CS 550 - Spring 2022
Isaias Rivera
A20442116

P2P File Sharing - Consistency - Manual

Note

This manual only covers how to use the program. Terms that will be not explained in this manual are `ip address`, `ports`, `port forwarding`, `directories`, and `file hashes`. If this program is only needed to run locally, the *Simple* examples should work just fine. However, this manual is also mainly for Windows, meaning, the exact syntax of some commands may depend on your system.

Procedure

The typical procedure for using this program is as follows

1. Create valid folders for each client
2. Move files to folders for each client's origin folder that want to be shared
3. Create the static connection config (or use the one provided)
4. Start at least two clients, using the folders previously created, and where at least one of the clients is a super-peer
5. On one client, search for file
6. Request file, given the file hash (The long string of characters) from searching
7. Wait for file download or repeat steps 5-6
8. Close all clients

Program arguments

A <int> just means it should be a number.

Client

USAGE:

Client [-ahls] [–version] [-c <filePath>] [-d <directory>] [-i <int>] [-r <int>] [-u <directory>]

Where:

-i <int>, –identity <int> Unique ID identifying this client

-c <filePath>, –configFile <filePath> The config file to use

-u <directory>, –uploadFolder <directory> The local folder files should be uploaded from

-d <directory>, –downloadFolder <directory> The local folder files should be downloaded to

-r <int>, –ttr <int> Time To Refresh (TTR) in seconds, if in pulling mode

-a, –all2all enable all2all mode

-s, –pushing enable pushing of invalidation calls

-l, –pulling enable pulling of invalidation calls

Example

***Simple* - Local Only**

```
.\Client.exe -i 0 -s -c "test_config.json" -u "watchFolder" -d "downloadfolder"
```

Using the Interactive Console

The main interface with this program involves using the terminal of your device when using the client.

The following options are allowed when using the client.

- `ping`
  - Ping a client's superpeer's indexing server to check it's response time
- `refresh`
  - If in pulling mode, this command forces a refresh on all invalid files
- `list`
  - List all the files available on the network, meaning, show all the files we can either download or already have
- `search [query]`
  - Search for the query as a substring the in the name of all the files on the network, meaning, search for a file where `query` is what you want to look for
  - Example: `search am`
    - Search for files with an `am` in it's name
- `request [hash]`
  - Request a specific file, given the hash, meaning, copy the `hash` (The long string of characters) from a single file when you did either a `search` or `list` and paste it where `hash` is.
  - Example: `request E2D0FE1585A63EC6009C8016FF8DDA8B17719A637405A4E23C0FF81339148249`
- `q` or `quit` or `exit`
  - Stop the interactive console, this will close the client

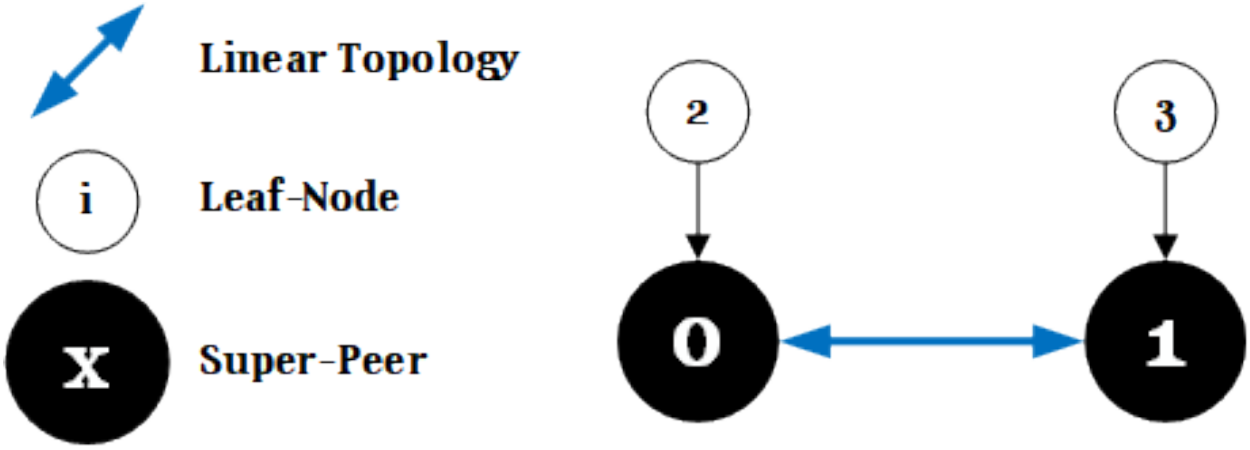
Remember, to add or remove files for sharing. Simply delete or add them to your origin folder that you initially created for the client.

