NAME

WTF.c

This program is a version control system, where multiple clients can edit and access an online repository located on a server

SYNOPSIS

This program has 13 commands that take either 3 or 4 arguments. The first argument is the file ./WTF the second argument is a word that specifies which command you want to perform (configure, checkout, etc…) the third argument is typically the project name on your repository the only exception is for the configure command where the third argument is a IP, and the fourth argument is needed is either the name of a file, a version number, or a port. An example of the use of one of these commands is below:

./WTF add project test.txt

NOTE: Details on multithreading and mutexes are at the end of the doc after all the commands

NOTE: The server should be run on one of the ilabs at rutgers ex: ilab1.cs.rutgers.edu and the server port should be between 5000 and 65000 (this range was said during one of our professor's lectures). The server will run until the user hits ctrl + c on the server side.

NOTE: for all the .files (.Manifest, .Commit, .Upgrade… etc) we formatted the entries for a file such that it looks like this:

<filepath> \t<status flag>\t <version number>\t <hash>
./proj/file1.txt A 3 123456abdfea1234435

This differs from the faq because the faq was made after we started coding and an instructor said on piazza that we could make custom entries which were most useful to us. We set it up this way so that it would be easier to have access to the file paths as it was the first entry, which helps when comparing different manifests and .files. We use the flags “A” for add, “R” for remove, “S” for synchronized(when a file remains the same), and “I” for ignore(ignore occurs when a file is added then removed before a push). These flags tell the status of the entry through what operation was performed on the file.

Make:

We have several make commands:

make - creates all the executables needed to run the code as well as intializes the repository and the backups folder in the repository
make clean - deletes all the executables created in make
make test - creates a test project called “test” and runs operations on it
make cleantest - deletes all the general executables and the testing files created by “make test”
make cleanrepo - deletes all the executables, the test files from make test, along with the whole server repository(can be used to return the Asst3 file to its original form)

Structure:
Our online repository folder is where all the projects are however there is a special folder called backups. When the project is created in the repository a corresponding folder with the same project name is created in the backups folder. This folder contains a .History file specific to the project along with older compressed versions of that project.

Below is a list of the commands look to the description portion for more detail on what each command does.

1. ./WTF configure <IP/hostname> <port>
2. ./WTF create <project name>
3. ./WTF destroy <project name>
4. ./WTF add <project name> <filename>
5. ./WTF remove <project name> <filename>
6. ./WTF currentversion <project name>
7. ./WTF checkout <project name>
8. ./WTF commit <project name>
9. ./WTF push <project name>
10. ./WTF update <project name>
11. ./WTF upgrade <project name>

12. ./WTF history <project name>

13. ./WTF rollback <project name> <version>

DESCRIPTION

This program is a version control system used to track changes made to files (similar to the online program Git). This program is a server containing a repository, which contains project files, along with all of the changes made to these project files. Multiple clients can access and edit the code on this repository by connecting to the server and using a system involving commits, pushes, upgrades, updates, and several other commands. Mutexes are utilized to ensure that multiple clients are not editing the same project at once.

./WTF configure <IP/hostname> <port>

The **configure** command will connect the client to the server at the given IP/hostname and port. The IP/hostname and port will be saved in a .configure file which is in the root directory of the repository.

./WTF create <project name>

Create initializes a project on both the client and server side with <project name>. The created project on the server side will create a .Manifest and send it to the client. The .Manifest will be stored in the <project name> directory on the client side

./WTF destroy <project name>

Destroy removes the entire project with the name <project name> on the server side only (not the client side) along with all the associated history and backup files.

./WTF add <project name> <filename>

Add adds a <filename> to the manifest with a version number 0 and a hash code. The status flag should be set to ‘A’ for add.

./WTF remove <project name> <filename>

Remove sets the status flag of the <filename> entry in the manifest to ‘R’ to signify removed

./WTF currentversion <project name>

Currentversion requests the server’s version number of the current project. It will also print all of the files in the current project along with the file’s corresponding version number.

./WTF checkout <project name>

Checkout clones the current version of the project on the server side to the client.

./WTF commit <project name>

Commit prepares and finalizes the changes on the client side so that they can be pushed to the project on the server side. A .Commit file is created with entries that contain the file path, status flag, file version number, and a hash code. It is similar to the manifest and an example of an entry is below:

project/test.txt M 0 1233424521341314abcdeff
<filepath> <status flag> <version number> <hash>

The hashes in the manifest are compared to the live hashes of the files and if the hashes are changed then the flag is changed to ‘M’ for modified and the old hash is replaced with the live hash. All of the changed entries are then displayed in stdout under the format “A <file path>” , “D <file path>”, or “M <file path>”. For Added file, Destroyed file, and Modified file. When Commit is called the project locks, so that other clients can not edit the project while a change is being committed.

./WTF push <project name>

Push is when all the changes are applied to the server. The client sends its .Commit to the server and if the server has a matching .Commit file. If there are other .Commit files pending, they expire, so that other clients can not push at the same time. Push updates the new project and creates a new manifest that corresponds to the new project. The .Commit file is then erased. When push occurs the client’s manifest is replaced by the new server manifest of the current project created by the push. The previous current version is then saved as a compressed file in

the backups folder of the server directory and the project's history file is updated with the successful push's changes.

./WTF update <project name>

Update gets the server's manifest and compares the entries between the server's and client's manifest to see the changes. If there are changes a .Update file is created (similar to the commit) with the added, modified, or removed files. These changes are printed to stdout in the same format as in commit. If the manifest of the client has a file whose hash is different from both the live hash and the server's has for that file than a .Conflict file is created with the <file path> <status flag 'C'> <version number> <live hash> and "C <file path> " is printed to STDOUT.

./WTF upgrade <project name>

Upgrade applies the changes in the .Update file to the client's local project. It will remove the files with a 'R' flag, add files with an 'A' flag and overwrite the files with an 'M' flag. Then it deletes the .Update file. If no .Update file exists then the program will ask the user to perform the update command.

./WTF history <project name>

History makes the server send over a file containing the history of all operations successfully pushed to the project since its creation. It will contain the version number of the project followed by the changes committed in the successful push

./WTF rollback <project name> <version>

Rollback reverts the current version of the project back to the previous version number specified and deletes any backups that have a higher version number than that version number. In the history a note is made whenever a rollback occurs with the version number that the project was reverted back to.

Multithreading:

Our server has an infinite loop that constantly is accepting new commands from the client. When a command is received the server creates a pthread using pthread_create to handle the command. The client handler function(the void*(void *arg) function parameter of pthread) is where we then process the command sent by client and perform other function calls whether it be push, commit, etc… When the pthread is created it's id is stored in a linked list, so that we can keep track of the

threads. When the thread finishes and leaves the client handler function the node is removed from the linked list. If the main thread is terminated using ctr + c and threads still remain, then we use a sig handler and atexit() to call a function that iterates through the linked list and joins the remaining threads.

Thread Synchronization:

Everytime we create a thread we pass pthread_create a void function. In this function whenever a thread needs access to the server project we lock the specific project being accessed with a mutex lock. This prevents multiple threads from editing the server's project at one time. This is so the changes in one client can not get overwritten by another and cause errors. We prioritize first come first serve, so the first thread to access the server and use a function that requires server access locks it and prevents other threads from accessing it until the thread that locked it has finished. The project is then unlocked at the end of the void function which is when the thread has finished, this way other threads can access the project after it is done.