Project: Dropbox - Part 3

CS131: Fundamentals of Computer Systems

Thu., April 18, 2013

1 Introduction

In, the third part of this project, you will modify your program to meet some ACID standards. As covered in class, ACID is a set of guarantees used normally in the context of database systems. Nonetheless, these guarantees can be applied to most computer systems. For this project, we will be modifying our program to meet these standards as specified below. These guarantees will help define the behavior of your program if it crashes.

2 Carrying on from Part 2

We will not be grading your functionality from part2 of this project when grading part3, although you should make sure that your part2 implementation has no fatal bugs, as that will make distinguishing your part3 bugs from your part2 bugs harder.

3 Specification

Since it has not been formally specified, here we will formally specify this: you should assume that no one can and will modify files on the server, so the only way to make changes on the server is for your program to propagate changes from clients that it is connected to.

As far as our project is concerned, we will be applying these ACID guarantees in such a way as to provide some guarantees if the server or client crashes. Note that when considering the cases below, there are two cases: either the client/server can crash while operating, or the client/server will be operating and will have its communication with the server/client interrupted because a server/client crashed.

3.1 Atomicity

If the client or server crashes while receiving or sending a file, the file that was being modified should not be left in a half-modified state. This applies to both the client and server equally. File updates should follow an all-or-nothing model, such that a file is either fully updated or fully untouched. This is best done by using a temporary file and atomically renaming it to the desired file, as that will take care of the case where the receiver of the file crashes before it fully receives the file's contents.

3.2 Consistency

We will define consistency on a per-file basis, so consistency in this case is taken care of by our all-or-nothing semantics for atomicity. Since files are either updated or untouched, the server's files and the client's files should remain consistent at all times given that the atomicity guarantee is provided.

3.3 Isolation

By locking files before they are modified on the server, we have already guaranteed that updates to files from different clients are isolated. One important thing to make sure of is that having a client crash or having the server crash does not break your locking scheme. This especially applies if you are using `FileLock`, since if the server crashes while holding a `FileLock`, that file will remain locked since that class uses the `flock` system call.

3.4 Durability

If the client or server crashes, then upon resuming it should be able to resume synchronizing where it left off. To do this, the client and server must store some persistant state on the hard disk. It is not necessary for the client or server to pick up exactly where it left off (if it crashes halfway through synchronizing a file, for example), although that would certainly be nifty if it could. However, the server and client should be able to resume synchronizing as before, though, with files that are in conflict remaining in conflict, etc., as if no crash had happened.

4 README and Commenting

You will have to update your README for this part of the project, again documenting any high-level design decisions made and any changes made to your program's structure. Please also document how your program may differ because of any bugs in your part2 implementation, if it does at all.

5 Handin

To hand in your database implementation, run

```
cs131_handin dropbox-part3
```

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.