Overall Example

This example will demonstrate following the procedures for a *simple* local case.

Topology

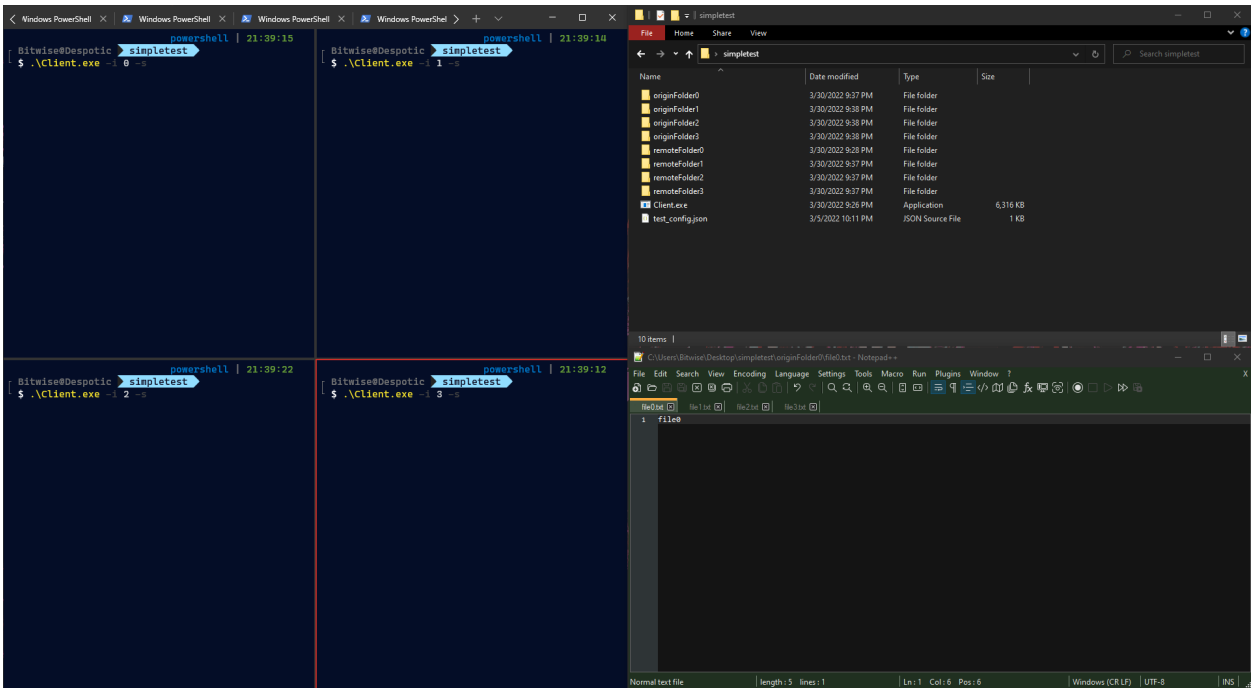
Because this is a simple example, it will only use two super-peers and two leaf-nodes, layed out as such



The config file for this is included.

