**Operating Systems**

**CS4348**

**Project #3: Network Communication Using Sockets**

**Due Date: Saturday, April 15, 2017**

## I. Project Organization

This project will utilize Sockets for communication between processes.

You should do the following pieces to complete your project. Each piece is explained below:

* Design 15 points
* Code 50 points
* Output 25 points
* Summary 10 points

# Design

The messages exchanged between client and server should be documented in a table, listing each message sent, the purpose of the message, and the data contained in the message.

# Code

Your code should be nicely formatted with plenty of comments. The code should be easy to read, properly indented, employ good naming standards, good structure, etc.

# Output

Output will be graded by running your program.

# Summary

The summary section should discuss your project experience. You should discuss how you approached the project, including anything difficult or interesting, what was learned, and the end result. This should be at least 1 page in length.

**Teams**

This project may be performed alone or in a team of 2 people. In the case of a team, the work should be performed collaboratively with equal participation or divided evenly. The summary should include discussion of what each team member did. Each team member should be prepared to explain the code if asked to do so.

## II. Project Description

**Language/Platform**

This project must target a Unix platform and execute properly on our csgrads1 server.

The project must be written in C, C++, or Java.

If using C or C++, you must use the Sockets API with stream sockets.

If using Java, you must use java.net.Socket and java.net.ServerSocket.

Your approach should be similar to the examples given in class.

For C/C++, you must only use read and write system calls to communicate over the socket.

For C/C++, you must account for possible partial messages as discussed in class. This means using a loop to read or write all expected bytes. You should not make a separate call for each byte.

For Java, you may use any mechanism Java provides for synchronization and mutual exclusion.

Any other approaches require instructor approval.

### Client/Server Demonstration

This project will utilize Sockets for communication between processes.

Your task is to write a client and a server.

The client and server will demonstrate a message posting system. The server will maintain messages posted by clients, which clients can retrieve and view.

The client will:

1. Accept a machine name and port number to connect to as command line arguments.
2. Connect to the server.
3. Prompt for and send the user’s name.
4. Present the following menu of choices to the user:
   1. Display the names of all known users.
   2. Display the names of all currently connected users.
   3. Send a text message to a particular user.
   4. Send a text message to all currently connected users.
   5. Send a text message to all known users.
   6. Get my messages.
   7. Exit.
5. Interact with the server to support the menu choices.
6. Ask the user for the next choice or exit.

The server will:

1. Accept a port number as a command line argument.
2. Accept connections from clients.
3. Create a new thread for each client.
4. Store messages sent to each user.
5. End by termination with control-C.

The server thread will:

1. Accept and process requests from the client.
2. Add the user’s name to the list of known users.
3. Provide mutual exclusion protection for the data structure that stores the messages.
4. Send only the minimal data needed to the client, not the menu or other UI text.

Other rules:

1. Each client transaction should interact with the server. Clients will not communicate directly with each other.
2. Configuration: your server should support multiple different clients at the same time, but should not allow the same user name to have more than one connection at the same time.
3. Authentication: assume the client has privileges to use the system—do not require a password.
4. Limits: you can assume a maximum of 100 known users, and a maximum of 10 messages each, where each message is at most 80 characters long.
5. Persistence: when the server exits the messages it is storing are lost. They will not be saved to a file. When a user gets their messages, those messages are removed from the server.
6. Users: a known user is any user who has connected during the server session, but may or may not be currently connected. Also, a message sent to an unknown user makes them known.
7. Errors: obvious errors should be caught and reported. For example, an invalid menu choice.
8. Output: your output should use the same wording and format as the sample output.

Sample output (assumes Joe has already connected):

Client output for user Sue:

>client cs1 2005

Connecting to cs1:2005

Enter your name: Sue

1. Display the names of all known users.

2. Display the names of all currently connected users.

3. Send a text message to a particular user.

4. Send a text message to all currently connected users.

5. Send a text message to all known users.

6. Get my messages.

7. Exit.

Enter your choice: 1

Known users:

1. Sue
2. Joe

1. Display the names of all known users.

2. Display the names of all currently connected users.

3. Send a text message to a particular user.

4. Send a text message to all currently connected users.

5. Send a text message to all known users.

6. Get my messages.

7. Exit.

Enter your choice: 3

Enter recipient's name: Joe

Enter a message: Hello Joe

Message posted to Joe

1. Display the names of all known users.

2. Display the names of all currently connected users.

3. Send a text message to a particular user.

4. Send a text message to all currently connected users.

5. Send a text message to all known users.

6. Get my messages.

7. Exit.

Enter your choice: 6

Your messages:

1. From Joe, 04/11/13 03:14 PM, Hello Sue
2. From Joe, 04/11/13 03:15 PM, What’s new?

1. Display the names of all known users.

2. Display the names of all currently connected users.

3. Send a text message to a particular user.

4. Send a text message to all currently connected users.

5. Send a text message to all known users.

6. Get my messages.

7. Exit.

Enter your choice: 7

Server sample output:

>server 2005

Server is running on cslinux1.utdallas.edu:2005

…

04/11/13, 3:10 PM, Connection by known user Sue.

04/11/13, 3:12 PM, Connection by unknown user Joe.

04/11/13, 3:13 PM, Sue displays all known users.

04/11/13, 3:13 PM, Sue posts a message for Joe.

04/11/13, 3:13 PM, Joe displays all connected users.

04/11/13, 3:14 PM, Joe posts a message for Sue.

04/11/13, 3:15 PM, Joe posts a message for Sue.

04/11/13, 3:17 PM, Sue gets messages.

04/11/13, 3:20 PM, Sue exits

04/11/13, 3:21 PM, Joe exits

…

## III. Project Guidelines

### Submitting

Submit your project on eLearning. Include in your submission the following files:

1. readme.txt. A readme file describing how to compile and run your project
2. team\_members.txt list of team members, only required for teams
3. design.doc Your message list
4. summary.doc A Word document for the summary
5. Your source files

For a team submission, only one person should submit.

### Academic Honesty

All work must be your own. If cheating is suspected, you will be referred to the Judicial Affairs Office for further discussion. Copying may be detected in a number of ways, including by software which compares your code with all other students’ source code, by comparison with code on the Internet, or by a visual inspection of your source code.

### Resources

The web has many articles on Sockets. There are also books available on Sockets. The course website also contains an example of Socket source code.