User Manual

The guide assumes that you have the ability to compile Rust code on your system.

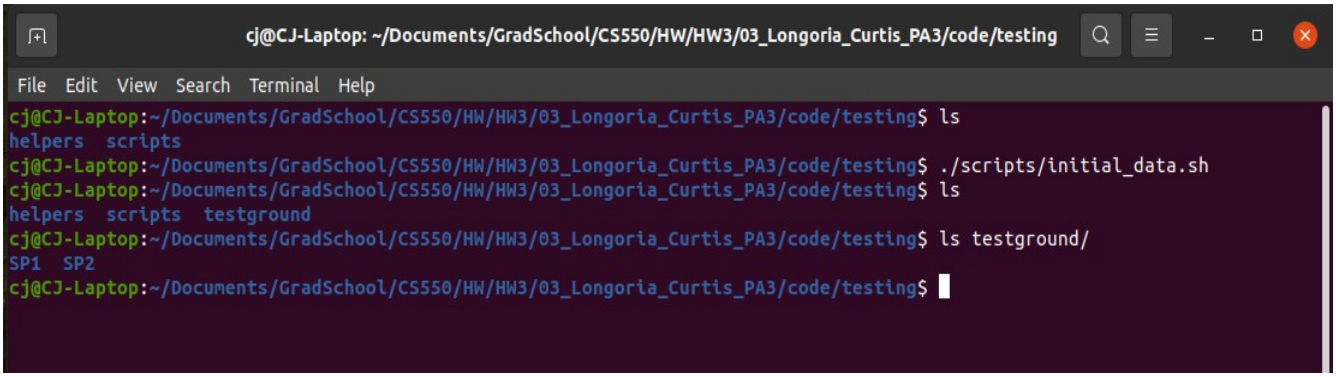
The file structure is depicted below:

```
docs
  |------Performance.pdf
  |------Design.pdf
  |------Tests.pdf
  |------Manual.pdf
  |------Output.pdf
code
  |------build.rs
  |------Cargo.toml
  |------Cargo.lock
  |------proto
          |------p2p-server.proto
  |------src
          |------client.rs
          |------server.rs
  |------testing
          |------helpers
                  |------all-config.ini
                  |------linear-config.ini
        |------scripts
                  |------file_mod.sh
                  |------initial_data.sh
                  |------make_data.sh
                  |------perf.sh
  |------target
```

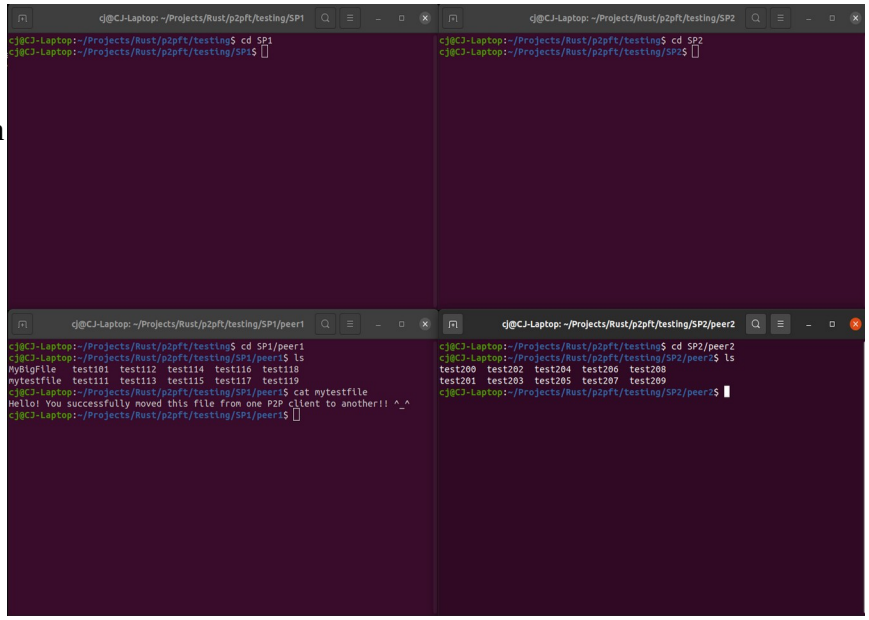
Quick Start

Step 0

Begin by navigating to the testing folder (Code>testing) and running the initial_data script using the command "./script/initial_data.sh". After running the command verify that you have a "testground" folder inside the "testing" folder. Additionally, verify that the "testground" folder has the three directories "SP1" and "SP2" inside it.



