CIS427 Winter 2022
Programming Assignment 3
John P. Baugh, Ph.D.

MatheMagic Online – The Sequel

Introduction

This programming assignment is designed to continue to all you to familiarize yourself with the socket interface and client-server applications.

For this assignment, you will continue to design and implement an online geometry solver using network sockets. You will write both the client and server portions of this application. The client and server processes will communicate using **TCP** sockets and will implement the protocol discussed below.

The software must support the same commands as before, and additionally:

- Allow multiple simultaneous clients to be connected
- Allow the server to know which client is which
- Allow the clients to communicate using the **MESSAGE** command

The Assignment

You will write two programs, a server and a client. The **server** creates a socket in the Internet domain bound to port SERVER_PORT (a constant you should define in both programs, you may use last 4 digits of your UM-ID). The server receives requests through this socket, acts on those requests, and returns the results to the requester. The client will also create a socket in the Internet domain, send requests to the SERVER_PORT of a computer specified on the **command-line**, and receive responses through this socket from a server.

This time, you must allow ***multiple*** clients to be connected simultaneously, and the server must know which client is which. This should include authorization characteristics, such as administrative privileges as root.

Your clients and server should operate as follows. Your server begins execution by opening a file named **logins.txt**. This file contains the following usernames and passwords:

root	root22
john	john22
sally	sally22
qiang	qiang22

These are the only authenticated users allowed. For this project, you will not allow new user sign-up or deletion of any of the users.

Your clients will operate by sending **LOGIN, MESSAGE, SOLVE, LIST, SHUTDOWN, LOGOUT** commands to the server. You should create clients that are able to send any of the commands above, and allow a user to specify which of the commands the clients should send to the server.

The details of the protocol depend on the command the client sends to the server.

LOGIN

This is the command a client must initiate in order to gain access to anything on the server. It would be in the following format:

```
C:      LOGIN john john22
S:      SUCCESS
```

Or, if the login information is incorrect (username or password, or both are wrong):

```
C:      LOGIN dude dude111
S:      FAILURE: Please provide correct username and password.  Try again.
```

SOLVE

Only after being logged in may a user access the SOLVE command.

When the user issues a SOLVE command, the server will return the solution or error message, and also must create (or overwrite) a file with the user's username followed by _solutions.txt. For example, solutions that user **john** requests will be saved on the ***server*** side as **john_solutions.txt**. Likewise, for sally, the file will be **sally_solutions.txt**.

Additionally, the solutions are ***maintained*** in the file between different logins. For example, if john logs in and requests three solutions, and gets solutions (or errors), then those solutions (or errors) are recorded in the file, and are still present when john logs out and then later back in. If john requests one more solution, the john_solutions.txt will then contain ***four*** solutions. Etc.

The solver can be used for either solving for rectangles or for circles, which it distinguishes by the **-c** and **-r** flags. The general format is:

SOLVE -c 6

or

SOLVE -r 2 4

The -c flag indicates a circle, and the number following it is the radius. The -r flag indicates a rectangle, and is followed by two numbers: the lengths of the sides. **If the user puts only 1**

number for the rectangle, the server assumes it's a square with two sides, both equal to that value. The output for the returned calculations should be to two decimal places.

Interactions might look like this:

```
C:    SOLVE -c 4
S:    Circle's circumference is 25.13 and area is 50.27
```

```
C:    SOLVE -c
S:    Error:  No radius found
```

```
C:    SOLVE -r
S:    Error:    No sides found
```

```
C:    SOLVE -r 2
S:    Rectangle's perimeter is 8.00 and area is 4.00
```

```
C:    SOLVE -r 2 6
S:    Rectangle's perimeter is 16.00 and area is 12.00
```

LIST

Two options for this command:

- With no flags, the LIST command returns a list of all the solutions requested by this particular user (e.g., john will only see the contents of john_solutions.txt)
- With the **-all** flag
    - If the user is **root**, who is successfully logged in, the command lists all of the solutions requested (including root's own) from all of the files, organized by username

A regular (flagless) interaction might look like the following (see above commands issued under SOLVE):

```
C:    LIST
S:    john
          radius 4:  Circle's circumference is 25.13 and area is 50.27
          Error:  No radius found
          Error:    No sides found
          sides 2 2:  Rectangle's perimeter is 8.00 and area is 4.00
          sides 2 6:  Rectangle's perimeter is 16.00 and area is 12.00
```

