Networking | **Programming Project 1**
CSC3935 Spring 2024 (Kissel) | **Due Date: Friday Feb. 9, 2024**

! This is a two-person group project.

Fortune (`fortune(1)`) is a UNIX/Linux program that provides quotes to the user from some quote database. In this project you will implement a networked version of the fortune program in the client-server model. Your server will be responsible for serving up quotes when requested to do so by the client. The client will ask the server to either send it a random quote from the database or a random quote by a specific author in the database. To prevent you from having to find a large collection of quotes I have provided a database of quotes in JSON format. You are free to add to this database if you wish.

STOP You will be writing networked code in this assignment, please start early as this presents its own challenges.

As mentioned you must write two applications for this project.

1. A client application that may either have a command line interface or GUI.
2. A server which *must* be a command line application. Services on your machine are never graphical.

I encourage you and your group to participate in the construction of all applications since their operations are joined by the protocol they utilize.

Fortune Client Application

The client application should do the following:

1. Ask the user for a type of quote: random or random from an author.
2. Connect to the fortune server and send the request packet.
3. Receive the response from the server and output the quote and its author (see examples below).

In order to connect to fortune server the client must know the address of the server as well as the port number that the fortune service (server application) is listening on. This will be specified in a JSON configuration file. This file contains one object that consists of two fields:

- A field with key `server-address` that represents the server IP address as string.
- A field with key `server-port` that represents the port for the fortune service as an integer.

By default the configuration information should be read from a file `config.json` *without* user intervention. You are allowed to have an option to your program that allows the user to specify the location of the configuration file but, again the user should not have to.

Fortune Server (Daemon)

The fortune server program (also called a service or daemon) is responsible for serving up quotes to clients. In general the fortune server should do the following:

1. Load the database of quotes into a data structure (e.g., an `ArrayList`).
2. Construct a `ServerSocket` to accept incoming connections.
3. On every new accepted connection should be placed in a thread pool. The connection thread should handle the interaction with the client. In particular it should receive a request, identify the type of request, and send the appropriate response packet to the client.

In order to operate correctly the server requires configuration information which includes its port number, the location of the quotes database file, and optionally the size of the thread pool (in number of threads). This will be specified in a JSON configuration file. This file contains one object that consists of up to three fields:

- A field with key `fortune-database` this gives a file path to the JSON file that represents the quotes database.
- A field with key `port` that represents the port for the fortune service as an integer.
- An *optional* field with key `pool-size` that represents the number of threads in the thread pool. If this field is not present, the server should use a thread pool of size 10.

By default the configuration information should be read from a file `config.json` *without* user intervention. You are allowed to have an option to your program that allows the user to specify the location of the configuration file but, again the user should not have to.

The quotes database is a JSON file is one class with a single field with key `quotes` whose value is an array of JSON `fortune` objects. The fortune object consists of two fields:

- A field with key `quote` that represents the quote as a string.
- A field with key `author` that represents the name of the author as a string.

Fortune Protocol

The client and server will utilize the *fortune protocol* to communicate with one another. I have the following global requirements for you.

1. All message should be sent as JSON strings representing objects. You will want to use the `toJSON` method which is provided by any descendant of `merrimackutil.json .types.JSONType`. This is a non-formatted form of the JSON that is amenable to communication over `Socket`s. This is a simplifying assumption in production grade protocols it is most common to encode the messages into a binary format, this is an extra layer of complexity we don't need.

2. To maintain interoperability, I will ask that your message formats match exactly as described below.

The protocol itself is fairly simple and consists of two messages a request message and a response message. The client is responsible for sending a request message and the server is responsible for sending the response message.

The message flow is as follows:

Message 1. (Request) The client send the sever a message which has a field with key `type` which is an integer. If the type is 0, no other fields should be in the message. If the type is 1 (an author query) there should additionally be a field with key `author` which is a string that represents the author to a get a random quote from. If the request is an author request, the server will return a random quote where the author name *contains* the value stored in the `author` field. Any matches should be found in a *case insensitive* fashion.

