man recv” for details on socket related system calls. These are a great reference for the library functions and system calls, describing their arguments, return value, and error codes precisely.

**Section VII: Submission Description**

The submitted project needs to compile and run on the UNIX instructional machines.

You have to deliver source code (no executables) and a brief Readme explaining what you've done.