Matt Wallace
CSCE 513

Introduction

The object on this project is to develop a peer-to-peer file storage and sharing system. The system consists of a centralized server and at least two hosts that store data files.

The server maintains the info about files on each host through a directory-like table. Each host is uniquely identified either by its name. Each file has a unique name.

Each host client can decide to do a few tasks, usually communicating through the server. Each host can add itself to the server, remove itself from the server, add files to be accessed by the server, and remove those files.

The server receives requests from the clients and responds accordingly. When a file request comes through, the server facilitates the sending of the file, but doesn't actually send a file itself, it simply tells the client who holds the file to send a file to the client that requested it.

For the distributed version, there is no need for a server. Each client holds its version of the file/host data structure, which gets added/removed to any time a client adds itself to the "cloud," removes itself from the "cloud," or adds/removes files. This data should, in theory, stay similar among all the clients.

Challenges

Most of my challenges came from thread management, honestly. I have already worked with C# sockets before, so this was fairly trivial to implement. I had to figure out how to manage a thread giving requests to a server, and other threads managing the file requests between clients.

Another challenge was sending messages back and forth between client and server. Once I figured out I could just keep a socket open to request multiple sets of bytes, and just build a string that way, this also became trivial.

For the distributed version, the biggest challenge was obviously removing the central server. However, once I figured out how to maintain the file/host data structure across all clients, this became quite easy to do.

Implementation Details

As I said before, the majority of the implementation comes from the client and server sending messages to each other. When a client wants to make a request to the server, it opens a socket to the server, and sends a request message, containing details about the message, and any

needed details about the client itself. The server will then process the message, and respond with any relevant information (it worked, it didn't work, why, etc.).

In the distributed version, these messages are used, and also extended to maintain the file/host structure between clients.

Experiment Results

To test, I used a simple configuration of three computers: two clients, one server.

Server starts, sets its relevant information (ports, unique name), and starts listening for requests.
The two clients start, set their relevant information, and starts listening for requests.
Both clients now have the ability to add and remove files. To test, add a file from each host, and request the files from the other host. In other words, add file1 from host1, and add file2 from host2. Then, request file2 from host1, and file1 from host2.

As a result, host1 will now have file1 and file2, and host2 will have file1 and file2.

For the distributed version, I do all these things, except don't start a server. Instead, start one of the clients as a "joiner" client, which will be the first addition to the cloud.

Conclusion

The intention of this project was to teach us about socket-based network programming. I believe the project was successful in this endeavor. I learned quite a lot more than I previously did about socket programming, and network-based projects in general, especially with the distributed version.