v1.2

LineQueue

In this exercise, you will build and document a system that is capable of maintaining a queue of text lines for access by clients.

You may do so in any language (though Java or another JVM-language is preferred).

You may use any reference and any (open-source) software you can find to help you build this system, so long as you document your use. However, you should not actively collaborate with others.

Specification

Your system should act as a console application that can both receive individual lines from the standard input and serve them back writing to the standard output using the following protocol:

PUT text

Inserts a ***text*** line into the FIFO Queue. ***text*** is a single line of an arbitrary number of case sensitive alphanumeric words ([A-Z]+[a-z]+[0-9]) separated by space characters.

GET n

Return ***n*** lines from the head of the queue and remove them from the queue.

If ***n*** is not a valid line count, the server should return a "***ERR\r\n***" without dequeuing any elements.

SHUTDOWN

Shutdown the application/server.

The system should perform well as the number of requests per second increases.

Optional extensions

You can focus on any part of this task that is the most enjoyable to you. For example if you want to submit the fastest solution possible, that would be able to handle the largest number of requests per second, you don't have to solve the other optional parts! We know that trying to solve all of the optional extensions can be quite a time consuming process. Please show us what you like and what you value the most.

State restore

Optionally, your application should support restoring the state of the queue after a clean ***SHUTDOWN***. In this case, if the application is restarted from the same current working directory, it should restore the state of the queue as it was before the shutdown.

Network server

Instead of reading from STDIN and writing to STDOUT, you can implement your application as a network server. If that's what you would like to do, the protocol should stay the same as described above, with one extra command:

QUIT
: Disconnect client.

The server should listen for connections on TCP port 10042.

Your server must support at least a single client at a time. Your server may optionally support multiple simultaneous clients.

Example

You could imagine the following interaction with your application:

```
INPUT     => GET 1
OUTPUT    <= ERR
INPUT     => PUT the
INPUT     => PUT quick brown
INPUT     => PUT fox jumps over the
INPUT     => PUT lazy dog
INPUT     => GET 2
OUTPUT    <= the
OUTPUT    <= quick brown
```

```
INPUT     => GET 42
OUTPUT    <= ERR
INPUT     => GET 1
OUTPUT    <= fox jumps over the
```

Example with optional stateful restore:

```
INPUT     => PUT the
INPUT     => PUT quick brown
INPUT     => PUT fox jumps over the
INPUT     => PUT lazy dog
INPUT     => GET 2
OUTPUT    <= the
OUTPUT    <= quick brown
INPUT     => SHUTDOWN
```

Application shuts down (optionally network clients are disconnected).
Application is restarted from the same working directory.:

```
INPUT     => GET 1
OUTPUT    <= fox jumps over the
```

Execution environment

You may assume that your system will execute on a machine comparable to an EC2 64-bit xLarge instance running Ubuntu 20.04.

A 64-bit large instance has:

- 16 GB of memory
- Four 64-bit cores
- a 10 GB of root partition
- a 420 GB drive mounted under /mnt
- a 420 GB drive unmounted

The entire machine is at your disposal.

What to submit

The top-level directory of your submission (or source-tree) should contain shell scripts to build and run your system, documentation for your system, and the source code for the system itself.

- **build.sh** - A script that can be invoked to build your system. This script may exit without doing anything if your system does not need to be compiled. You may invoke another tool such as Maven, Gradle, Ant, GNU make, etc. with this script. You may download and install any libraries or other programs you feel are necessary to help you build your system.
- **run.sh** - A script that takes a single command-line parameter which is the name of the file to serve. Ultimately, it should start the application you have built.

We will probably run these scripts manually, but it would be nice if they worked without manual intervention.

- **README** - A text file that answers the following questions:
    - How does your system work? (if not addressed in comments in source)
    - How will your system perform as the number of requests per second increases?
    - How will your system perform with various queue sizes?
    - What documentation, websites, papers, etc did you consult in doing this assignment?
    - What third-party libraries or other tools does the system use?
    - How long did you spend on this exercise?

The remainder of the files in your tree should be the source-code for your system.

Please make sure that the code has “production ready” quality. You or your co-workers should be comfortable running, maintaining and developing it in the future. Whatever this means to you.

How to submit

Please package everything up as a .zip file.