TEJAS SAMEERA | SHRAVAN PATEL

ASST LAST: WHERE'S THE FILE?

USAGE

Make File Implementation:

Run the Client and Server Executables:

```
make
WTFServer <port number>
WTF <args>
```

Run the Test executable:

```
make  test
(NO OTHER COMMANDS REQUIRED)
```

Clean executables and Test folders:

```
make clean
```

Function Specific Implementation (only for add and remove):

add: <"add"> <project name> <File path : ***from project name***>

remove **<**"remove"><project name><File path : ***from project name***>

EX 1:

Project Name: Test

|— File name: test1.txt

File path (from project name): test1.txt

EX 2:

Project Name: Test

|—(Subdirectory) A1

|— File name: test1.txt

File path (from project name): A1/test1.txt

THREAD SYNCHRONIZATION REQUIREMENTS

The multithreaded server, upon execution should not result in race conditions. To take this into consideration, we first create a table struct that acts as a linked list of project mutexes that are created / accessed whenever a project is being called for use. At any time, when a client wants to use the project, the server must first fetch the lock for that project and then proceed. The Linked list struct contains attribute fields of a pointer to the next project mutex, and the project name.

However, to prevent two clients from editing the linked list at the same time (whether that be adding/ removing project mutexes) we implemented a List lock mechanism that allows one client at a time to edit the linked list at a given time.

To keep track of all projects there exists a mutex counter and a mutex position counter. Initially per project, the mutex position counter is -1. Once the project has been initialized inside the mutex linked list, it will be given a position for later reference.

When a client runs a command interacting with the server, the project name associated with that command is sent over. Once the server receives the project name, it searches the linked list for that project and returns the position (place in the linked list) for that specific project. Then using an array of locks, the lock associated with that position in the linked list is locked. The server then runs the command and unlocks it afterwards.

This ensures that at a given time only one client can make changes/access information pertaining to that specific project. Any other client that wants to access a that specific project must wait for the lock to become available, which only happens when the previous thread is done with its command and unlocks it. This logic also make sure that clients can interact with different projects without having to wait for another project’s lock to become available.

COMPRESSION USING .TAR

When creating Older Versions of the project for later use in functions such as *Rollback,* we create a copy of the current project when the *Push* function is called and store it inside the appropriate slot inside the Older Versions directory as a **.tar** file. Here is an example of the directory tree for the Older Versions Directory:

Older Versions Tree:

olderVersions (Directory)

|—-<Project Name> (Directory)

|—<Project Name> _<Version number>.tar (Compressed Tar File)

Inside functions that need to access older versions of the project, such as *Rollback ,* we decompress the appropriate **.tar** file when an older version of a project needs to be restored.

NETWORK COMMUNICATION PROTOCOL

We defined our network protocol in terms of short send and receive messages in between any important data transfer. Some examples of messages include: “Ok!,” Received (information)” and “NO”. Note that these messages are **NOT** displayed to STDOUT and are just used internally.

GENERAL IMPLEMENTATION NOTES

2 .C source files, WTF and WTFServer

File Versions begin at : 1

Manifest Versions begin at: 0

Hash used: SHA

Older Versions: Older versions directory is created on the server side.

After a successful *push*:

- Client Manifest Version DOES NOT increment
- To make the manifest versions the same on both server and client side, there needs to be a call for *Update* and *Upgrade.*
- If the only changes between the client manifest and the server manifest are: Version Number:
    - *Update* will write out a blank update file and the **ONLY** change that will occur during *upgrade* will be changing the client manifest version to match that of the server.

SERVER SIGNAL HANDLING

The server can be quit with a SIGINT (Ctrl+C) in the foreground of its process. The server catches this signal with a signal handler and calls the ***Exit_Handler***. The ***Exit_Handler*** joins all threads from the linked list of threads, thereby removing all the threads. Afterwards the linked list data structure is freed.

MESSAGES TO STDOUT

Server: Creates socket: Waiting for connection

Client: Connects to server

Server: Lock and Unlock status of the operation

Client: Operational messages:

- Success Cases
- Failure Cases

Error Cases

- Client Errors
- Server Errors

Server Shutdown :

- Displays a “Removed Thread” message for each thread that has been connected to the server.
- Displays message to the server that it has been successfully shutdown