Where’s the file?

Where’s the file(WTF) is a project in which two programs, the server and the client, communicate to each other to maintain a version control system. The overall practical use of this project is to allow multiple people to work on the same version of code at once. As people update the code, the version number increments, and no other contributor can commit their code until they have the same version number as the current one. Not only can users update the project, but they can also rollback to previous versions in the repository.

How to run

First call “make all” in the WTFserver directory to make the executables. The server executable is stored in the server directory, and the WTF executable will be in the client directory. While inside the WTFserver directory, call

- ./server/WTFserver <port-number>

Then you can run the client executable using

- ./client/WTF <parameter>

“Make clean” should then remove the executable and object files, and “make test” should build all the files needed to run “WTFtest”. To run WTFtest, call

- ./WTFtest <port-number>

File Manifests

The .Manifest files purpose is to allow the changed files being uploaded to the server to compare versions more efficiently without having to compare entire files byte by byte. These manifests were formatted in order to make comparing versions easier, I used SHA to create the digest, more specifically SHA256. In addition, the manifest version will only increment when a successful push occurs.

The general .manifest file will look like:

<File version number>

<File version number> <flag> </path/filename> <hashcode/or DELETE>

Things to keep in mind is that the second line will repeat for as many files as there are and the legend for <flag> is, !AD=added, !RM=removed, !UT=untracked, and !MD=modified.

Networking and Thread Synchronization

First, The WTF client will always expect a response from the WTF server after a message is sent. In addition, the server always generated a thread specific to the connection requested it received from the client, which allows the server to handle multiple clients simultaneously.

To achieve thread synchronization/multithreading, the WTFserver would start with a pthread array of size 20 and spawn a new client service thread whenever it gets a new connection request. When the pthreads approached the limit, the threads are joined together and the counter is reset to accept new client connections.

Since the chance of multiple threads accessing the files in a directory at the same time is possible, I used one mutex per repository to control access to it. The mutex would lock whenever a thread was reading/writing from or to a project. For projects, a LL of mutex nodes was uniquely created for each one, and whenever a new project was created, a new mutex would also be created as well specific to that project. Deadlocks were prevented by forcing an unlock before exiting the thread.

Authors

Chenghao Lin – cl1141

Systems Programming, Spring 2020