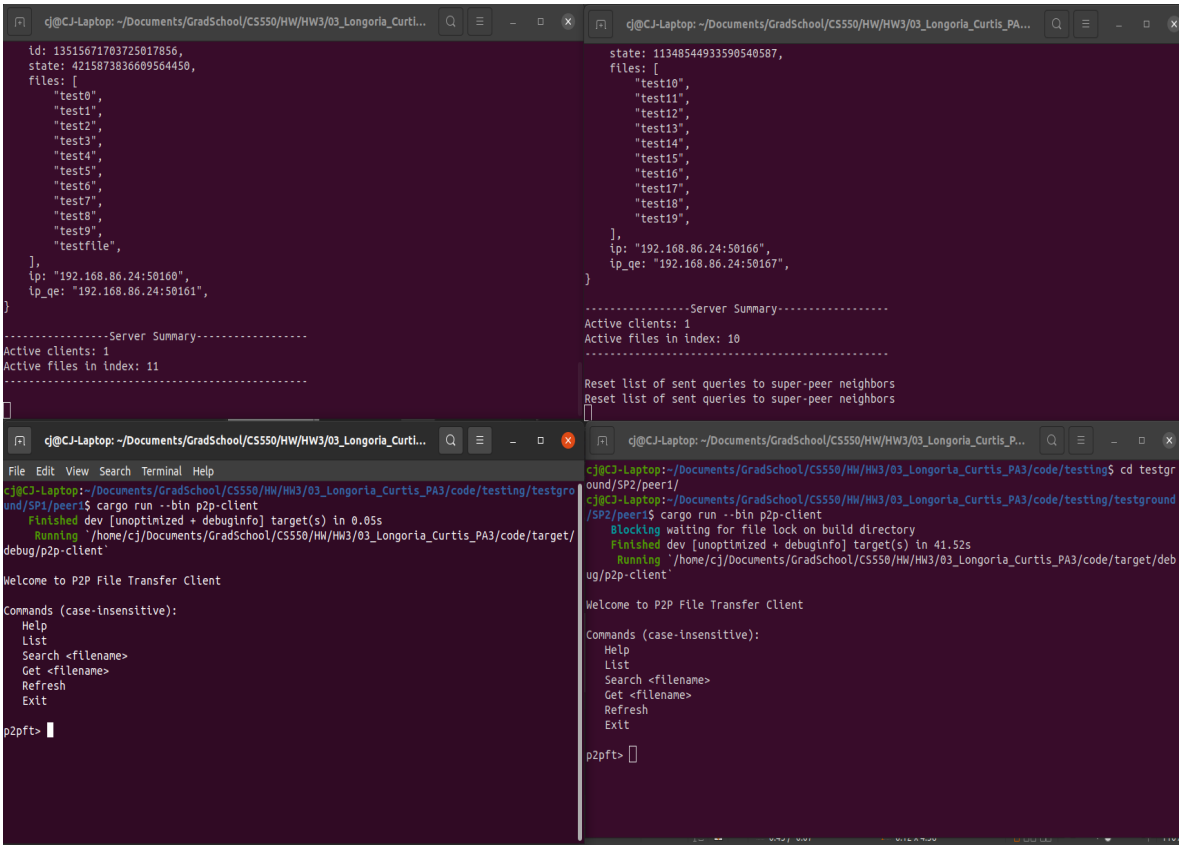
Step 1



Now open four terminals and navigating to the testing folder (Code>testing>testground) on each. Then navigate to the "SP1" (top-left) and "SP2" (top-right) directories in the top two terminals terminals. Navigate to SP1>peer1 (bottom-left) and SP2>peer2 (bottom-right) in the bottom two terminals. Notice that there is a test file "testfile" in the SP1>peer1>Owned directory. Go ahead and type "cat Owned/testfile" in the bottom-left terminal to see what it says. During this quick start we will move that test file from the SP1>peer1>Owned directory to the SP2>peer2>Downloaded directory using the P2P program. Refer to the picture on the right as the starting point for this quick start guide.