With the **-all** flag, the interaction would look similar, but list all interactions by all users. Note again, if the user is ***not*** the root, they should get an error:

```
C:   LIST -all
S:   Error:  you are not the root user
```

Example interaction if the user isn't the root

```
C:   LIST -all
S:   root
          radius 5:  Circle’s circumference is 31.42 and area is 78.54
          Error:  No sides found
          sides 2 3:  Rectangle’s perimeter is 10.00 and area is 6.00
     john
          radius 4:  Circle’s circumference is 25.13 and area is 50.27
          Error:  No radius found
          Error:    No sides found
          sides 2 2:  Rectangle’s perimeter is 8.00 and area is 4.00
          sides 2 6:  Rectangle’s perimeter is 16.00 and area is 12.00
     sally
          No interactions yet
     qiang
          No interactions yet
```

Example interaction if the user is the root

SHUTDOWN

The SHUTDOWN command, which is sent from the client to the server, is a single line message that allows a user to shutdown the server. A user that wants to shutdown the server should send the ASCII string "SHUTDOWN" followed by the newline character (i.e., '\n').

Upon receiving the SHUTDOWN command, the server should return the string "200 OK" (terminated with a newline), close all open sockets and files, and then terminate.

A client-server interaction with the SHUTDOWN command looks like:

```
c: SHUTDOWN
s: 200 OK
```

LOGOUT

Terminate ***only*** the client. The client exits when it receives the confirmation message from the server.

A client-server interaction with the LOGOUT command looks like:

```
C:   LOGOUT
S:   200 OK
```

MESSAGE

Allow each client to be able to contact any of the other clients. Additionally, the root has special ***broadcast*** privileges. The sending client sends a message to the receiving client using the recipient's username. The clients ***do not communicate directly***. The communication must go through the server.

A regular client-server-client between **john** and **qiang** should look like the following:

John's window	Server	Qiang's window
C: MESSAGE qiang Hello there!	S: Message from client: Hello there! Sending to qiang	C: Message from john: Hello there!

Regular chat session – between john and qiang

John's window	Server
C: MESSAGE qiang Hello there!	S: Message from client: Hello there! Sending to qiang qiang is not logged in. Informing client.
S: User qiang is not logged in	

Message failure if qiang isn't logged in

John's window	Server
C: MESSAGE kiumi Hello there!	S: Message from client: Hello there! Sending to kiumi User kiumi doesn't exist. Informing client.
S: User kiumi does not exist	

Message failure if no user exists with that name

Root's window	Server	ALL logged in users' windows
C: MESSAGE -all Hello there!	S: Message from client: Hello there! Sending to qiang	C: Message from root: Hello there!

Root message broadcast

Note the hyphen before the **all** command.

Invalid Commands

Note, “300 invalid command” or “301 message format error” should be returned to the client, if a server receives an invalid command or the command in the wrong format.

Format

You may work in a team of no more than **four students.** You may choose to work by yourself, or with a team of up to 4 students. Not 12 other people. Not 5 other people. Not 7 other people. Teams can be only 4 people, maximum.

Programming Environment

You can use either Java or Python to implement the server and client.

Requirements

The following items are required for full credit:

- Implement all of the commands: **LOGIN, SOLVE, LIST, SHUTDOWN, LOGOUT**
- You must implement the client and server in either **Java** or **Python**
- Make sure that you do sufficient error handling such that a user can't crash your server. For instance, what will you do if a user provides invalid input?
- The server IP address should be held in a variable on the client side so that it can connect
- The server should print out all messages received from clients on ***its*** screen
- When the previous client exits, the server should allow the next client to connect.
- Your source codes must be commented
- Include a **README** file in your submission.

Note, the README file should be a plain text file. In your README file, the following information should be included: the commands that you have been implemented, the instructions about how to build and run your program, any known problems or bugs, and **the ouput at the client side of a sample run of all commands you implemented**.

Grading (100 points)

- Correctness and Robustness (85 points)
  - You will lose at least 10 points for any bugs that cause the system crash.
  - You will lose at least 5 points for any other bugs.
- Comments and style (5 points)
- README and any required Project file(s)(5 points)
- GitHub source control (5 points)

Submission Instruction

Zip all of the files into a file named **first_last_p3.zip** and upload this to Canvas.

Additionally, you ***must*** place your code under source control using GitHub and include the link in your README file. No extra credit. Sorry 😊