CSEE 4119: Computer Networks, Spring 2014

Programming Assignment 1: Socket Programming

Due Tuesday, March 4th 11:55 pm

P. Prakash, W. An, P. Nirantar, R. Yu, X. Zhu (TAs), A. Chaintreau (instructor)

Academic Honesty Policy

You are permitted and encouraged to help each other through Piazza's web board. This only means that you can discuss and understand concepts learnt in class. However, you may NOT share source code or hardcopies of source code. Refrain from sharing any material that could cause your source code to APPEAR TO BE similar to another student's source code enrolled in this or previous years. Refrain from getting any code off the internet. Cheating will be dealt with severely. Cheaters will be penalized. Source code should be yours and yours only. Do not cheat.

---

1. Introduction

In this assignment, starting from the simplest chat functionalities, you will develop more comprehensive servers for messaging applications, including elementary security features and current state of the chat-room. This will require that various processes in different machines are able to work with each other, and recuperate in case of asynchronous messaging and failure. This chat program is based on a client server model consisting of one chat server and multiple chat clients over TCP connections. The server is mainly used to authenticate the chat clients and direct the chat messages (online or offline) to one another. Besides, the server also has to support certain commands that the clients can use. Detailed specifications of the functionalities are given under Section 2.

---

2. Specifications

a. You should write your program in one of the following languages

- **C/C++**
- **Java**
- **Python**

b. You should write a server program and a client program. The server program will be run first followed by multiple instances of the client program (Each instance supports one client). They will be run from the terminals on the same and/or different hosts.

c. Server program

- You should have a file "user_pass.txt" that contains the valid combination of usernames and passwords that it can use to authenticate the clients (or client programs). In each line, the first term in the username and the second term in the password. Use the following usernames-passwords to populate your file.

  Columbia 116bway
  SEAS winterbreakisover
  csee4119 lotsofexams
  foobar passpass
  windows withglass
  Google hasglasses
  facebook wastingtime
  wikipedia donation
  network seemsez

- The server program will be invoked as

  % **Server <server_port_no>**

  Ex: java Server 4119

  python Server.py 4119

- When starting up, server reads a list of username-password combinations from "user_pass.txt" and then listens on a given port (4119 in the above example) and waits for clients to connect.

- **Authentication**
  - When a new chat client requests for a connection, the server should prompt the client to input his username and password and should authenticate the user (using the username-password combinations from "user_pass.txt").
  - If the combination is correct, the chat client should be "logged in" and a welcome message may be displayed.
  - If the password is incorrect, the server should ask the user to try again until there are 3 consecutive failures. In this case, the server should drop this connection and block access only for this user from the failed attempt IP

address for 60 seconds. Please define 60 seconds using the variable "**BLOCK_TIME**" (we will be changing this value while testing your code!).

- **Commands**
  - After the client is logged in, the server should support the commands presented in Table 1. An example is presented in the Appendix.
  - If the server cannot recognize some command, an error message should be displayed.

Table 1: Commands

Command	Functionality
whoelse	Displays name of other connected users
wholasthr	Displays name of only those users that connected within the last hour.
broadcast <message>	Broadcasts <message> to all connected users
message <user> <message>	Private <message> to a <user>
block <user>	Blocks the <user> from sending any messages. If <user> is self, display error.
unblock <user>	Unblocks the <user> who has been previously blocked. If <user> was not already blocked, display error.
logout	Log out this user.

d. Client program

- The client program will be invoked as

  % **Client <server_IP_address> <server_port_no>**

  Ex: java Client 10.6.31.102 4119

  python Client.py 10.6.31.102 4119

- User should be able to connect to a given server and login themselves by entering a valid username and password.

- For the sake of simplicity, **prohibit concurrent duplicate users**. Ex.: While the user Columbia is already logged in, make sure that Columbia cannot log in from another console.
- After logging in, users should be able to give any of the commands specified in Table 1 to the server. Your program should be able to display what the server responds at the terminal.
    - A client must be able to view other users who are currently online
    - A client must be able to view other users who were online in the past one hour. When defining this one hour, use the variable "**LAST_HOUR**" (we will be changing this value to less than an hour while testing your code!).
    - The client should be able to send and receive private messages with other clients VIA the SERVER. If the receiving user is not online, the server must store the message and deliver it to the receiver when the user comes online. This is the off-line chat functionality.
    - A client can broadcast a message to all logged in users at any time. (Off-line chat function is not needed for broadcast messages.)
    - Since this is a simple chat program, all messages may be displayed in same console only. A separate console (or UI) for each private messaging is not compulsory.
    - A client can block a user from sending any further messages.
    - A client can also unblock a user that was earlier blocked.
    - When all work is done, user should be able to logout from the server.
    - Also, if a client is inactive (*i.e.* the client has not issued any command) for 30 minutes, the server should automatically log this user out. Please define 30 minutes using the variable "**TIME_OUT"**.

e. Any innovative and useful features can be developed. This is your chance for earning some for extra credit!

