Version Control

Version Control is a program for allowing multiple clients to edit projects stored in a server.

Description

Version Control is a program that consists of a server and clients. The server holds projects that the client can interact with by changing the projects in a manner that always changes the most recent version. Once the client configures the hostname/ip address and port number to connect to, all of the other commands can connect to the server to request and possibly modify resources. After creating a project, clients can destroy it, add files to it, or remove files from it. Another way to get a project into a client is through checkout. Pushing the code to the server is a 2-stage process, commit and then push. This is only possible if the client is confirmed to have the latest version. The opposite of this is when clients pull from the server, which is also a 2-stage process, update and then upgrade. Finally, the client can request the current version of the serverside project, the history of pushes to said project, and roll back to an earlier version.

Usage

Server

```
 ./WTFserver <port number>
(client connects, executes commands, disconnects)
^C (SIGINT)
```

Client

```
 (once server has started running...)
./WTF configure <hostname or ip address> <port number>
./WTF create <project name>
./WTF destroy <project name>
./WTF checkout <project name>
./WTF add <project name> <file name>
./WTF remove <project name> <file name>
./WTF commit <project name>
./WTF push <project name>
./WTF update <project name>
./WTF upgrade <project name>
./WTF currentversion <project name>
./WTF history <project name>
./WTF rollback <project name> <version>
```

Project Design

Both the client and server were designed to be modular and independent. Functions they both share, such as sending file functionality and error checking, were put into a utilities file. Both the client and server have independent functions for each possible command. In the client, the functions are called from the main function after parsing the argument. For the server, the main function handles the threading and passes new connections to a request handler, which then parses the code sent by the client to direct it to the correct function. For every message that the client sends, the server will acknowledge it and vice versa. To uniquely identify every client for commits and pushing, we generate a unique id per client upon their first connection to the server, and use that id in their commits as a unique identifier.

Implementation

Each of the functions that interact with project contents contains a critical section where the mutex for said project is locked until the function completes. Then inside the function, it connects to the client and receives whatever information is necessary for the function within a packet structure that we define. If it encounters any errors, it can send error codes back to the client so that the client knows what error prevented the operation from executing to completion. The server can send files back to the client in a compressed form, into a directory we named _wtf_dir where it is decompressed.

Thread Synchronization

The server begins by initializing an array of mutexes, one for each project directory found in its repository. Whenever create is called, it also adds to the array of mutexes. We use global ordering to ensure that no deadlocks occur within our code. This means that when we lock in a certain ordere, when we unlock we unlock in the reverse order we locked, and do not lock again until after all locks are unlocked. Then, for each function that interacts with a project's contents, the mutex for said project is locked and then unlocked after the function completes. This way, if another thread tries to call a function that interacts with the same project that is already being interacted with by another thread, it is blocked from continuing. Since mutexes are guaranteed to resolve in order, the next thread that asked for the mutex to edit the corresponding project will be the next one to lock the mutex and execute a function involving it. This system guarantees that 2 threads never interact with a project at the same time. Note: Destroy does not delete mutexes because all the functions first check if the project exists before locking a mutex, so a deleted project's mutex will never be locked again.

Protocol

We designed a protocol for ease of communication between the client and server. The format is as follows: `<character code><#args>< ><arg 1>< >....<arg`

`n>< ><tarlen>< ><tar>`. Essentially, the single character code corresponding to the specific command comes first. Next, we send the number of arguments, space terminated. Then, we send the arguments (usually project name or version number) which are all space terminated. For sending files, we always put all the files we need into a directory, compress and tar it, and send that over. Thus, we send the size of the file space terminated, and then send the contents of that tar. If tar length is zero, then we know there is no file being sent. Using this protocol, we are able to easily send arguments or files from client to server with minimal overhead because we know the length ahead of time. While the meta data is read one at a time, the contents of the file are read in chunks of 4096, greatly increasing the speed.