Name: Vaishnavi Manthena
NetID: vm504
Assignment: CS417 Project 1 Report

PROJECT IMPLEMENTATION

Implementation of server:
I created an RUStoreServer class that represents the blueprint of a server objects. The instance variables are a ServerSocket object (for listening on a port) and a HashMap to store the keys and the objects in the server. For the Hashmap, the datatype for the key is a String and the datatype for the object/value is a byte array. No matter whether the client is inputting a byte array or file data, it will be sent to the server as a byte array and stored as a byte array. So, the server can deal with any information as a sequence of bytes. The server object has an instance method called start which takes in a listening port as input and implements the server functionality. The main method in this class creates an RUStoreServer object and runs this start method. The RUStore server always listens on a port on the machine the program is running on.

Implementation of the start method for the server:
Firstly, binding between the ServerSocket object and input port occurs. Now a nested while loop is created. The outer while loop is responsible for accepting a new connection. Since, the server should listen forever until the program is explicitly terminated, this is an infinite loop. In this loop the server accepts a connection and goes into the inner while loop. In this inner while loop each iteration takes care of processing a single request (put/get/..). Each different request is taken care of using a switch statement. This is an infinite while loop too, but when there is a request to disconnect a break statement brings us out of the inner loop and back into the outer one which closes this connection and starts another iteration to accept another connection.

Note about accepting connections:
According to how I implemented the protocol, the server is required to read both lines of data and byte arrays. For this purpose, I needed to create two input streams and one output stream for an accepting socket.

- DataOutputStream
- DataInputStream
- BufferedReader

I used a similar approach for connecting sockets created at the client.

Implementation of client:
The RUStoreClient class serves as a blueprint for a client object that can connect and make requests to a server. The instance variables for this class are:

- **private** Socket clientSocket; //connecting socket
- **private** DataOutputStream out; //output stream
- **private** BufferedReader in; //For reading input line by line
- **private** DataInputStream input; //For reading input as bytes or byte arrays

- **private** String host; //Machine to connect to (where RUStoreServer.java is running)
- **private int** port; (port where the server is listening)

Then I implemented the different instance methods of this class by following a custom protocol.

Connect Method

public void connect()**throws** UnknownHostException, IOException

This is just creating a connecting socket and initializing the input output streams. At this point the server is at the start of the outer while loop mentioned earlier and is ready to accept a connection and initialize its input and output streams for the accepting socket.

Put Method for Byte Arrays

public int put(String key, **byte**[] data) **throws** IOException, InterruptedException

The below protocol describes the implementation of this "put" through communication between client and server.

#	Client	Server
1.	Sends the line "P\n"	
2.		This way the server knows to follow protocol corresponding to "put".
3.	Convert key to a byte array and just send its length	
4.		Reads the length of the key, initializes a byte array of that size and sends a response containing the key length itself as an ACK.
5.	Reads the ACK. Sends the byte array corresponding to the key.	
6.		Reads in this byte array and converts it to a string.
7.	Sends the length of the data array.	
8.		Reads the length of the data array, initializes a byte array of that size and sends a response containing the data length as an ACK.
9.	Reads the ACK. Sends the byte array corresponding to the data.	
10.		Reads the data into a byte array. Checks if key already exists using containsKey method. If so, sends 1. Else, inserts the key and data into HashMap and sends 0.
11.	Reads the result and outputs it.	

Put Method for Files

public int put(String key, String file_path) **throws** IOException, InterruptedException
The client converts the content of the file into a byte array using File.readAllBytes() method. Then the same exact protocol above is reused used to send this byte array.

Get Method for Byte Arrays
public byte[] get(String key) **throws** IOException

The below protocol describes the implementation of this "get" through communication between client and server.

#	Client	Server
1.	Sends the line "G\n"	
2.		This way the server knows to follow protocol corresponding to "get".
3.	Convert key to a byte array and just send its length	
4.		Reads the length of the key, initializes a byte array of that size and sends a response containing the key length itself as an ACK.
5.	Reads the ACK. Sends the byte array corresponding to the key.	
6.		Reads in this byte array and converts it to a string. Checks if this key is present in HashMap. If yes, retrieves the value (corresponding byte array) and sends its length. If no, then sends -1 and breaks out.
7.	Reads in the length. If it is -1 returns null. Else, ACKS it by sending the length back.	
8.		Reads in the ACK and sends the data array
9.	Reads in the data array and outputs it.	