Initial setup

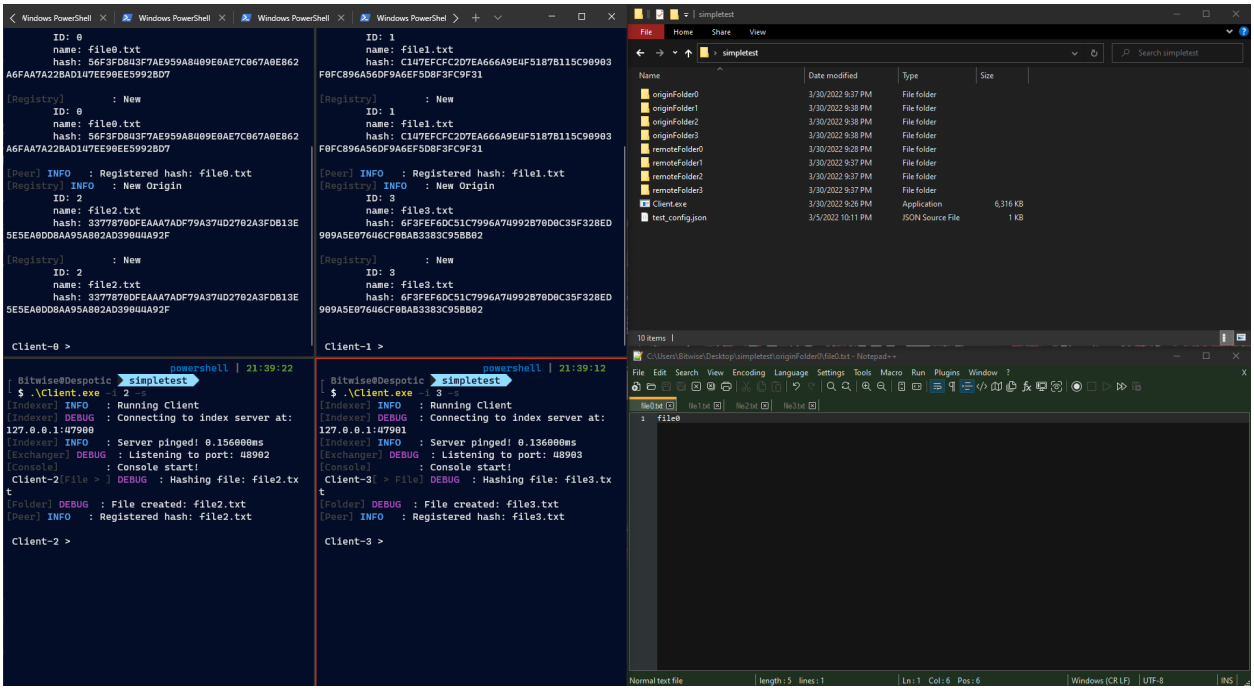
Here we can see the terminal is split into four for each peer. The folders relative to the app. And the files that are in each origin folder (Only file0 is shown open but all the other files also have unique data).



`.\Client.exe -i 0 -s` start super-peer with id 0 in push mode
`.\Client.exe -i 1 -s` start super-peer with id 1 in push mode
`.\Client.exe -i 2 -s` start leaf-node with id 2 in push mode
`.\Client.exe -i 3 -s` start leaf-node with id 3 in push mode

NOTE: if no argument is given for the watchfolder it defaults to `XID` where `X` is either `originFolder` and `remoteFolder` and where `ID` is the id of the client.

