Project 1

CIS 427 – Dr. John P. Baugh
Fall 2018

Points: _____________ / 100

This programming assignment is designed in order to help you become familiar with sockets and client-server applications.

1. Yet Another Message of the Day (YAMotD)

In this programming assignment, you will create a server process that implements "yet another message of the day" protocol with a client process you will also create. The client and server processes will communicate using TCP sockets and will implement the YAMotD protocol discussed below.

The YAMotD server performs four functions:

- Returns a message of the day to any user that sends the server an MSGGET message
- Allows a user, whose identity has been verified by the server, to send the server a MSGSTORE message to upload one or more messages of the day to the server
  - These messages will be appended to the file the server maintains (messages.txt) and can be returned to any client later that sends the MSGGET message to the server
- Allows the root user to send the server a SHUTDOWN message, which will cause the server to close any open sockets and then terminate
- Verified the identity of a user through LOGIN

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 must define in both programs. It should be four digits, and can be the last 4 digits of your UMID for example.) The server receives requests through this socket, acts on those requests, and returns the results to the requester (client.)

The **client** will also create a socket in the Internet domain, send requests to the server using SERVER_PORT, and receive responses through this socket from the server. For this assignment you just need to allow one active client to connect to the server at any given time.

Your client and server should operate as follows:

- Your server begins execution by opening a file that you have created (messages.txt) that contains five messages of the day that you've written in manually

    - A “message of the day” is just a simple, short saying or nugget of wisdom like, “Early to bed, early to rise, makes one healthy, wealthy, and wise” or “One can never be too rich, too thin, or have too much bandwidth”
- Your server should read these messages into a server-internal data structure and keep track of the number of messages of the day
    - Use a dynamic data structure, such as ArrayList in Java, to keep track of an arbitrary number of messages
- Once the server has initialized the data structure, it should wait for the connection requests from the clients
- Your client operates by sending a MSGGET, MSGSTORE, SHUTDOWN, LOGIN, LOGOUT, and QUIT commands to the server
- You should create a client that is able to send any of the six commands above, and allows a user to specify which of the commands the client should send to the server.

MSGGET

The MSGGET command, which is sent from client to server, consists solely of the string “MSGGET” followed by the newline character (i.e., you hit enter/return, generating a newline ‘\n’.) After sending MSGGET, the client waits to receive a message of the day from the server via the socket. After displaying the message of the day, the client should loop back and allow the user to indicate the next command to be sent.

When your server receives a MSGGET command from a client it should return the string “200 OK” (terminated with newline), followed by one of the messages of the day (also terminated with a newline.)

The server should just return one message. That message should be chosen sequentially by cycling through its message of the day.

A client-server interaction might look like this:

```
C:  MSGGET
S:  200 OK
Anyone who has never made a mistake has never tried anything new.
```

MSGSTORE

The MSGSTORE command allows a logged in user to upload one message to the server. A client that wants to store one message should begin by sending the string “MSGSTORE”, followed by the newline character (‘\n’.) The client should then wait for the server to return a “200 OK” message indicating the client is authorized to upload messages, or a “401 You are not currently logged in. Log in first.” If the client has been authorized to upload a message, the client response to the 200 OK message by sending one message to be added to the server’s message store.

The message should be terminated with a newline. After sending MSGSTORE and the message of the day, the client should read and display the return code sent by the server. When the server receives the MSGSTORE from a client, it should check the login status of the client. If the client has not logged in yet, the server should return the string, “401 You are not currently logged in. Log in first.” Otherwise, it sends “200 OK” message, as discussed earlier. If the user is allowed to execute the MSGSTORE command, the server should then read in and store (in its internal data structures and in the file) the new additional message of the day. If the message of the day is received correctly, your server should return the string “200 OK” (terminated with a newline).

A client-server interaction with the MSGSTORE command thus looks like:

```
C:  MSGSTORE
S:  200 OK
C:  Imagination is more important than knowledge.
S:  200 OK
```

SHUTDOWN

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

The client should then read and display the return code sent by the server, and then close the connection. When your server receives a SHUTDOWN command from a client, it should check if the current user is the root. If it is not the root user, the server should return the string “402 User not allowed to execute this command.” If the user can execute 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 thus looks like:

```
C:  SHUTDOWN
S:  200 OK
```

ERROR

In case of an error, the string, “300 message format error” should be returned.

LOGIN

Identify the user to the remote server. A client that wants to login should begin by sending the ASCII string “LOGIN” followed by a space, followed by a UserID, followed by a space, followed by a Password, and followed by the newline character (i.e., ‘\n’). Your server should be initialized with the UserIDs and Passwords of at least four users who will be allowed to execute the MSGSTORE and SHUTDOWN (the root user only) commands at the server. When the server receives a LOGIN command from a client, it should check if the UserID and Password are correct and match each other. If they are not correct or don’t match each other, the server should return the string “410 Wrong UserID or Password,” otherwise it should return the “200 OK message. A client-server interaction with the LOGIN command thus looks like:

```
C:  LOGIN john john
S:  200 OK
```

LOGOUT

Logout from the server. A user is not allowed to send MSGSTORE and SHUTDOWN commands after logout, but it can still send MSGGET command. A client-server interaction with the LOGOUTcommand looks like:

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

QUIT

Terminate the client with the remote server. The client exits when it receives the confirmation message from the server. A client-server interaction with the LOGOUT command looks like:

```
C:   QUIT
S:   200 OK
```

2. Requirements

You should form a team of two students (or work individually) and then jointly design your project. While each project should be an integrated effort, you should identify in your README file what part of the project each member is responsible for. You can use either Java or Python to implement the assignments. Please don't use any GUI. The following items are required for full-credit:

- Implement all six commands: MSGGET, MSGSTORE, SHUTDOWN, LOGIN, LOGOUT, QUIT
- The users' information should be maintained by the server. You must have the following users (lowercase) in your system:

UserID	Password
root	root01
john	john01
david	david01
mary	mary01

- Both the server and client should be able to run on any reasonably modern Windows machine
- When the previous client exists, the server should allow the next client to connect
- Your source code must be commented
- Include a README file in your submission
  - In the README, you must include the functions that have been implemented, instructions about how to run your program, known bugs, and sample outputs

3. Submission

Place all of your source files (.java or .py) and README file into a folder. Zip the folder, and upload it under the appropriate submission directory on Canvas.

4. Hints

Make sure to research the **Scanner**, **String, Socket**, and **ServerSocket** classes. Consider how the String class's various methods (e.g., split) might help you. Use Scanner to read from user and from file (using the **File** class.) Also, use **PrintWriter** to write to a local file.

If you’re not familiar with Java, for example, you might consider making a few simple projects in NetBeans first before you dive into the deep end with this project.