CSCD 467 Final Project Report

Nick Condos, Michael Cartwright

Idea

In this project, we decided to create a dropbox like application that ran on android devices. We took some ideas from one of the past homework assignments and applied them in this project. We also utilized our knowledge of Android application development (from CSCD 372) as well. These ideas include:

- Use Amazon Cloud Buckets to store and retrieve files.
- Using a proxy server to talk to the Amazon Buckets.
- Using a client (android) application to talk to the proxy server.
- Develop a GUI in the android application to allow a user to easily navigate the contents of, download, and upload to an Amazon Cloud Bucket.

With this idea in mind, we set out to implement each of the specified classes and objects.

Development

Proxy Server

First, we set off by designing and implementing the proxy server. In our source code, the classes that compose the proxy server are called ProcessRequest.java and RServer.java. After looking over the code we used to implement a dropbox-like application in homework 6, we adapted it to work on a local server as a standalone application.

Our implementation contains the following design ideas:

1. Utilizing a server socket to accept incoming connections (**RServer**)
    - Use built in java classes for ServerSocket. Blocks until a connection is received, and then wraps that socket connection in a thread to be ran separately from the proxy server.
2. Once a connection has been established (meaning the android application has connected to the proxy server) utilize that socket connection to process asynchronous events from the client side and pull/push data to Amazon Buckets (**ProcessRequest**)
    - While the client still wants to maintain the connection, this class busy waits.
    - Once a request has been sent from the client, process it as either upload or download.

- If upload:
    1. Grab corresponding streams.
    2. Using grabbed stream, create a temporary file to send up into the Bucket.
    3. Put that file object in the specified Bucket.
- If download:
    1. Retrieve an Amazon S3Object from the bucket specified.
    2. Create a temporary file to hold the contents of the incoming S3Object
    3. Copy the contents of the object into the temporary file.
    4. Write the contents of that file to the socket output stream that maintains the connection to the client.

From here, we had the glue of the client-server paradigm implemented and we proceeded to implement the client-side (Android application).

Android Application

After developing most of the proxy server, the goal was to create a functioning GUI to interact with the proxy server and consequently the Amazon Buckets. The source files for the client side are MainActivity.java, MyListAdapter.java, ClientConnect.java, and RainFileObject.java. The concepts and ideas we took from our previous Android class and adapted to this problem space is as follows:

1. Using list adapters to populate and change list views (**MyListAdapter**)
    - Create a custom list adapter class to manage and the list views that is apart of the main layout of the application.
2. Connect to the proxy server by instantiating a thread that connects and sends requests to the server (**ClientConnect**)
    - While the client still wants to maintain the connection to the server, process various actions instigated by the main layout of the application.
    - If we are uploading a file:
        1. Using the specified bucket, write the contents of the local file to the existing output stream maintained by the socket connection to the proxy server.
    - If we are downloading a file:
        1. Create a file object to write into from the existing socket's input stream.

2. Write from the input stream into the new file.

3. RainFileObject is used as a way to encapsulate the information about a bucket and the bucket's containing files. (**RainFileObject**)

   - MyListAdapter class uses a LinkedList of RainFileObjects to populate the expandable list with the parent list (bucket name) and the child lists (files in each bucket)

4. Drives the above classes and the application as a whole via a main activity (**MainActivity.java**)

   – Instantiate the layout, views, and the corresponding handling source files (like MyListAdapter).

   – Connect to the proxy server and maintain that connection while the application is alive.

   – Create methods that handle selection of files and whether the user is intending to download or upload a file.

Problems

The development of this application as a whole was not entirely smooth. We ran into a couple of problems that really caused us to sit down and think about what we were doing, how we were doing it, and what could possibly be going wrong.

- Our initial plan was to research Amazons "Amplify" for Android Programming which would allow us to connect an Amazon S3 object directly from the android device to Amazons server. We ran into some complications here when realizing that this framework required us to have knowledge of other Amazon services that we hadn't learned. So instead, we decided to download and add all the JAR files to our project that are necessary to use Amazons S3 service. This unfortunately didn't work because every time we'd build the project after adding one JAR as a dependency, the project would break saying it was missing an object of some type. We continued adding more JAR dependencies until we had so many that project looked like a botched surgery. So, we decided to build a proxy server so Android Studio never had to directly talk with Amazon services and we would leave that job to the proxy server in eclipse that already had the AWS toolkit hooked up.

- Another big issue we ran into was sending/receiving bytes from the buckets to the proxy server, and then from the proxy server to the client. Initial implementations would either have extraneous bytes appended or prepended to the actual file contents, or the server/client would not send any bytes at all. Finding the solution to this was a time consuming effort, as it appeared like we were implementing/utilizing byte streams and input/output streams correctly.

Eventually, we settled on using temporary files to act as a medium for the bytes coming from a bucket that needed to be sent to the client.

Reflection

If we were to do it again, we would maybe explore not constantly keeping a socket open while the client is up and running. Instead, we would maybe only instantiate a connection to the proxy server when the client actually needed to download or upload a file. This reduces resource consumption and makes sending and receiving bytes via the streams easier.

Another thing we wished we had time to do was put the proxy server on an Amazon EC2 instance. This way, we would be able to connect to the proxy server from where ever we wanted to and not have the state of the server tied to whether it was running on a local machine or not.