Running Program

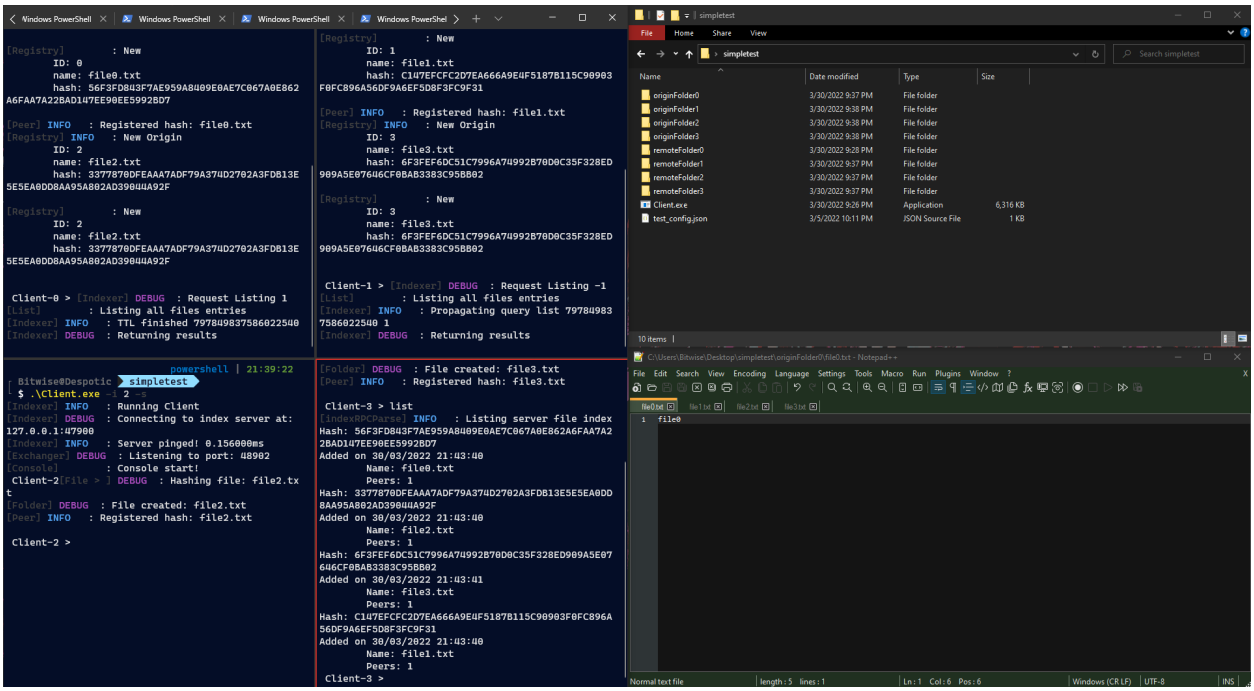


After running them all, you might notice the prompt `Client-x >` becomes a bit garbled, simply pressing `enter` once should fix it. Also, take note that the `x` is the id of the client.

Listing all files

Here we are listing all the available files from client 3

We run the command `list`



Take notice of the `Hash` associated with each file listed, this is what we use to request a file to download

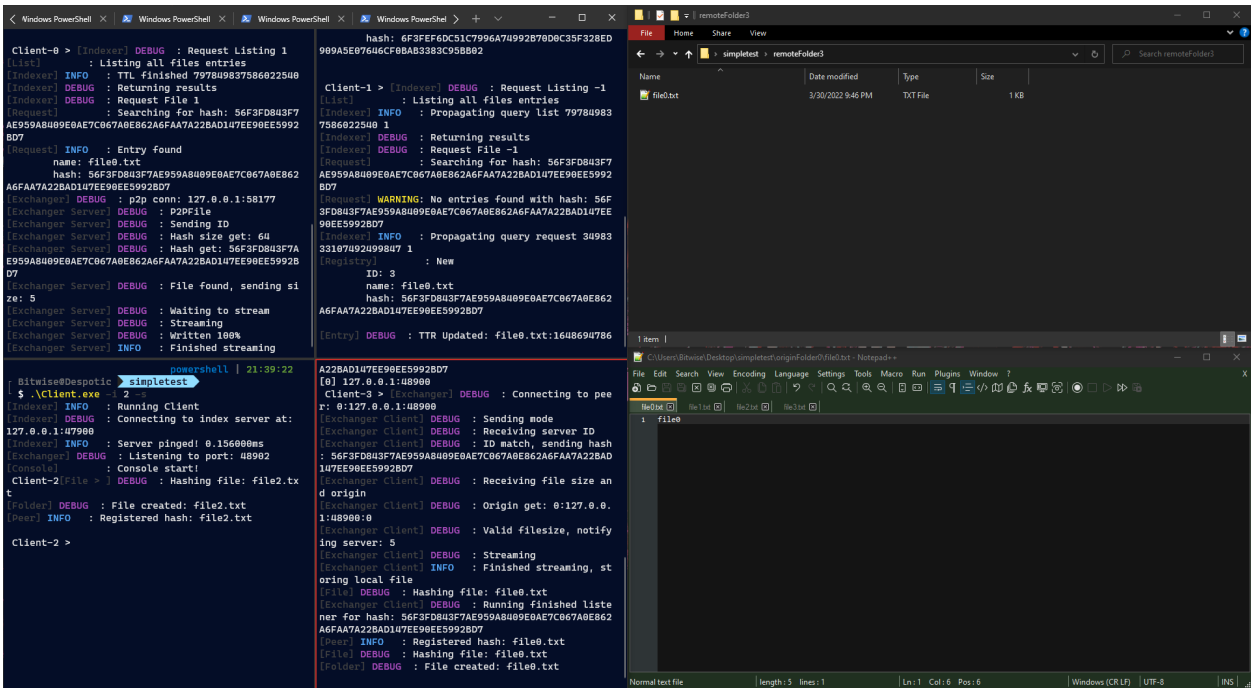
Lets focus on the file `file0.txt` which has the hash of `56F3FD843F7AE959A8409E0AE7C067A0E862A6FAA7A22BAD147`