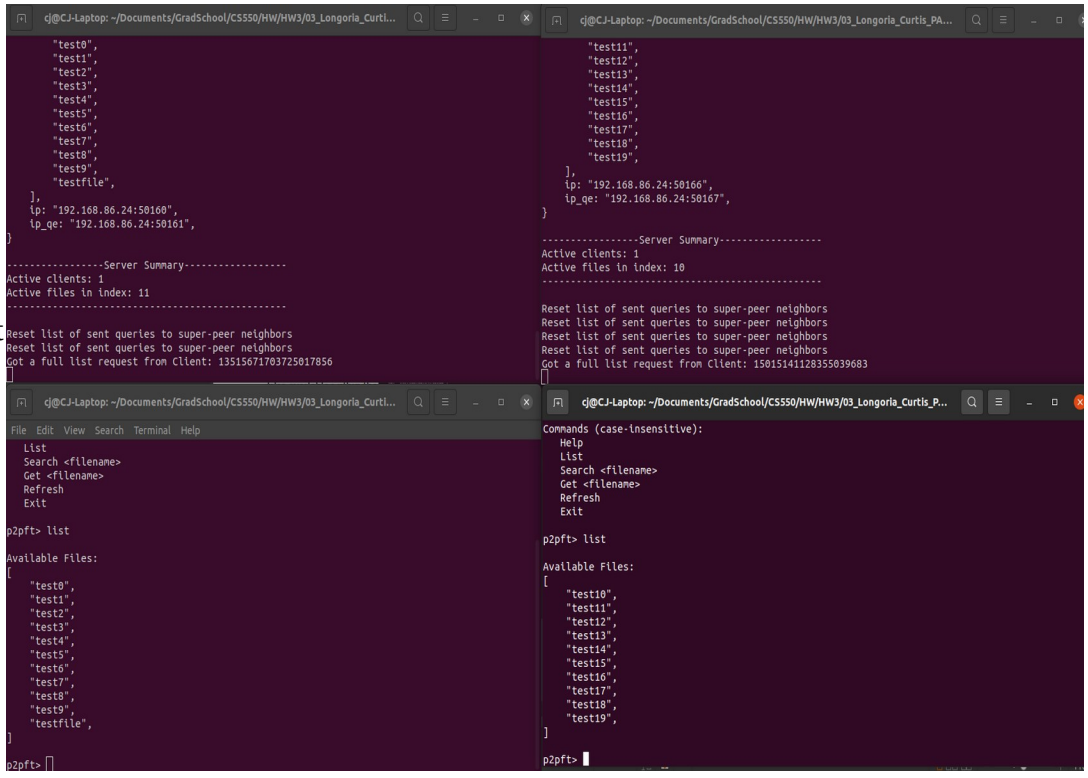
Step 2

First run "cargo run --bin p2p-server" in the top terminals and "cargo run --bin p2p-client" in the bottom terminals. Notice that the server logs that new clients have registered to the P2P system and lists each client's info, including what files they have available for download. Refer to the picture below for how this should look. *Note: if you get any errors about a thread panic exit (CTRL+C or type "exit") and re-launch the binary.



