Luna Szymanski
6650 Project 2 - Multithreaded RMI server

How to Compile and Test:

1. Run RMIServer.java
2. Run RMIClient.java*
    a. Here the put() commands populate the server
3. Open terminal, and use to run a second client using RMIClient.class
    a. The second client does get() and delete() commands

*Note- RMIClient.java has my PUT, GET, and DELETE remote calls so you can comment them out and use "javac" in terminal to compile various combinations of commands.

Assignment Overview

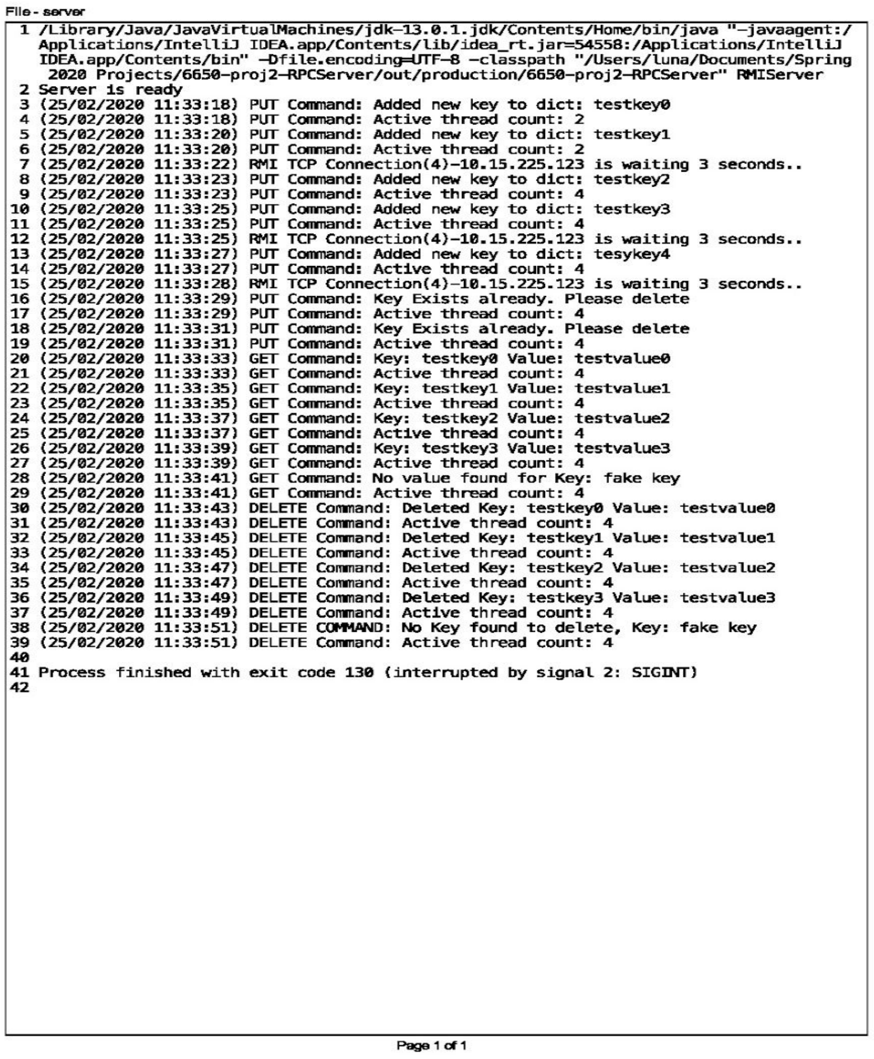
The point of this assignment was to create a server that create new threads for each client that connects to it, while using Remote Procedure Calls. Having separately threaded clients means that the shared resources, in this case my HashMap, needed to be protected. I implemented a basic semaphore that used my threadCount variable to communicate whether or not a thread was currently waiting. I did not explicitly create new threads for each client because RMI handles this under the hood, so what I did is printed the activeCount() for each command to communicate that the server is multithreaded, and used the semaphore to ensure that the threads access the resources at mutually exclusive moments, ideally in the order that was sent out by the clients. What I consider out of scope on this assignment is making sure the order is always guaranteed, so I did not keep a Queue of my waiting threads but instead had them simply wait 3 seconds and try again in an infinite loop. By doing it the way I did I do not guarantee the order. I could also implement a logical clock of some kind, which I also decided was out of scope.

