wtfclient.c

wtfserver.c

WTF

wtfclient.c - a program that handles communication with server
wtfserver.c - a server that can connect with multiple clients at a time

SYNOPSIS

./WTFServer <port number>

./WTF configure <IP/hostname> <port number>
./WTF create <project name>
./WTF checkout <project name>
./WTF add <project name> <file name>
./WTF remove <project name> <file name>
./WTF destroy <project name>
./WTF update <project name>
./WTF upgrade <project name>
./WTF commit <project name>
./WTF push <project name>
./WTF rollback <project name> <version number>
./WTF history <project name>
./WTF currentversion <project name> <version number>

BRIEF OVERVIEW

The files in this folder are used to create a simplified version control system. We have completed all the requirements for this project as well as implemented the "Future Work", specifically compressing old versions of projects in the repository. The extra credit implementation is documented in 'Push' and 'Rollback'.

INSTRUCTIONS TO RUN SERVER AND CLIENT

In order to make all executables,
- Run 'make' in the same directory as the Makefile.
- The server and client executables are made in the server and client folder respectively

In order to start the server,
cd server
run 'WTFServer <port number>'

In order to start a client,
cd client
run 'WTF <command>'

WTFTEST

The make file has a WTFTest target in it which compiles test.c and builds executable ./WTFTest. This file goes through several (not all) testcases for the WTF project. There are two sequences that testers can choose from

1 or 2. Each sequence is a series of commands that rely on the previous result and test different functionality of the program. For each command, it will display both the Server and Client STDOUT which will be differentiated by [SERVER] and [CLIENT]. It will also display the command being run, expected output, and then the output of the command. The sequences for the test cases are outlined in testcases.txt. Test.c first starts the server, then runs a client command and lastly kills the server process.

Note: In order for WTFTest to run correctly, Asst3 is in the machine's home directory.

HOW TO RUN:

make WTFTest

./WTFTest <host> <port> <sequence # [1 or 2]>

Example: ./WTFTest java.cs.rutgers.edu 8000

DESCRIPTION OF DESIGN CONCEPTS:

Protocol

Our protocol is formatted so that the total number of bytes of the command is given followed by command.

Example: 19:create:8:project1

19: the total number of bytes of the whole message

create: the command that should be executed

8: the number of bytes of the project name

project1: the project to be created

Threading

Pthread_create is called for each client that connects with the server. The threads are all contained in an array, and each pthread is given an id that is incremented with each thread that is created. When one thread terminates, it joins on all the other threads and waits for them to all finish. On success, returns nothing. Otherwise, prints an error to STDOUT.

Mutexes

Mutexes are initialized and stored in a struct that maps a project name to a particular mutex. Each project is locked with a different mutex in order to allow multiple projects to be modified at the same time. The mutex is used locked when destroying a project and pushing a project. In order to prevent deadlock, the mutexes are not only locked and unlocked once. On success, prints nothing. Otherwise, prints an error to STDOUT.

DESCRIPTION OF WTF COMMANDS IMPLEMENTATION:

Configure

Creates a .configure file that the client will later use to establish a connection to the server. The validity of the arguments will not be checked until another command is run where the connection is established to the server.

Create

The create command makes a new project with the given name in the server and client repository. The new project contains a .Manifest file with the project version set to 0. It also contains a .History file which is empty. On success, prints a message to STDOUT. Otherwise, prints an error.

Checkout

Checkout makes a copy of a project from the server onto the client repository. On success, prints a message to STDOUT. Otherwise, prints an error.

Add/Remove

The add command adds a given file to the client manifest. The remove command deletes a file from the client manifest. For add and remove, if the file does not exist in the project directory, it will error. For add, it will also error if it is already in the manifest. For remove, it will error if it is not in the manifest. On success, prints a message to STDOUT. Note that remove and add do not create or remove any files from the project directory. It only modifies the Manifest.

Note: Add takes in a path relative to the project directory

E.g. ./WTF add proj0 somefile.txt

Destroy

Deletes the project from the server repository. The destroy function is locked by a mutex so that no other client can destroy it at the same time. On success, prints a message to STDOUT. Otherwise, prints an error.

Update

The update command creates an .Update or .Conflict file showing the changes that were made in the server. Update fetches the file from the server and compares every entry in each file. If there is a conflict in the file at all, it will output all modifications to the .Conflict file. If there are no conflicts, it will output modifications to the .Update file. Update compares the manifest version numbers if they are the same it will have an empty update file. Before comparing hashes of each file, Update compares the version numbers which have to be different in order to have an update.

Upgrade

The upgrade command modifies the client project to reflect the changes in the server. If there are conflicts, it will ask the user to resolve them. It creates and writes to a file if the server has new files. It overwrites files if the server has modifications. It will delete from the manifest (not the actual directory) if the server deleted the file. It

will sync version numbers of both the manifest and files as well as the hashes.

Commit

The commit command creates a .Commit file showing all the changes the client has made. The server creates a folder named pending commits and stores a commit for each client that commits. Each commit file has the format commit-<ip address>. If multiple commits for one client are made, each commit is appended to the file and overlapping commits are deleted. On success, prints the files that were added/removed/modified, as well as,a message to STDOUT. Otherwise, prints an error.

Push

The push command modifies the server project to reflect the changes in the client. Push creates a history folder named 'history' in the current working directory that stores all the previous pushes. Each previous version is tarred using system calls. The history folder is organized by version number.

Example: history

```
|---- project-0.tar
|---- project-1.tar
|---- project-2.tar
```

Push is locked by a mutex in order for one client to make changes to the server at a time. The pending commit for the current client is grabbed and the changes are applied to the server project. On success, prints a message to STDOUT. Otherwise, prints an error.

Rollback

The rollback command restores the project to an older version. It traverses through the history folder created by push in order to find the project version with the requested version. The project with the corresponding version number is untarred. The original version is renamed with <project name>-<"old">-<number>. This project can thus be ignored.On success, prints a message to STDOUT. Otherwise, prints an error.

History

The history command creates a log of modifications at each push. If a rollback occurs, it will have the history of the older version of the project.

Example log:

```
A file1.txt <hash>
D file2.txt <hash>
M file1.txt <hash>
```

Prints the history log to STDOUT on success.

Current Version

Gets the current version of the project and versions of files by reading the first line of the manifest file. On success, prints the current version to STDOUT. Otherwise, prints an error.

SIGINT Handling

SIGINT (CTRL-C) is caught using a signal handler which closes the socket, terminates all the threads, and destroys all the mutexes that were made.

WRITTEN BY

Tanvi Wagle (tnw39) and Ivy Wang (iw86)