John Abrams (jsa109)
Chris Zachariah (cvz2)
CS 214 – Systems Programming
Prof. Francisco
Asst3 – Where's the File? (WTF)

README

General Overview:

Both the server and the client use the command line arguments from the user to figure out which command the client wants to accomplish. The command is then stored as a char and passed into a switch statement that handles each case accordingly. The server takes each new client and makes them a thread. The thread is then stored into an array of threads where each one then will be taken care of before thread_join(thread[index]) is called.

When each thread is made passed through the doSomething function, the server will communicate with the client accordingly to proceed and complete the indicated command. Mutex lock and unlock is used once the thread is created and the doSomething function is called.

The server, once it creates a socket and checks to make sure that the port number is valid and can be used to make new client connections, will go into an infinite loop where it will listen for new clients in order to make them into threads and handle them accordingly.

Client Commands:

Configure:

When the client runs this command, the client checks to make sure that the IP address and the port number (the other arguments that are passed through the command line) are valid and can be used with the socket to connect to a server. The information is then copied into a .Configure file that is in the client's home directory that will then be accessed by other client commands in order to connect to the socket.

Create:

When the client runs this command, the client will first try to find the .Configure file in order to use the IP
address and the port number in order to connect to the server. Once connected, the client will send the server the "E" command

which will indicate to the server that the client wants to create a new project in the repository.

The server then is ready to receive the name of the project that the client wants to make. Once receiving the name, the server checks its repository to make sure that there is no other project with the same name. If the name is okay to use, then the server will make the project directory and a .Manifest that starts at version 0.

Finally, the server's newly created .Manifest file is sent over to the client that will then make the same project directory and then place the copy of the .Manifest inside it.

Currentversion:

When the client runs this command, the client will first try to find the .Configure file in order to use the IP address and the port number in order to connect to the server. Once connected, the client will send the server the "V" command which will indicate to the server that the client wants a copy of the .Manifest of a specific project.

The server waits for the client to send the name of the project before taking the necessary steps to find the project directory and then sending over a copy of the .Manifest. The client then reads through the .Manifest and prints to STDOUT its contents (file names/paths and the version number).

Commit:

When the client runs this command, the client will first try to find the .Configure file in order to use the IP address and the port number in order to connect to the server. Once connected, the client will send the server the "M" command which will indicate to the server that the client wants to make a .Commit that will tell the server which files need to be added or modified to a newer version.

The client will first check for a .Update file in the specified project directory to make sure it is empty (if the file doesn't exist, that is fine).

Then the client will ask for a copy of the server's .Manifest for the specified project. Next the version numbers of both the .Manifest files of the server and client are checked to make sure they are same. If they are not the same this indicates that the client needs to update with the server before making any new commits.

The client then goes through each of the files found its own project directory and makes a fresh rehash to compare it to the hash found in its own .Manifest file. If the hash does not match, then the client will check the .Manifest of the server to see if it can match the hash. If the server has the file and the hash does not match with the fresh new hash, then the file name will receive a “M” command (modify) that will be written into the .Commit file. If the server does not have the file in its project directory, then the file will be given the “A” command (add) which will be written into the .Commit.

Add:

When the client runs this command, it first checks to make sure that the specified project directory exists (on the client side). Next, it checks to make sure that the specified file (to be added) exists (also on the client side), looking under both <projectname>/<filename> and just <filename>.

Then, it checks if the .Manifest (on the client side) exists. If so, it lastly checks if the file already exists in the .Manifest. If not, the client generates a 64 character SHA-256 hash based on the contents of the file and adds it to the Manifest along with its correct filepath and a version number of 0.

Remove:

When the client runs this command, it first checks to make sure that the specified project directory exists (on the client side). Next, it checks if the .Manifest (on the client side) exists. If so, it lastly checks if the file exists in the .Manifest. If so, the client generates a new .Manifest that excludes the entry corresponding to the file for removal.