Technical Impressions

One of the challenges of this assignment was in testing, because I needed to show it was multithreaded, but my remote procedures are simple and happen super fast. What I did was use thread.sleep() to simulate a longer command, which then allows me to display how my threads interact and share resources (HashMap dict and int threadCount). Another decision I made was to not explicitly create new threads for each client, as the RMI library does that automatically. Instead I focused on documenting the threads that are active to establish a multithreaded nature, and then I create a mutual exclusion lock on my HashMap to account for multiple threads. If my software was single threaded, then no threads would ever wait and you would never see my std.err messages that occur when threads are waiting.

Attached log shows the server log of two clients communicating at the same time, first one client sending several PUT commands and the second server concurrently makes GET and DELETE commands that are queued until the shared resources are not being used.

<u>Console Log</u>



File - server

```
/Library/Java/JavaVirtualMachines/jdk-13.0.1.jdk/Contents/Home/bin/java "-javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=54558:/Applications/IntelliJ IDEA.app/Contents/bin" -Dfile.encoding=UTF-8 -classpath "/Users/luna/Documents/Spring 2020 Projects/6650-proj2-RPCServer/out/production/6650-proj2-RPCServer" RMIServer
Server is ready
(25/02/2020 11:33:18) PUT Command: Added new key to dict: testkey0
(25/02/2020 11:33:18) PUT Command: Active thread count: 2
(25/02/2020 11:33:20) PUT Command: Added new key to dict: testkey1
(25/02/2020 11:33:20) PUT Command: Active thread count: 2
(25/02/2020 11:33:22) RMI TCP Connection(4)-10.15.225.123 is waiting 3 seconds..
(25/02/2020 11:33:23) PUT Command: Added new key to dict: testkey2
(25/02/2020 11:33:23) PUT Command: Active thread count: 4
(25/02/2020 11:33:25) PUT Command: Added new key to dict: testkey3
(25/02/2020 11:33:25) PUT Command: Active thread count: 4
(25/02/2020 11:33:25) RMI TCP Connection(4)-10.15.225.123 is waiting 3 seconds..
(25/02/2020 11:33:27) PUT Command: Added new key to dict: tesykey4
(25/02/2020 11:33:27) PUT Command: Active thread count: 4
(25/02/2020 11:33:28) RMI TCP Connection(4)-10.15.225.123 is waiting 3 seconds..
(25/02/2020 11:33:29) PUT Command: Key Exists already. Please delete
(25/02/2020 11:33:29) PUT Command: Active thread count: 4
(25/02/2020 11:33:31) PUT Command: Key Exists already. Please delete
(25/02/2020 11:33:31) PUT Command: Active thread count: 4
(25/02/2020 11:33:33) GET Command: Key: testkey0 Value: testvalue0
(25/02/2020 11:33:33) GET Command: Active thread count: 4
(25/02/2020 11:33:35) GET Command: Key: testkey1 Value: testvalue1
(25/02/2020 11:33:35) GET Command: Active thread count: 4
(25/02/2020 11:33:37) GET Command: Key: testkey2 Value: testvalue2
(25/02/2020 11:33:37) GET Command: Active thread count: 4
(25/02/2020 11:33:39) GET Command: Key: testkey3 Value: testvalue3
(25/02/2020 11:33:39) GET Command: Active thread count: 4
(25/02/2020 11:33:41) GET Command: No value found for Key: fake key
(25/02/2020 11:33:41) GET Command: Active thread count: 4
(25/02/2020 11:33:43) DELETE Command: Deleted Key: testkey0 Value: testvalue0
(25/02/2020 11:33:43) DELETE Command: Active thread count: 4
(25/02/2020 11:33:45) DELETE Command: Deleted Key: testkey1 Value: testvalue1
(25/02/2020 11:33:45) DELETE Command: Active thread count: 4
(25/02/2020 11:33:47) DELETE Command: Deleted Key: testkey2 Value: testvalue2
(25/02/2020 11:33:47) DELETE Command: Active thread count: 4
(25/02/2020 11:33:49) DELETE Command: Deleted Key: testkey3 Value: testvalue3
(25/02/2020 11:33:49) DELETE Command: Active thread count: 4
(25/02/2020 11:33:51) DELETE COMMAND: No Key found to delete, Key: fake key
(25/02/2020 11:33:51) DELETE Command: Active thread count: 4

Process finished with exit code 130 (interrupted by signal 2: SIGINT)
```

Page 1 of 1