Step 3

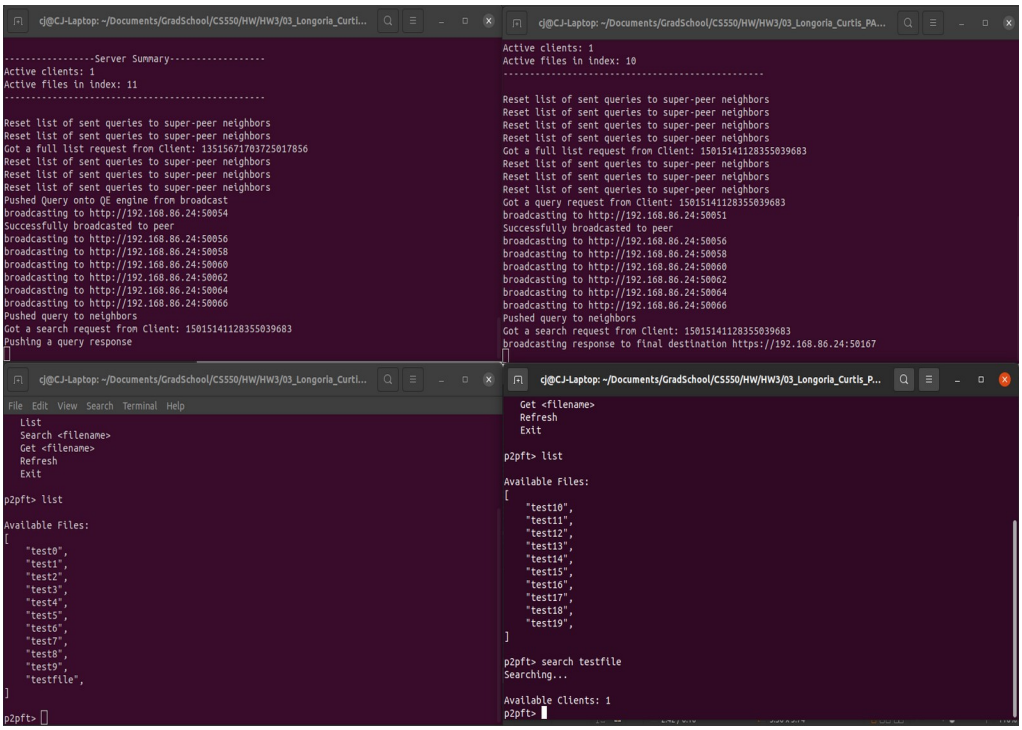
Now enter the command “list” then press enter in both bottom terminals. You can see that the client returns a list of all files available within the respective super-peer group. Notice that each client sees a different list of files. This is because one client is on super-peer 1 and the other is on super-peer 2. We will be able to move files via search function and super-peer broadcasting. The top terminals that are running the index servers (super-peers) might have a log entry



that the list of sent queries to super-peer neighbors has been reset. This is normal as the list is periodically cleared.

