Project 2

Goal

The goal of Project 2 is for students to build *UFmyMusic*, a file/music synchronization application. Students will build both client and server capable of range of different functions.

Description

Due: Thursday, October 17th, 1:55pm ET
Points: 100
Teams: You *must* work as a pair. Students should note that both members receive the same grade.

Like so many other people, your professor is frustrated with the fact that every machine he owns has a different collection of songs in his music library. Students will solve this problem and develop UFmyMusic, a networked application that enables multiple machines to ensure that they have the same files in their music directory.

Clients can send one of **4** messages to the server: 1) List Files (ask the server to return a list of files it currently has); 2) Diff (based on 1)), the client should show a "diff" of the files it has in comparison to the server); 3) Pull (request all files identified in 2)) from the server and store them locally) and 4) Leave (the client should end its session with the server and take care of any open connections). The server must be multithreaded to handle multiple concurrent client requests (reads and writes) and store/retrieve historical information about each client in a file.

In particular, the following functions will need to be implemented:

- `LIST`
- `DIFF`
- `PULL`
- `LEAVE`

Unlike the last project, no skeleton files will be provided. However, I will instead provide design goals which must be met along the path towards implementing this project. They are:

1. Students must develop an overview of the architecture they plan to build that will achieve all of the above tasks. This should be a written document, saved as a PDF. (5%)
2. Given the above message types, students must define one or more structs capable of carrying this information. (10%)
3. Students must develop a centralized server capable of dealing with multiple clients concurrently. You may use either pthreads or select(), but you must justify your decision to do so. The server should indicate the current operation it is doing in the terminal window. Also, "diff" must actually crawl the directory structure and return all files in the current directory (you do not need to open sub-directories). (35%)

4. Students must develop a client implementing a simple interface to allow a user to enter their choices. (35%)
5. Students must develop a means of robustly determining whether or not two files, even if named differently, contain the same content (without actually sending the entire file). (15%)

Submission Instructions

Students will be using GradeScope to submit and get feedback and scores for all projects. More information to get familarized with GradeScope can be found here.

Students can directly submit their individual files using the drag-and-drop feature on GradeScope. These files include your client.c and server.c.