Project: Dropbox - Part 1

CS131: Fundamentals of Computer Systems

Thu., Feb 21, 2013

1 Introduction

In this project, you will be creating a service that is similar to Dropbox, though much simpler. You will create a server and a client, and each client will have the ability to synchronize its files to the server. Later, you will add many features to this service, such as allowing for multiple simultaneous clients or including atomicity guarantees. Part 1, however, focuses on creating the initial client and server and designing the system in such a way that makes these future additions relatively easy, using modularity or another technique as described in class.

2 Structure

The structure of the project is as follows, and is what the stencil code contains. The stencil code is located in `/course/cs131/pub/drobox-part1`.

- `FileSynchronizationServer.java`: An interface specifying the required methods for the file synchronization server. Not to be modified.
- `DropboxServer.java`: A class implementing the FileSynchronizationServer interface. Only a skeleton file, needs to be filled in.
- `FileSynchronizationClient.java`: An interface specifying the required methods for the file synchronization client. Not to be modified.
- `DropboxClient.java`: A class implementing the FileSynchronizationClient interface. Only a skeleton file, needs to be filled in.
- `SynchronizationConstants.java`: An interface containing useful constants for the project. Modify as needed.
- `dropbox-part1.pdf`: This handout.
- Of course, you may create other classes or interfaces as needed, but make sure to include them in your handin.

3 Design

The server will be able to receive files from one client at a time over a network connection, and will modify a given directory (more on this later) to match the exact contents sent to it by the client. The client will continuously sync the contents of a given folder to the server, synchronizing every so often. The client and server will communicate over a network using sockets. Note that the synchronization is only one-way here, from the client to the server. It will be your responsibility to design a protocol that the client can use to communicate with the server. It is important to note that every socket connection between a server and

client will result in one full synchronization. For a client to synchronize a second time, it must reconnect to the server, since the socket connection (by convention) only lasts for one synchronization.

4 Server Specification

4.1 Interface

Your server should follow the interface given in FileSynchronizationServer.java, which is a high-level interface. You will have to develop many of the implementation details further, such as the communication protocol.

- `listen`: This method should listen continuously on the server's socket connection for one client at a time, then listen each client and populate the Dropbox directory according to the client. It is recommended that you have a `listenOnce` method that you can call in a loop.
- `main`: The main method should call `listen`.

4.2 Potentially Useful Java Classes

- java.net.ServerSocket
- java.net.Socket
- java.io.DataInputStream
- java.io.File
- java.io.FileOutputStream
- java.io.ObjectInputStream

5 Client Specification

5.1 Interface

Your server should follow the interface given in FileSynchronizationClient.java, which is a high-level interface. You will have to develop many of the implementation details further, such as the communication protocol. Note that all of the methods that return booleans return `true` if the method executed successfully and `false` otherwise.

- `run`: The main method should start continuously synchronizing files to the client. It should call `sync` in a loop, though the client should pause between synchronizations, which can be done using `Thread.sleep`.
- `sync`: This method should read the files and folders from the Dropbox directory and send them over to the server using some communication protocol.
- `main`: The main method should call run. Your client should take the hostname of the server as an argument.

5.2 Potentially Useful Java Classes

- java.net.Socket
- java.io.DataOutputStream
- java.io.File
- java.io.FileInputStream
- java.io.ObjectOutputStream

6 README and Commenting

As you may expect, you need to handin a README file along with your project handin. This README should describe any high-level design decisions made, and should also describe the communication protocol between your client and server. The README should contain any high-level documentation you may have, and the rest of your documentation should be in the form of comments. These should serve to explain the function of pieces of code that are particularly long or cryptic.

7 Notes

- The dropbox folders that the client and server are trying to keep in sync must be in the `/tmp` directory, since the `/tmp` directory is locally stored on the machine in the Sunlab you are currently working on, as opposed to the rest of the filesystem, which is on NFS (and there would be no point in trying to synchronize it). Your dropbox directory should be named `/tmp/<login>` to prevent any collisions between students who may be working on the same machine. If you are testing on one machine, then you need to have different names for your directories, but please revert this before handing in your final code.
- For the purposes of this project, you may assume that the dropbox directory is not being modified at the same time that the client is syncing with the server, and you may assume that the files on the server will never by modified by anyone other than the server. If this happens to not be the case, your program may throw an error.
- You should change the default port that your server listens on to some other random port number. This will make sure that you do not have collisions with any other students' programs.

8 Sockets

The client and socket will be communicating over a socket connection. A socket connection is a connection between two programs over a network connection. The java.net implementation of a socket abstracts away the network protocols for you so that you can treat the socket as a data transfer channel. In the java.net package, there are two classes, Socket and ServerSocket, which are used by the client and server, respectively. The standard flow when using sockets is as follows:

- A server listens on a ServerSocket bound to a specific port.
- A client uses a Socket to attempt to connect with the ServerSocket. Note that the client should already know the hostname of the machine the ServerSocket is running on, as well as the port number of the ServerSocket.
- The ServerSocket accepts the client's connection, and now has a socket it can use to communicate with that client.

- The client now also has a socket that it can use to communicate with the server.
- The server and client can now read and write on their sockets to communicate!

For more information on sockets, and their use specifically in Java, the Oracle documentation contains more information (http://docs.oracle.com/javase/tutorial/networking/sockets/index.html).

9 IDEs

It is recommended that you use an IDE while developing with Java, as IDEs help make code editing and compilation much easier. Start a new project using your IDE of choice (Eclipse and IntelliJ are popular choices), and add the two interfaces provided to the project. For further help on using an IDE, come to TA hours.

10 Handin

To hand in your database implementation, run

```
cs131_handin dropbox-part1
```

from your project working directory. Make sure you include all .java files and your README.

If you wish to change your handin, you can do so by re-running the handin script. Only your most recent handin will be graded.