Requesting a file

Now we need to actually request this file by running the following command from client 3

We run the command `request 56F3FD843F7AE959A8409E0AE7C067A0E862A6FAA7A22BAD147EE90EE5992BD7`

After this runs, we can now see that the file is in the remote folder for client 3.



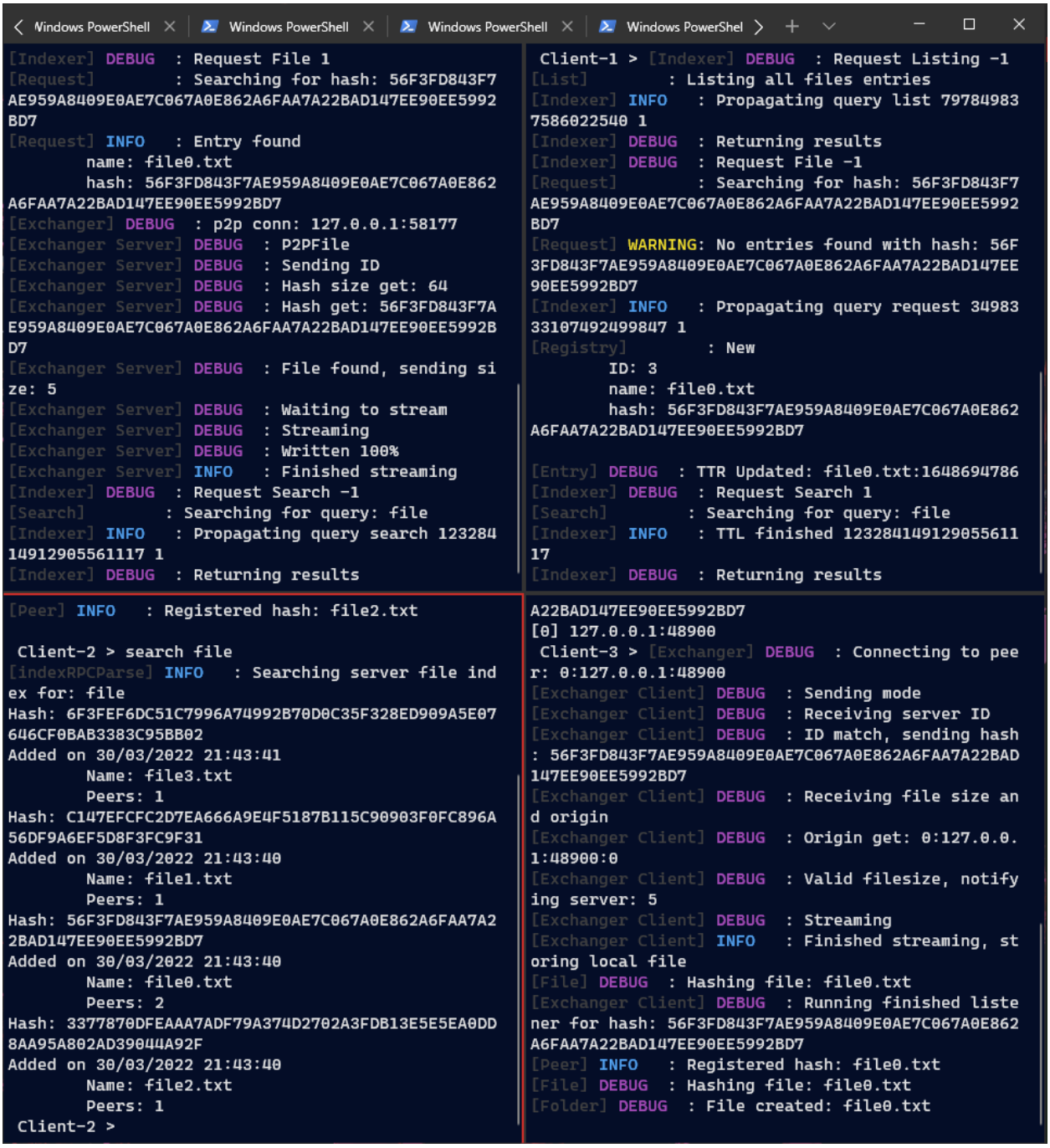
We can also see that the origin for this file was received by client 3 with the message