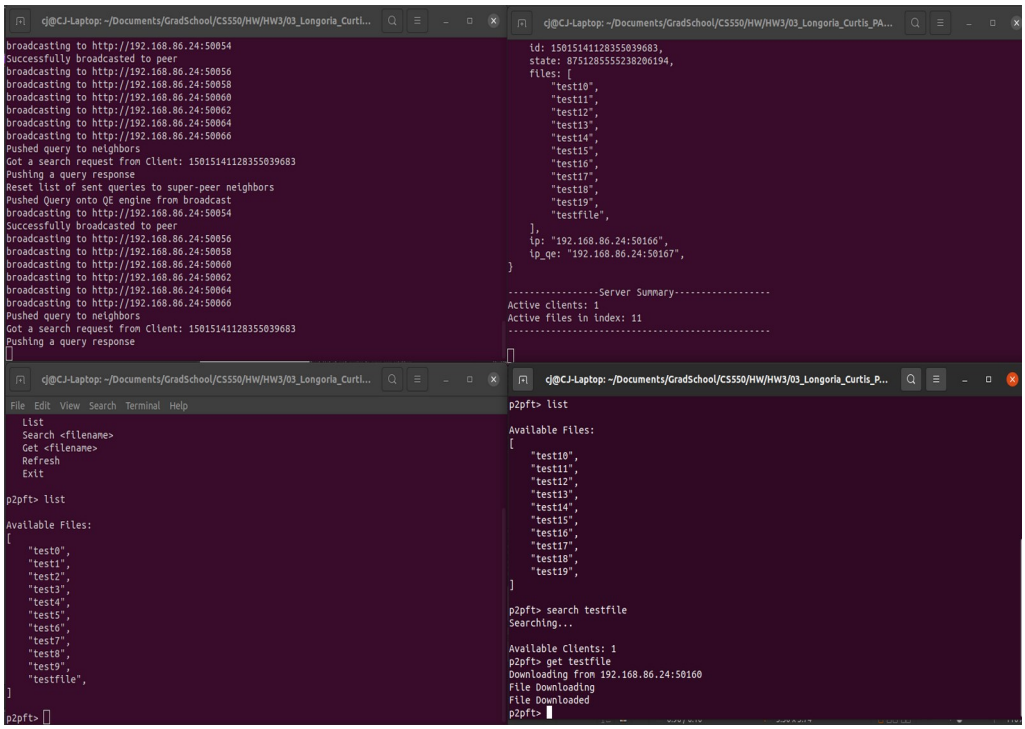
Step 4

Now enter “search testfile” and press enter in the bottom-right terminal. The program returns that the file is available from one other client. The server logs that a client made a search request. The top terminals will display a lot of messages on this step. You can see that since we didn’t provide any CLI arguments the servers attempt to broadcast to all super-peers assigned in our static configuration file. However, the only successfully ones are from the servers running in the terminals. The top-left terminal shows a search request entry followed by a “pushing a query response” this means the super-peer had a query hit and is back-propagating the results. The top-right terminal shows an entry that it is broadcasting a response to the final destination. This is just pushing the query results from another super-peer along.



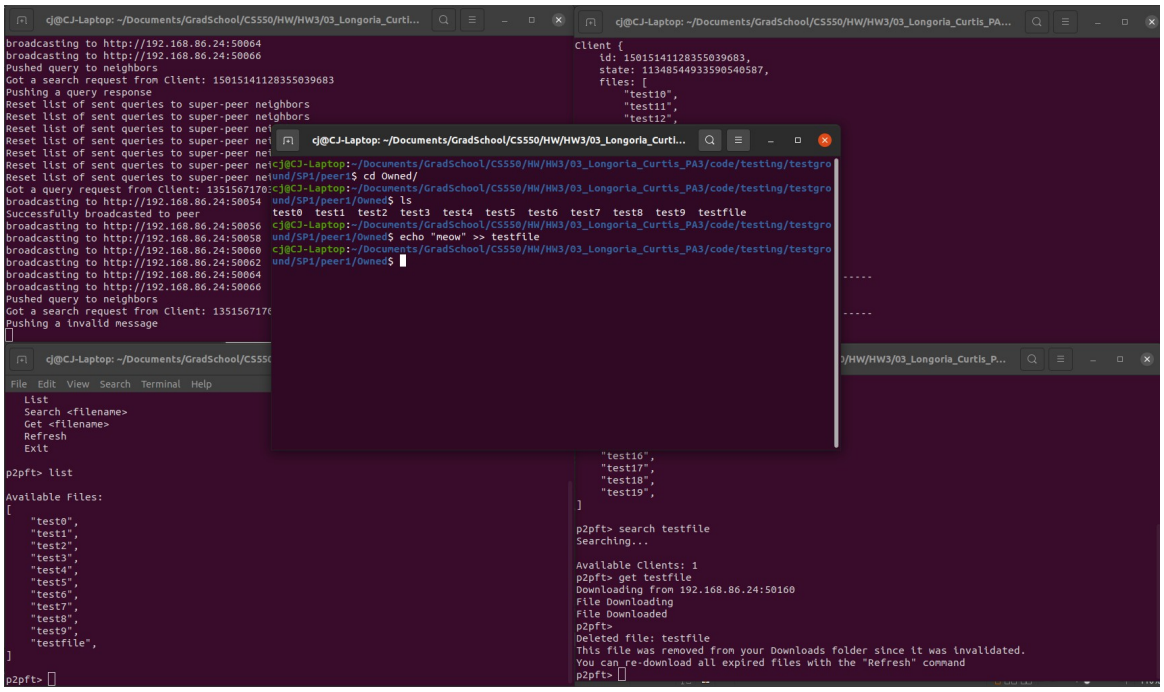
Step 5

Now enter “get testfile” and press enter in the bottom-right terminal. The program returns that the file is available and references the client’s IP address that has the file. The program then transparently reaches out to the peer and downloads the file. The server logs that a client made a search request, but has no visibility on if the client downloaded a file from another peer. After the file is downloaded the auto-update mechanism notices a change in the client’s state and deregisters/re-registers the client with the indexing server.



