CSEE 4119: Computer Networks, Spring 2016

Assignment 3: Socket Programming

Due March 23, 2016 – 5 pm

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 hard copies 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.

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, using TCP connections, not HTTP. The server authenticates the chat clients and direct the chat messages to the correct client.

Overall specifications

- You should write your program in C/C++, Java or Python.
- Your program can either use terminal (text) input and output OR a graphical user interface (GUI).
- You should write a server program (named Server.java for instance) and a client program (named Client.java for instance). The server program will be run first followed by multiple instances of the client program, where each instance supports one client. They will be run from the terminals on the same or different hosts.
- External variables referenced below should be set as environment variables.

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 is the username and the second term is the password. Use the following username-password combinations to populate your file; the file should use the SHA1 hash function to encode your password since storing plain-text passwords in a file would be really bad programming practice. User names can contain only letters (case-sensitive) or numbers.

```
columbia 116bway
seas winterisover
csee4119 lotsofassignments
foobar passpass
```

```
windows withglass
google partofalphabet
facebook wastetime
wikipedia donation
network seemsez
```

The server program will be invoked as

```
$ Server <server_port_number>
```

Examples:

```
java Server 4119
python Server.py 4119
```

When starting up, the 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"), *using the SHA1 hash function*. If the combination is correct, the chat client should be logged in and a welcome message may be displayed. If the user name does not exist, simply display an error message.

If the password is incorrect, the server should ask the user to try again until there are three consecutive failures. In this case, the server should drop this connection and block access only for this user from the failed attempt IP address the number of seconds contained in the environment variable `BLOCK_TIME`, with a default of 60 seconds if undefined (we will be changing this value while testing your code!).

Commands and examples

After the client is logged in, the server should support the commands presented in the table below. If the server cannot recognize some command, it should display an error. You may find regular expressions a quick way to recognize commands. Items enclosed in <> are variables; do not include the <>. For the examples below, consider the following scenario four clients, facebook, apple, wikipedia and network, are currently logged in. One client, windows, was logged in, but logged out more than an hour ago. One client, google, was logged in, but logged out half an hour ago.

Command	Functionality	Users affected
`who`	Displays name of other connected users, including the querying user	`wikipedia apple network`

`last <number>`	Displays name of those users connected within the last *<number>* minutes (0…60), e.g., *`last 32`*, including self.	wikipedia apple network google
`broadcast <message>`	Broadcasts *<message>* to all connected users, but not sender.	wikipedia apple network google
`send (<user> <user> ... <user>) <message>`	Sends *<message>* to the list of users, separated by space. Ignore users who are not logged in.	
`send <user> <message>`	Private *<message>* to a single *<user>*	
`logout`	Log out this user	

Client program

The client program will be invoked as

```
$ Client <server_IP_address> <server_port_no>
```

Example:

```
$ java Client clic1.cs.columbia.edu 4119
$ python Client.py clic2.cs.columbia.edu 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. For example, when 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 anytime in the past one hour (0-60 minutes).

The client should be able to send and receive private messages with other clients via the server. If the user is not online, simply discard the message.

A client can broadcast a message to all logged in users at any time. All other logged in clients should be able to view the broadcast message at their terminal instantly.

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.

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 environment variable TIME_OUT.

The client and server should accept the keyboard input 'Control + C' as a signal to exit the program. The programs should exit gracefully upon this input.

<username>, <password> shall not contain spaces or newlines, but *<message>* may contain anything (in UTF-8) except newlines.

When a user is blocked, this means we drop the connection and prevent any new connections being made from the IP Address of the user for the specified BLOCK_TIME for that user, but any other logins made from that IP address should be allowed.

On a duplicate user login and we reject the login, this does not count as a failed login attempt.

Any innovative and useful features can be developed. This is your chance for earning some for extra credit! (One such example is implementing offline messaging. If the receiving user of a private message is not online, the server can store the message and deliver it to the receiver when the user comes online).

NOTE: All messages between clients must be sent via the *server.*

Deliverables

Submission will be done on Courseworks. Please post a single **< UNI>_<Language>.zip** (e.g., zz1111_java.zip) file to the Assignment 3 folder. The file should include the following:

`README.txt:` This file should contain the following:

- A brief description of your code
- Details on development environment
- Instructions on how to run your code
- Sample commands to invoke your code
- Description of any additional functionalities and how they should be executed and tested.

`Makefile`: The Makefile 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!

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.

Name the executable programs as Server and Client, names being case sensitive.

TA's will compile and test the code you submit on **CLIC machines**. It is your responsibility to make sure your code is able to be compiled 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 (or other development environment) project folder. Submit only the relevant files.

We will examine your source code. Poorly structured code, comment without comments or human-unreadable code will 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 is just an example. We may test other ports, including ports in use, and your code should handle any errors.

Terminal 2

```
$ java Client 10.11.12.13 4119
Username: network
Password: seemsez
Welcome to the simple chat server!
Command: who
```

4119 is the port that server listens on and 10.11.12.13 is the IP address of the server program. You should also be able to use a domain name. After connecting to server, your program should prompt for "Username:" and wait for user to type his or her user name. Then, your program should prompt for "Password:" and wait for user to input the password. Since this is an introductory assignment, showing the password as plain text is acceptable. From now on your server should respond to the commands specified in the table above.

Grading

Functionality	**Maximum points awarded or deducted**
Basic client and server programs with following functionality as per the specifications: ➢ Successful log in ➢ Multiple clients support (from multiple machines) ➢ Implementation of who ➢ Implementation of logout ➢ Error handling (socket errors)	40

Correct implementation of last	10
Correct implementation of broadcast	10
Correct implementation of private messaging	10
Automatic logout of the client after 30 minutes of inactivity	5
Graceful exit of client and server programs using control + c	5
Code quality (structure, documentation)	20
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. At TA discretion.
Total maximum extra feature points	20