Get Method for Files
public int get(String key, String file_path) **throws** IOException
By reusing the exact protocol above, the byte array corresponding to the key is returned. If the byte array is null (key does not exist) 1 is returned. Else, the bytes in the array are transferred to the file using FileOutputStream. Then 0 is returned.

Remove Method
public int remove(String key) **throws** IOException

The below protocol describes the implementation of “remove” through communication between client and server.

#	Client	Server
1.	Sends the line ”R\n”	
2.		This way the server knows to follow protocol corresponding to “remove”.
3.	Convert key to a byte array and just send its length	
4.		Reads the length of the key, initializes a byte array of that size and sends a response containing the key length itself as an ACK.
5.	Reads the ACK. Sends the byte array corresponding to the key.	
6.		Reads in this byte array and converts it to a string. Checks if this key is present in HashMap. If yes, removes it from the HashMap and sends 0 to the client. If no, then sends 1 to the client.
7.	Reads the result and outputs it.	

List Method

```
public String[] list() throws IOException
```

The below protocol describes the implementation of “list” through communication between client and server.

#	Client	Server
1.	Sends the line ”D\n”	
2.		This way the server knows to follow protocol corresponding to “list”.
3.		Checks the size of the HashMap and returns the number of keys.
4.	Reads in the number of keys. If 0, returns null.	
5.		Loops through each key in the HashTable

Protocol for each of these loops:

#	Client	Server
1.		Convert key to a byte array and just send its length
2.	Reads the length of the key, initializes a byte array of that size and sends a	

	response containing the key length itself as an ACK.	
3.		Reads the ACK. Sends the byte array corresponding to the key.
4.	Reads in the key byte array. Converts key to string and adds this string to a String array called “keys.” Acknowledges the processing of this key.	
5.		Reads the ACK

Finally, after all iterations (after all keys are taken care of) the String array “keys” is returned by the client.

Disconnect Method
public void disconnect() **throws** IOException
Here the client simply writes “D\n” into the socket and closes the input/output streams and the connecting socket. When the server reads “D\n”, it closes the input/output streams and accepting socket. Then in starts listening for new connections again.

PROJECT EXPERIENCE

Making the project better

I think the project can be improved by making the protocol simpler. Instead of sending so many messages back and forth maybe all the request information (request type, key, data , etc ..) could be combined into one byte array and parsed accordingly at the server. Then the server could encode all the response information in one byte array and send that. This process would only require one input stream as opposed to the two I used. This protocol seems to be a lot simpler and efficient in terms of the number of times data needs to transferred between the client and server. This implementation seems easier to follow as well.

Hardest Parts of implementing your project

1. Deciding on the protocol
   I spent quite some time moving back and forth between different protocols and finally stuck to the one I used. However, finally, now I think the protocol is more complicated than needed.
2. Broken Pipe Errors or Loosing Data
   - While transferring data through byte arrays I realized that the receiving side may not read all the bytes of the transferred byte array. So, I had to introduce a loop to make sure all elements of the transferred byte array are read properly.

- Sometimes I realized that the client and server may not coordinate properly and one party might send bytes. If the other party is not ready to read these bytes then the data is getting lost and the program is getting stuck. This is why I had to introduce ACK's to coordinate the timings properly. This made my protocol more complicated than anticipated.

TESTING RESULTS

Just in case needed I also added some testing results of my program.

Apart from the given test cases, the below are some of the particular cases I tested:

CASE 1:

In TestSample.java, I added extra code at the end which disconnects from the server, reconnects to it and gets the file with key "chapter1.txt". The main purpose is to check if even after reconnecting, the server is able to retrieve a file put into it in the previous connection.