Message 2. (Response) The server sends this message which has a field with key `success` which is a boolean. If the success field is false, no other fields should be present in the message. This field is false only if the request could not be satisfied, this happens if

- The type field in the request is invalid.
- A quote satisfying the criteria can not be found.

If the success field is true, there should be a field with key `fortune` which contains a fortune object (as described in the server overview).

Helpful Hints

I offer the following helpful hints as you embark on the project:

- Networking code is notorious for getting complicated fast, be sure to abstract and modularize your code for ease of debugging.
- Employ good design principles.
- Use Wireshark to aid you in debugging your code. If you set Wireshark to look at the loopback interface only you should be able to quickly isolate your protocol messages (remember the will show up in TCP segments).

- Only one `ServerSocket` can be bound to a port at a time. Remember these port numbers should be on the high end of the numbers (anything 1024 or below is reserved, and probably used by your machine) It is possible that a port can be hung during development. You can correct this by resetting your network card (or just restarting your computer).
- All JSON objects should have a corresponding Java class that implements the `JSONSerializable` interface.
- Use the *localhost* IP address `127.0.0.1` for any host name you use in this application. While your code can work on a LAN, it will add unnecessary complications to development and testing. Once you get your application working on one machine, you are welcome to come see me to borrow a switch and try it out on a LAN you construct in the CaDS lab or out of you and your team mates computers.
- You should *not* have to disable your firewall to work on this project. Again, **do not disable your firewall**.
- I've mentioned it earlier but, **start early** and work as a team.
- In order to get the JSON support you need to download or clone `merrimackutil` from `https://github.com/kisselz/merrimackutil`

Testing

To test your application I have provided you with necessary files for the client a configuration file `config.json` be found in the `client-config` folder and for the server in folder `server-config` a configuration file `config.json` as well as a quote file `quotes.json`. Your code should work with my provided files.

Examples

My server has the following basic options.

```
$ java -jar dist/fortuned.jar -h
usage:
    fortuned
    fortuned --config <config>
    fortuned --help
options:
    -c, --config Config file to use.
    -h, --help Display the help.
```

My client has the following basic options

```
$ java -jar dist/fortune.jar -h
usage:
    fortune [ --author <author> ]
    fortune --config <config> [ --author <author> ]
    fortune --help
options:
    -c, --config Config file to use.
    -a, --author Author to get quote from.
    -h, --help Display the help.
```

Where options in brackets are not required.

As an example of running the client to request a random fortune:

```
$ java -jar dist/fortune.jar
"Courage is going from failure to failure without losing enthusiasm."
   -- Winston Churchill
```

As an example of running the client to request a random fortune by an author.

```
$ java -jar dist/fortune.jar -a "Edsger Dijkstra"
"Computer Science is no more about computers than astronomy is about
telescopes."
   -- Edsger Dijkstra
```

Source Control

In all projects in this class you will use source control. The source control system the department has chosen is `git`. In particular we will use GitHub. For this project please create a *private* GitHub repository under and account linked to your Merrimack e-mail.

As part of your grade on this project I will be requiring good use of the source control. This means your project should have several commits over the course of time.

Don't develop your entire program outside of source control and then add it to source control before submission!

Every commit should have a well worded commit message that explains what the update to the repository is all about. Please be sure to not put automatically generated files in the repository. This includes `class` files, `jar` files, and `zip` files.

Since this a group project each members contributions will be judged based on the commit history. For example, a group member that has one tiny commit will receive a lower grade than other members. To make your submission clear, please have a `README` in your repository that lists all of the members of the group.

Submitting

To submit this program, please add me as a contributor to your private GitHub repository, my username is (`kisselz@merrimack.edu`). Since this is a group project, there should only be one repository which each group member contributes to.

Rubric

Your grade on this project will be determined as follows:

- Good use of source control (10 points).
- Code is well architected (10 points).
- The protocol messages allow for interoperability (10 points).
- The client and server correctly use configuration files (5 points).
- The fortune protocol operates correctly (30 points).
- The thread pool is correctly utilized (15 points).
- Use of the socket API is correct for the client-server model (10 points).
- The fortune database is correctly implemented (10 points).