NOTE: All messages between clients MUST be sent VIA the SERVER.

---

3. Deliverables:

Submission will be done on courseworks. Please post a single **<UNI>_<Language>.zip** (Ex. zz1111_java.zip) file to the Programming Assignment 1 folder. The file should include the following:

- **README.txt**: This file should contain the following.
    a. A brief description of your code
    b. Details on development environment
    c. Instructions on how to run your code
    d. Sample commands to invoke your code
    e. Description of an additional functionalities and how they should be executed/tested.
- **Makefile**: The make file is used to build your application. Please try to keep this as simple as possible. If you don't know how to write a makefile, read this quick tutorial http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/.
- **Source code** and **related files** (like user_pass.txt). (Make sure your code is commented!)

---

4. Testing, Sample Run, and Grading

- Please make sure that your programs compile and run correctly. To grade your program, we will extract the received zip file, type make and run the programs using the same syntax given in Section 2. Please make sure that your submission successfully passes this simple procedure.
- TA's will compile and test the code you submit **on CLIC machines**. It is your responsibility to make sure your code is compliable and runnable on CLIC machines. The **CLIC lab has following setup: Java 1.6, gcc version 4.6.3, python 2.7.3**
- Do not simply submit the entire eclipse/your favourite project folder. Submit only the relevant files.
- Although we mainly observe the functionality and correctness of your program, we might examine your source code a little. Too poorly structured code or human-unreadable code may result in a score penalty

Here is a sample run showing how we will be testing your code:

Terminal 1

```
>make
>java server 4119                    // You don't have to use Java, and 4119 here is just an
                                     // example. We may test on other unoccupied ports
```

Terminal 2

>java client 10.11.12.13 4119	// 4119 is the port that server listens on
	// and 10.11.12.13 is the IP address of the server
	// program

>Username: network	// After connecting to server, your program should
	// prompt "Username: " and wait for user to put in
	// his/her name. Here our user is "network"

>Password: seemsez	// again your program should prompt "Password: " and
	// wait for user to put in the password. Since this is a
	// introductory assignment, the password displayed as
	// plain text is acceptable

>Welcome to simple chat server!	// welcome message from the server acknowledging that // the user has logged in
>Command:	// from now on your server should respond to // commands specified in Table 1

---

5. Appendix: Example

Consider the following scenario

3 clients - facebook, wikipedia and network are currently logged in.

1 client - windows was logged in, but logged out more than an hour back.

1 client - Google was logged in, but logged out half an hour back.

The following should happen on entering the given commands on the terminal of facebook.

- **whoelse**

  Since whoelse should print all the other users currently logged in, the following should be displayed on the terminal of facebook.

  wikipedia

  network

- **wholasthr**

  Since wholasthr should print all the users who were logged in during the last one hour, the following should be displayed on the terminal of facebook

wikipedia

network

Google

Note: windows is not displayed!

- **broadcast hello world**

  The following should be printed on all the terminals of all the users who are currently logged in, i.e., wikipedia and network.

  facebook: hello world

- **message wikipedia hi**

  The following should be printed on the terminal of wikipedia

  facebook: hi

- **message foobar hi**

  Note that foobar is currently not online. So, once foobar comes online, the following should be printed on the terminal of foobar

  facebook: hi

  This is support for offline chat function.

- **block facebook**

  An error message like the following should be printed on the terminal of facebook.

  Error! You cannot block yourself!

- **block wikipedia**

  The following message should be printed on the terminal of facebook.

  You have successfully blocked wikipedia from sending you messages.

  The following message should be printed on the terminal of wikipedia when it tries to send a message to facebook using “message facebook hi”.

  You cannot send any message to facebook. You have been blocked by the user.

- **unblock wikipedia**

The following message should be printed on the terminal of facebook.

You have successfully unblocked wikipedia.

- **logout**

  facebook should be logged out and the client program supporting this user should terminate.

Grading rubric

Functionality	Max Points awarded / deducted
Basic client and server programs with following functionality as per the specifications: - successful log in - Multiple clients support (from multiple machines) - Implementation of whoelse - Implementation of logout	35
Correct implementation of wholasthr	5
Correct implementation of broadcast	10
Correct implementation of private messaging	10
Correct implementation of offline message	10
Correct implementation of user issued block and unblock	10
Blocking the user from a failed login terminal after 3 unsuccessful logins	5
Automatic “logout” of the client after 30 minutes of inactivity	5
Well documented README, makefile (if required)	5
Code quality	5
Extra feature: The feature should be useful and grading will be done on its innovation, utility and amount of effort.	Max 10 points per extra feature. On TA discretion.
Total max extra feature points	20
