Rithvik Aleshetty

Implementation

- How does the client and server handle different object store operations?
    - Since there were only a handful of operations between the server and the client, for example: put, get, remove, list…, I was able to encode a 3 letter request that both the client and server understood as a part of the message.
        - Examples:
            - rem signifies remove
            - puf signifies put file
            - put signifies put string
- Design and implementation of communication protocol
    - The server is constantly reading lines in a while loop with a buffered reader
    - The messages when related to strings consisted of:
        - The three-letter request as described previously
        - The 1024 byte key to get, add or remove
        - The String data to place (if applicable)
        - A new line character so that the buffered reader knows when the end is
            - It looked like this when putting a string with “key1” that contains the string “hello world”

put	**Key1 (buffered with 0s to fill 1024 bytes)**	**Hello world**	**\n**

    - The one-word messages like “disconnect” did not contain any key or data but did contain a newline character
    - Operations related to files was done a bit differently:
        - Using a buffered reader for sending and receiving files didn’t work
        - Instead, the server listens for a put or get request for a file specifically and then the number of bytes in the file is first sent over, then that number of bytes is read through a data input stream
        - Here’s how that looked in order:

puf	**key**	**\n**

Number of bytes to read

Bytes of the file

- Data structures
    - The server used a HashMap to store and retrieve data
    - The HashMap contained keys of strings and byte[] for the storage of data
    - This made retrieving keys, removing , adding data seamless

Testing

- The following is the result from the TestStringCLI program with the same inputs entered in the project pdf. The correct/expected output is acheived

```
This little client utilizes a uses an RUClient to allow
you to send and store strings within an object store.

Usage:
   connect <host> <port>
   put <key> <string>
   get <key>
   remove <key>
   list
   disconnect
   exit

> connect localhost 12345
Connecting to server at localhost:12345...
Connection established.
> put "key1" "Hello World"
Putting string: "Hello World" with key "key1"...
Successfully put key1
> put "key2" "Foo Bar"
Putting string: "Foo Bar" with key "key2"...
Successfully put key2
> list
Going to get object keys...
recieved array:[key1, key2]
Successfully retrieved keys
Object Keys: key1, key2
> get "key1"
Getting string with key "key1"...
Successfully received string.
Received string: "Hello World"
> remove "key1"
Removing object with key key1...
Successfully removed object with key key1
> list
Going to get object keys...
recieved array:[key2]
Successfully retrieved keys
Object Keys: key2
```

- The following result is achieved when running the TestSample file provided
    - The correct and expected output is achieved

```
Connecting to object server at localhost:12345...
Sucessfully established connection to object server.
Putting string "Hello World" with key "str_1"
Successfully put string and key!
Getting object with key "str_1"
Successfully got string: Hello World
Putting file "./inputfiles/lofi.mp3" with key "chapter1.txt"
placed file in server
Successfully put file!
Getting object with key "chapter1.txt"
File contents are equal! Successfully Retrieved File
Deleting downloaded file.
Attempting to disconnect...
Sucessfully disconnected.
```

- Along with these given tests, I have also tested the rest of the files in the input files and they have worked

Overall Experience

- Areas of improvement
    - User error checking
        - Not sure if I have done enough checking for user error or things like: checking if a user is trying to ask for a list of keys but the list is empty
    - I used multiple streams for a single socket
        - Although this worked in my testing, I am not sure if this was the best approach.
        - In the future, in recitations, I think it might be a good idea to go over the input/output streams
    - Added a newline character in bytes at the end of each message to tell the server the end of the command
        - I can see this breaking
- Hardest parts
    - Creating the message and extracting the correct parts of it was initially a challenge to figure out
    - The different types of streams and when to use them was confusing at first as well
        - For example: objectinput streams, buffered reader, etc.
    - Figuring out how to send the different types of data within messages
        - Files, text, commands, objects(sending list of keys)
    - Keeping the server active and listening to requests from the client