Commands:

```
java -jar ./target/RUStoreServer.jar 12345
java -jar ./target/TestSample.jar localhost 12345
```

Result:

```
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/dummysite.html" with key "chapter1.txt"
Successfully put file!
Getting object with key "chapter1.txt"
File contents are equal! Successfully Retrieved File
Deleting downloaded file.
Attempting to disconnect...
Sucessfully disconnected.
Connecting to object server at localhost:12345...
Sucessfully established connection to object server.
Getting object with key "chapter1.txt"
File contents are equal! Successfully Retrieved File
Deleting downloaded file.
Attempting to disconnect...
Sucessfully disconnected.
```

CASE 2:
Here I test the same case above, however the test application is run on a different server (kill) from the RUStoreServer (iLab2).

Commands:

```
java -jar ./target/RUStoreServer.jar 12345
java -jar ./target/TestSample.jar iLab2.cs.rutgers.edu 12345
```

Result:

```
Connecting to object server at iLab2.cs.rutgers.edu: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/dummysite.html" with key "chapter1.txt"
Successfully put file!
Getting object with key "chapter1.txt"
File contents are equal! Successfully Retrieved File
Deleting downloaded file.
Attempting to disconnect...
Sucessfully disconnected.
Connecting to object server at iLab2.cs.rutgers.edu:12345...
Sucessfully established connection to object server.
Getting object with key "chapter1.txt"
File contents are equal! Successfully Retrieved File
Deleting downloaded file.
Attempting to disconnect...
Sucessfully disconnected.
```

CASE 3:
Here I used the TestStringCLI test to check the following specific cases:

- Basic put, list, remove
- List when there are no keys
- Putting different kinds of values (including whitespace characters)
- Putting a key that already exists
- Getting a key that does not exist
- Forming 3 connections one after the other to make sure previous content is retained.

Commands:

```
java -jar ./target/RUStoreServer.jar 12345
java -jar ./target/TestStringCLI.jar
```

Result:

```
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
> put "key3" "CS417"
Putting string: "CS417" with key "key3"...
Successfully put key3
> list
Going to get object keys...
Successfully retrieved keys
Object Keys: key1, key2, key3
> remove "key2"
Removing object with key key2...
Successfully removed object with key key2
> list
Going to get object keys...
Successfully retrieved keys
Object Keys: key1, key3
> remove "key1"
Removing object with key key1...
Successfully removed object with key key1
> remove "key3"
Removing object with key key3...
Successfully removed object with key key3
> list
Going to get object keys...
No available keys
> put "1stKey" "dsfhsdefufjksfcslkdfi"
Putting string: "dsfhsdefufjksfcslkdfi" with key "1stKey"...
Successfully put 1stKey
> put "2ndkey" "abc"
Putting string: "abc" with key "2ndkey"...
Successfully put 2ndkey
> get "1stkey"
Getting string with key "1stkey"...
Failed to get string with key 1stkey (key does not exist)
> get "1stKey"
Getting string with key "1stKey"...
Successfully received string.
Received string: "dsfhsdefufjksfcslkdfi"
> put "2ndkey" "123"
Putting string: "123" with key "2ndkey"...
Failed to put 2ndkey. (key already exists)
> list
Going to get object keys...
Successfully retrieved keys
Object Keys: 1stKey, 2ndkey
> disconnect
Disconnecting from server...
```

```
Sucessfully disconnected from server.
> connect localhost 12345
Connecting to server at localhost:12345...
Connection established.
> list
Going to get object keys...
Successfully retrieved keys
Object Keys: 1stKey, 2ndkey
> get "2ndkey"
Getting string with key "2ndkey"...
Successfully received string.
Received string: "abc"
> put "3rdkey" "Bar  2"
Putting string: "Bar   2" with key "3rdkey"...
Successfully put 3rdkey
> disconnect
Disconnecting from server...
Sucessfully disconnected from server.
> connect localhost 12345
Connecting to server at localhost:12345...
Connection established.
> list
Going to get object keys...
Successfully retrieved keys
Object Keys: 1stKey, 3rdkey, 2ndkey
> get "3rdkey"
Getting string with key "3rdkey"...
Successfully received string.
Received string: "Bar           2"
> disconnect
Disconnecting from server...
Sucessfully disconnected from server.
> exit
```