Robert Clabough
ComS 454
Project 2 Report

Overview

This project contains the source code for a stateless distributed system. The system contains two primary components, the server and the client.

Before getting into the server and client, we must first go over the commands package. This project uses a simple serializable command. These commands include the requires open, read, write, close, etc. These are contained as an enum value.

Wrapping this enum command value is the ServerCommand object. ServerCommand contains the command enum, along with some required information. A byte array to store data, the filename as a string, a filehandle object, and a string simply labeled 'message'.

This ServerCommand object is all that is needed to send a command between the client and the server. The creation of this showed to be tedius without creating a lot of overhead super classes. I instead opted to create the CommandFactory, a static class containing simple static methods which would return a well defined ServerCommand. As long as all arguments were valid to the CommandFactory method, this will return a ServerCommand that both sides can use.

To describe the client side then, to use the API, one only needs to instantiate the class. There is a required constructor that takes in a host and port number. These are required so that the API knows where to send the data.

After the object is instantiated, it is free to use. ConnectionAPI is the only class that outward facing clients should ever need to use. The required methods are in there, including open, read, write, close, etc. There are also some that I toyed with, but didn't have the time to actually finish up. (ReadEntireFile was to go through the entire file, opening and closing, reading multiple times without the client ever knowing that there were multiple calls.) There were also some async methods that probably do work, I could just never test.

So the client now works as planned. There were some problems getting the reads to take place at the correct spot, but from what I've seen, this has been taken care of.

The server is started by starting ServerRunner. This contains a main method which starts the SocketListener.

SocketListener runs in its own thread, always listening for a connection. When one arrives, it spins off a new thread to handle the connection. In this separate thread, it will open up the FileManipulator, which is the class that handles all interactions with files. In FileManip it breaks out the original ServerCommand, and processes the request of the command. It then creates a new reply command, and sends the new command back to the calling process.

This simple structure is mostly supported by the Serializable interface, which then is easily transported over sockets using ObjectInput/OutputStreams. Adding functions to this is as easy as adding a command, adding it to the API, and adding it to the FileManip.

Compiling

To compile the project, I have included a makefile. The client has a dependency on all commands, and the server has the same dependency.

To make everything simple, calling "make client" will create and compile the client. To run the client test, run:

```
java -cp bin client/ClientTest
```

(You may need to make sure the bin directory has been created before compiling)

The server is pretty much the same, call "make server" and it will create the server. To start the server then run:

```
java -cp bin server/ServerRunner
```

Now both will be able to run. If you prefer, there is also:

```
make startserver
```

```
make startclient
```

Both of these commands will compile and run each respective component.

Notes

My project does not follow the exact spec, the IP and port for the time being are hard coded. I ran out of time before being able to swap those over. It should work over any network to read and write files, that has been tested numerous times.

If there are any problems, feel free to reach out to me at rmcn96@iastate.edu. I would be more than happy to resolve any issues that might arise.