Client-Server-Interactor

- **Purpose of the Project:**

- To create small scale version control system with various functionalities such as configure, checkout, update, upgrade, commit, push, create, destroy, add, remove, current version, history and rollback using sockets, multithreading, forking and file api.

- **Algorithm for assignment:**

- For this assignment we are using sockets to make connection between client and server (TCP/IP) and multi-threading to handle cases when more clients join, or the intensity of commands increases. For multi-threading part we made an array of size 60 which will hold all running threads and when it reaches to limit (60) we run another for loop which joins all the running threads and resets the array index to 0. As each thread is running under mutex lock, no other thread will be able to change/edit files or any data. When the thread starts executing the first thing we do is check if everything is in all right condition on client side like the file that are being pushed, added are present on client side and then proceed to check various other things as per different command on server side and finally execute the command. The "configure" command saves ip address and port number so the user doesn't have to type ip address and port number for each command. One the .configure is made we can run other commands such checkout, update, upgrade, commit, push, create, add, remove etc that uses reads and writes between client and server. We have used linked list to read and store all buffer (SocketNode struct) that is passed between either client or server and struct SocketBuffer to store each file as multiple files can be transferred between server and client. For some of the above functions for example remove we make changes only in .manifest to handle such cases we have manifest.h which contains functions that are used to deal with .manifest. And at last for compression of files we used zlib library (supporting functions are written in compressor.h) to first compress the files and then send them between server and client. On server side when new files are pushed the version number changes and we compressed older version as per the assignment decryption.

- **Analysis of time complexity:**

-We have tried to keep our program as efficient as possible and to make the transfer and reading of files easy and less time consuming. We have used linked-list to read/store file path, file data and everything therefore the running time for this project will be approximately O(n).

- **List of all files and their working for better understanding:**

1. server_repo:

-This is directory/folder will contain all the files and that are being pushed to server and will reflect all changes will happen on server side.

2. test.c:

-This c file contains auto runner for different all basic functions using fork and execv, press control + C at the end to exit.

3. Makefile:

-This file complies everything and makes executables WTFserver, WTF and WTFtest, util.o, socket_client.o socket_server.o. WTFserver is for server side, WTF is for client side and WTFtest is for automated testing of various commands.

4. socket_client.c:

-This c file contains the client side code which will takes all commands and first checks if all parameters are present as per the assignment description and then execute various commands.

5. socket_server.c:

-This file contains server side code which performs different actions depending on how the client operates.

6. util.c:

-This file contains code that is being used by both server and client side, computing hash (MD5) for files, functions for checking whether file or directories exits, computing their size, dealing with manifest and various other supportive functions.

7. util.h:

-Header file containing all declarations from util.c

8. socketBuffer.h:

-Contains two important structs: SocketBuffer and SocketNode, which are used to read all buffer and various other supporting functions.

9. manifest.h:

-Contains two important struct ManifestNode which stores hash (MD5) , version, path and struct Manifest which stores project name and ManifestNode as linked list. Manifest.h file also contains various supportive functions for .manifest such as add, remove, free etc.

10. compressor.h:

-This file contains two functions compress and decompress to compress and decompress files using zlib library. We also compress all the old versions of the project using ".zlib" extension and use them when needed. We also try to transfer files using compression so that it becomes less time consuming and be more efficient.

11. testcases.txt:

-A file called testcases.txt contains a thorough set of test cases for our code, including inputs and expected outputs on both client and server side.

12. testplan.txt:

-This file explains how we choose testcases that cover every aspect of the assignment including all the necessary commands as show in testcases.txt

- **Using Makefile :**

  - Step 1 - Running "make" will create two executables named "WTF" and "WTFserver".

  -Step 2 - Running "make test" will create one executable named "WTFtest" to run all the testcases as specified in testcases.txt.

  -Step 3 - Running "make clean" will clear all files and folders created above and bring it to default state.

- **List of all the files of project :**