`[Exchanger Client] DEBUG : Origin get: 0:127.0.0.1:48900:0`

Search for a file

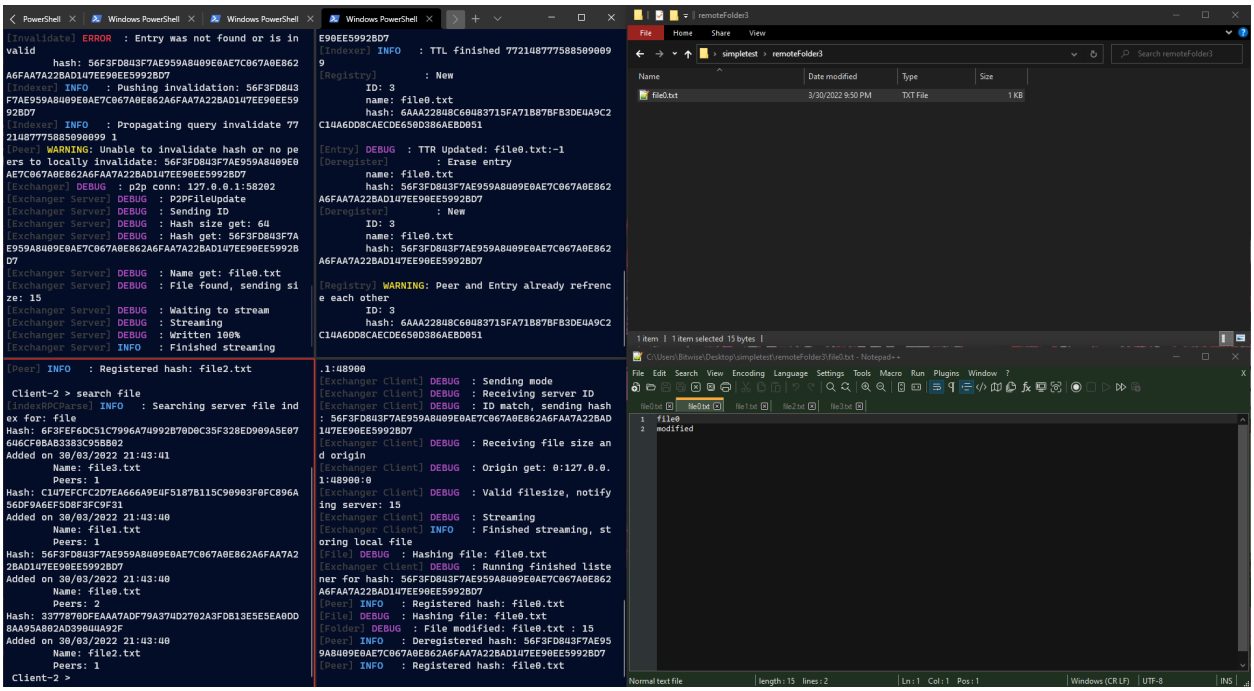
Here we can now see, after searching for the term `file` from client 2, that there are now two peers which have the same file.

