Asst 3 ReadMe WTF

Assignment 3 has to do with File commands and implementing our own version of git. It is designed to handle multiple clients through multi-threading. Each project in the server is a critical section that is protected by mutex locks.

Global structures used to protect the project. In order to handle multiple threads, the socket from the client is stored in a global array protected by a mutex. A structure called ProjectNode* is a global linked list on *server.c* that matches a mutex lock with each project. The mutex is acquired every time a project is accessed (in each WTF command in the client). This ensures that no two clients access the same projects. The ProjectNode* linked list is also protected by a mutex when clients try to initially access a project.

Brief Overview of Source Files

WTF Assignment produces 4 source files.

- **server.c** : produces the WTFserver executable and handles all server-level operations.
- **client.c**: produces the WTF executable and handles all client-level operations.
- **fileHelperMethods.c**: functions created to support file, socket, and compressing operations
- **structures.c**: functions and structures used to keep track of information in server.c and client.c

How WTFserver and WTF work

WTFserver runs with a terminal command and takes in the port number. WTFserver only closes when it receives the signal SIGINT.

WTF runs with multiple terminal commands depending on the client command. It exits naturally after the command is run.

The client and server communicate through TCP sockets. The client runs a command

Both the server and client manages a project directory file system where each project is broken down into two folders: the project folder and the project backup folder.

The project folder maintains a Manifest file and a History file. The Manifest keeps tracks of the current files in the project directory. The History file keeps track of all successful pushes.
The backup project folder maintains all older versions of the project compressed in a tar file.
The client calls one of 12 commands at a time for a single project to either update the client-side project directory, the server-side project directory, or output information.

All files are compressed in a tar file before being sent over to each socket.

Command Summary and Efficiency Analysis

There are 12 main client commands that are run.

- **configure:** configure takes in the IP address and a port. If valid arguments were passed, the client then makes a .Configure file in the client's root directory for future connections.
- **checkout:** checkout takes in a project name as an argument. The client connects to the server. The Server then takes the project folder from the Server, tars it, then sends it over to the client, then deletes the tar file. The client untars the project folder and deletes the tar file and stores it in the client's root directory. The client then makes a backup directory if successful for future operations.
- **update:** update takes in a project name as an argument. The client connects to the server and then receives the server's Manifest file. The client compares its Manifest to the server's Manifest using a linked list. Then update writes to an Update file of all valid update commands (reference project description)
    - Once a node is compared, it is deleted.
      Searching a linked list takes O(n) time, where n represents the number of files. Note that because after every search, a node is deleted, the runtime is slightly more efficient. However in the worst case, no common files are found, therefore the worst case, search and delete would be O(n^2).
- **upgrade**: upgrade takes a project name as an argument. The client then connects to the server and the server sends all necessary files to the client. The client then applies all the changes in the Update file to the project. Then the client's manifest file is replaced with the server's manifest version. Then it notifies the server if it was successful.
- **commit**: commit takes in a project name as an argument. The client connects to the server and then receives the server's Manifest file. The client compares its Manifest to the server's Manifest using a linked list. Then commit writes to a

Commit file of all valid commit commands (reference project description). Then, on success, the Client sends over the commit file, and the server moves the commit file into a directory of all pending commits. Each commit is renamed with the commit number to avoid collisions.

    - Once a node is compared, it is deleted.
      Searching a linked list takes O(n) time, where n represents the number of files. Note that because after every search, a node is deleted, the runtime is slightly more efficient. However in the worst case, no common files are found, therefore the worst case, search and delete would be O(n^2).

- **push:** push takes in a project name as an argument. The client then connects to the server and then the client sends all necessary files to the server. The server then finds a matching Commit file in the client and then applies all necessary changes to the server. The server then replaces its Manifest, sends it over to the client, then updates the history file on success. It notifies the client on success or failure. The client deletes its Commit file regardless.
- **create**: create takes in a project name and connects the client to the server. Create on the server side creates a project directory, a project backup directory, a Manifest file, and a History file. The client makes its own project folder and project backup folder and sends over the Manifest file to the client.
- **destroy** destroy takes in a project name and connects the client to the server. Destroy completely removes a project directory along with all the files contained in it along with the backup project folder. It sends a signal to the client if it was successful or not.
- **add** add takes in a project name and a file name. It adds a file to the Client's Manifest given if the file exists in the project file. Add is also unable to add any duplicates to the Manifest if it already exists.
- **remove** remove takes in a project name and a file name. It removes the file line from the Client's Manifest given if the file doesn't exist anymore in the project file.
- **currentversion** currentversion takes in a project name and connects the client to the server. takes information from the server's Manifest and gives it to the client, then the client outputs the file name and the version number.
- **history** history takes in a project name and connects the client to the server. It tars the file and sends over the information to the client. Then the client prints out the History file.
- **rollback** rollback takes in a project name and a version number. Then the client connects to the server. The server searches through the backup folder and searches for the version number. It sends back a signal to the client if it failed to find a version number. On success, rollback replaces the current project in the

server with the project version number and deletes all stored versions higher than the version rollback is replacing.

Note that other than update and commit where we compare two Manifest files. Otherwise, WTFserver and WTF do basic string operations and searches. In most cases, these operations are some constant times n where n is the number of files/ the number of lines in a file. Space is maximized through the use of compressed tar files and everything is freed after use.