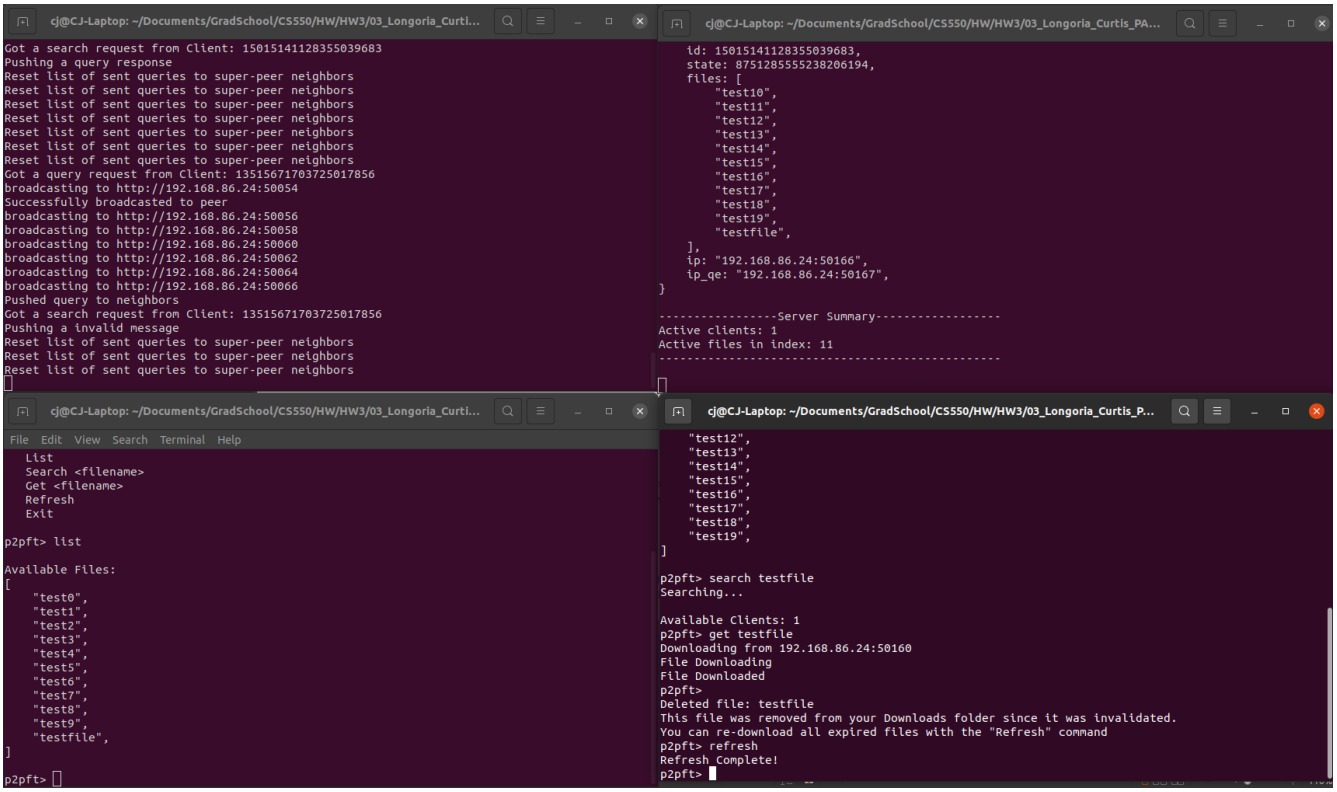
Step 6

Now open a new terminal and navigate to SP1's peer1's Owned folder (Code>testing>testground>SP1>peer1>Owned) and type the command "echo "meow" >> testfile". This will update the state of the file and trigger an invalidation message to be sent out.