We run the command `search file`



Modify file

Here we can see super-peer 0 propagated an invalidation after modifying it's file which caused peer 3 to re-request the file from peer 0.

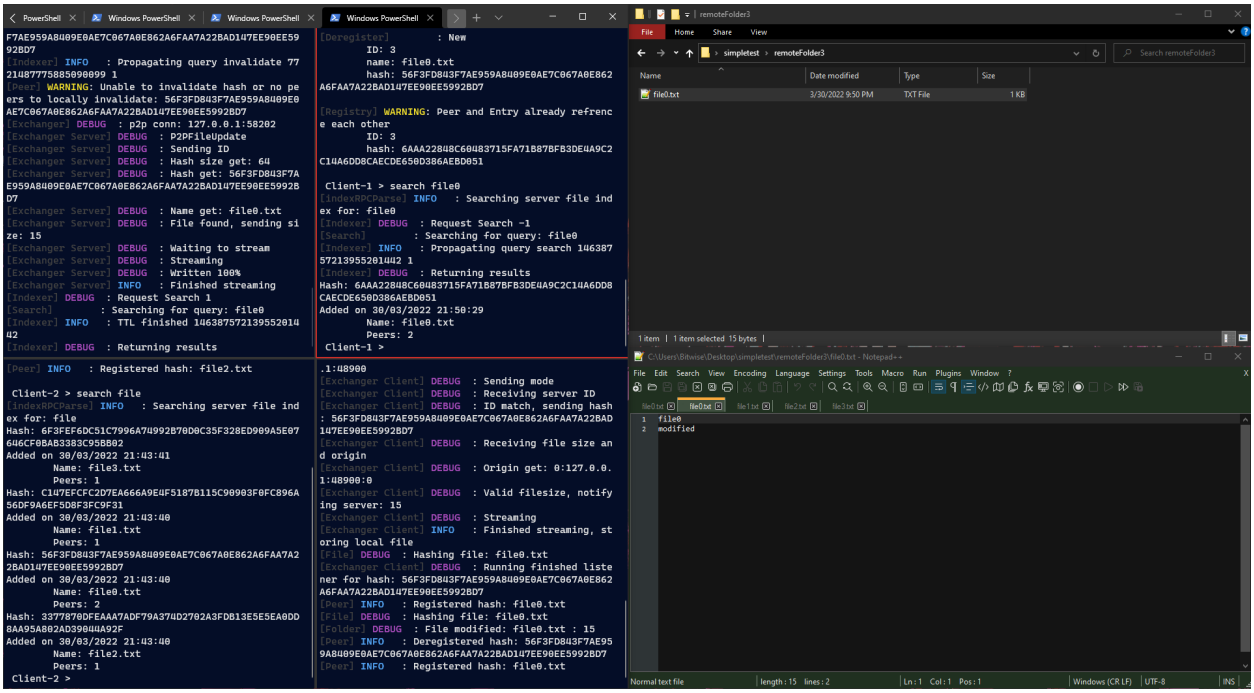


note that the file open is not the original file, it is the file located in peer 3's remote folder

Search for modified file

Here we can now see, after searching for the term `file0` from client 1, that there are still two peers which have the same file.

We run the command `search file0`



Closing

We now can close all clients with `q` and we are done.