How to Run

Running server: *python3 server.py*

Running each peer: *python3 peer.py*

Program Structure

The program is divided up into two scripts, one to handle the functions, input, and output of the server and one for the peer.

The file-structure is broken up as follows:

- The root folder contains the peer script (peer.py), the server script (server.py), the program instructions (README.pdf), and a subdirectory titled “files”.
- Contained within the “files” folder are the files which the system can detect and register, and another subdirectory titled “peers”. Two files are prepackaged: one is titled “doug.png” and another is titled “sample.txt”.
- Contained within the “peers” folder is a collection of folders, one for each peer and named after each peer.
- Within the respective peer folder, there are “seeding” and “downloading” folders. The chunks are located here; the “seeding” folder contains chunks intended to be seeded by others, the “downloading” folder contains chunks which are intended to be merged into a file which will also be contained within said directory.

The server script is divided as follows:

- The program begins with some simple logic to set up the server itself and get it listening for peers. Then, the main thread executes a loop where it is always ready to accept new connections: for each connection, a new thread is created to be able to handle its logic.
- Each of these threads are constantly listening for instructions sent by their associated client. When a signal is received, it’ll enter one of many different operation modes:
    - OP_NULL is the default signal. Whenever this is running, the server continues to wait for a signal indicating other operation logic. Any other operation must set the operation mode back to OP_NULL to ensure that the server can continue to receive input.
        - In addition at the end of each operation, the working directory should be reset back to the ‘files’ directory.
    - OP_Q is the logic for handling the cleanup of a disconnected peer. The peer is removed from existing dicts, its files are deleted, and the server indictates who disconnected and how many peers remain.
    - OP_R is the logic for requesting the file. Server-side is responsible for taking the filename and filesize provided by the client, then identifying, splitting, hashing, and registering rhe file into the relevant dictionaries.
    - OP_LST lists files currently registered on the network for the requesting client to be able to view.
    - OP_LOC provides the chunk locations for a given file and then forwards it to the requesting client for viewing.
    - OP_D is the logic for downloading the file. Server-side is responsible for taking the requested filename, identifying the file it is related to, identifying which peers have the file’s chunks, then creating a list of parameters to forward to the requesting peer. Within this list are:
        - A dictionary containing each peer and a list containing their respective owned chunks
        - The address of the requesting peer

        - The name of the file being requested
        - The size of each chunk in the file
        - A dict containing each chunk's ID and hash

The peer script is divided as follows:

- The program begins by setting up the initial conditions (changing to the "files" directory and marking down its own IP address/port). It then establishes a connection to the server, and prepares to establish a connection to other peers
- A thread is started for the seeding functionality, which waits for other peers to connect, takes in the parameters passed in by connected peers, then sends the files in the "seeding" directory to the peer requesting it
- A thread is started for the command-line input, which sends out signals to the server to coordinate its operations. In this thread, the user can input the following commands to send to the server for processing:
    - q: quits the program
    - r: registers a new file (requires filename as subsequent input)
        - client-side logic handles taking a filename input, then passing the filename along with the size to the server
    - d: downloads a file from the registered list (requires filename as subsequent input)
        - client-side logic handles taking a filename input, passing the parameters between the server and the peer, and downloading, merging, then seeding the files
    - lst: lists the currently registered files
    - loc: lists the endpoints associated with a given file

Program Functionality

Parts of the program which work:

- Disconnecting a node
- Requesting a file
- Requesting multiple files
- Downloading a file
- Downloading a file from multiple peers
- File registration/progress/download notifications/replies
- Seeding a chunk once downloaded
- Splitting and merging files
- Listing endpoints and registered files
- Hashing and decoding hashes
- Other stuff necessary to the basic functionality of the program

Parts which don't work:

- downloading invalid filename inputs
- handling unexpected quits in the peers

Parts which may/may not work:

- Multiple peers downloading the same file simultaneously

Sample Output

Downloading file:

```
P2P File Sharing Service

---------

Enter a command:
```

```
d
The following files are currently registered:
doug.png

What file would you like to download?
doug.png
Recieved chunk 1 from 127.0.0.1:42492
Recieved chunk 2 from 127.0.0.1:42490
Recieved chunk 3 from 127.0.0.1:42490
Recieved chunk 4 from 127.0.0.1:42492
Recieved chunk 5 from 127.0.0.1:42492
Recieved chunk 6 from 127.0.0.1:42490
Recieved chunk 7 from 127.0.0.1:42490
Recieved chunk 8 from 127.0.0.1:42492
Recieved chunk 9 from 127.0.0.1:42490
Recieved chunk 10 from 127.0.0.1:42490
doug.png has completed downloading.
```

Registering File:

```
P2P File Sharing Service
---------
Enter a command:
r
What file would you like to share?
doug.png
Registration of doug.png of size 107330 bytes successful.
```

Disconnection/Connection of Peers (Server):

```
Waiting for peers...
127.0.0.1:42390 has connected. 1 peers on network.
127.0.0.1:42392 has connected. 2 peers on network.
127.0.0.1:42394 has connected. 3 peers on network.
doug.png has completed downloading.
127.0.0.1:42394 has disconnected. 2 peers on network.
```

Endpoints Location Request:

```
P2P File Sharing Service
---------
Enter a command:
loc
What file would you like to check?
doug.png
There are 2 endpoints seeding this file.
Peer 127.0.0.1:54374 currently has chunks [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].
Peer 127.0.0.1:54372 currently has chunks [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].
```

Requested File List:

```
P2P File Sharing Service

---------

Enter a command:

lst

There are currently 1 files in the network.

(1) Name: doug.png    Size: 107330 bytes
```