Jesi Merrick A20264903
Ron Pyka A20250364
Dylan Boliske A20252450

Design Document

Overall Program Design

Initially, we each branched off to research and start a basic program using sockets and asynchronous communication in C individually. Once we each finished this, we discussed our research, and chose one of our programs to use as a base for the assignment (each of us came back with a working client and async server.) We did this so that each member could have some time to work with the base code and understand what was happening, as well as ask questions, before we really dug into the details of the project.

There are three key parts between the server file and the client file. The client file has two main threads: the original program, which acts as the client, and the server thread, which acts as a server for the client's peers. The indexing server and the client servers are implemented in the same manner. They keep a pool of sockets, which are handled using select(), and allow multiple clients to be connected and interacting with the server.

From the client side, there are six commands that are accepted: addf, remf, getf, retf, regf, and exit. The command addf takes a filename to be added to the index of the indexing server. This index is represented by a linked list of index items, each item containing a file name and an IP address. The command remf is similar, but that it will remove the specified file name from the index. The third command, getf, takes a filename and then displays a list of peers that have that file. With this, the client can use the retf command, or "retrieve file". The command take a host name and file name. Using the host name, the client can open a new socket and connect to their peer, sending a request with the file name for the data. A more complete version of addf, the regf command will register all the files in the directory, except for the application files, with the indexing server. Finally, there is the exit command as well, to disconnect from the indexing server and close all the sockets (including the client's server-side).

Design Tradeoffs

The initial tradeoff considered was with whether to use Python or C. While we knew we could do it rather quickly in Python, as one of our members had already built almost exactly the same program for CS451, we decided to invest the time in using C, so that it may be easier to build on and expand the program later on, as it was mentioned in class that it might be a base for later projects.

We also considered whether or not we should use global IP addresses or simply local network ones for our client ids. Due to time constraints and the complexity of dealing with global addresses and port-forwarding, we decided to stick with the local IPs, but likely would have expanded this in the future.

Another tradeoff that was considered was how to deal with files that have been edited. We had initially wanted some way for the indexing server to detect updates and push them to the other clients, but when this began to seem to complex, we decided to change our approach. We decided that when a file is changed or removed, whether intentionally or unintentionally, the client server will send a request to the indexing server to remove it from the index.

Improvements and Extensions

While our current program is only capable of file transfers between machines on a local network, it could be expanded to work more widely by using global IP addresses rather than only the address on the local network. This would require that the indexing server figure out what the IP address is for each client is globally rather than just pick it up on the local network.

In addition, we still have a couple features not quite working. The remf command can occasionally cause a segmentation fault on the server, that we ran out of time to resolve. Also, polling the local files for changes was interfering with adding files to the indexing server, and chewing up an entire core since it was still a very simple approach. We have commented out the lines which create that thread as it has not been resolved yet.