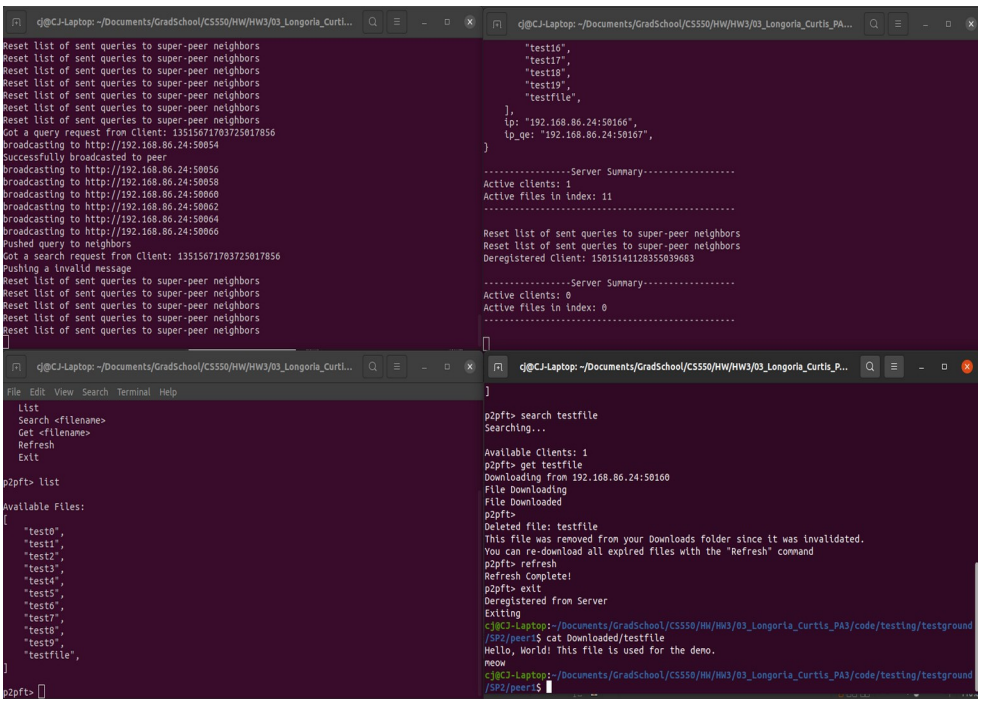
Step 7

Close out of the new terminal you just opened. Notice that an invalid message was logged by the server in the top-right terminal and the bottom-right terminal notifies the user the the file has been deleted since it was marked as invalid. In order to re-download the newest version of the file type “refresh” in the bottom-right terminal and press enter.



Step 8

Now that the file is downloaded type “exit” and press enter in the bottom-right terminal. Read the downloaded file by typing “cat Downloaded/testfile” to verify you received the same file we started with on SP1 please the additional text “meow” added in to version 2.



User Notes:

- The super-peers default to a broadcast configuration (where every super-peer is connected to every other super-peer). This can be changed to a linear configuration by running “cargo run --bin p2p-server linear” instead of “cargo run --bin p2p-server”

- The client program run with a push based consistency model by default. This will broadcast out an invalidation message anytime the original file is modified. This can be switched to a pull based method with a configurable TTR value located in the config ini files in the helpers directory. In order to use the pull based method run the client binary with “cargo run --bin p2p-client pull”

- The super-peers and their associated clients (leaf-nodes) are pre-configured in a static configuration file under the “helpers” directory. This means you have to run the p2p-server binary from one of the SP directories and you have to run the p2p-client binary in one of the peer directories within one of the SP directories.

- Since the ports are static and pre-configured the thread will panic if the port is already in use. In this event you will have to re-run the program.

- TTL (time-to-live) is set to 3. So if you run the servers in a linear configuration you wouldn’t have complete coverage of all super-peers.

- The servers can eat a lot of CPU if you run a lot of them at the same time. Each server consumes about 6% (while idle) of CPU on my machine