Design Doc:

Indexing Server

To get the server running, 2 arguments must be passed down in the python script:

- The IP address of the server
- The Port of the server

Therefore, the script carries a quick checking for the number of parameters to keep running.

The main feature of the server is that includes a dictionary with all the peer nodes that are registered for P2P downloads. This dictionary is as follows:

- Each key corresponds to the address of a client (IP and port)
- Each value is a tuple which corresponds to:
    - The port of the client's server (so that it can be shared to other clients to establish the connection to download files)
    - All the files that the client hosts.

Then, the server is configured by creating its socket, binding it with the IP and Port chosen and start listening for possible connections. To be able to accept multiple connections, a forever loop is implemented to accept every new connection and manage it with a new thread that gets the basic information of the connection: its socket and address.

The thread runs a function with another loop which starts by a synchronous receive from the client. This means that until the client does not send a command to the server, this thread will be waiting.

If the data received is empty, the server handles the issue and prints out that the client has disconnected and, in case that the client was registered for P2P downloads, it also gets removed from the dictionary. In case it is not empty, it gets decoded to get a string and split by empty spaces to be able to handle all the possible input commands.

If the command is register, the client sends its own server port and the list of files it hosts, and the indexing server adds the client to the dictionary.

If the command is unregister, the indexing server removes the client from the dictionary and handles the exception in case it was not registered.

If the command is get_files_list, the indexing server gets all the available files from the other registered peer nodes and sends them to the client.

If the command is delete, the server receives the new list of files from the client and updates its dictionary value in the case it is registered.

If the command is update, the server receives the new list of files from the client and updates its dictionary value in the case it is registered.

Finally, if the command is download, a list of peer nodes which host the file requested is sent to the client. To do so, if the list of nodes is empty, a message telling the client that No registered Nodes host that file is sent.

Finally, the server apart from printing the information also records it in a file call log.log.

Peer Node

To get every peer running, 4 arguments must be passed down in the python script:

- The IP address of the indexing server
- The Port of the indexing server
- The Port of the client's server
- The folder in which the files of the **client** are kept

As with the server, the script carries a quick checking for the number of parameters to keep running.

It is important to note that each Peer Node is composed of a client and a server:

- The client is the one that connects and shares information with the Indexing server. Therefore, the client is the one that tells the indexing server its own server port for connections with other peers and also connects to other peer's servers for downloading files.
- The server is the one that listens and allows connections with other client peers and sends the files requested from other peers.

For that reason, each peer node has two main threads: one for running the client and other for running the server.

Therefore, the first thing is creating a thread that calls the function server with the Port of the arguments. This function creates a socket, binds it and starts listening for possible connections. For that reason, a loop that runs forever is implemented to accept all the connections and launch a new thread (with the function send_file) for every connection.

The function send file is the same function that was implemented in the Programming Assignment 1, which works by receiving the file name from another peer, sending the file specifications (including its hash) and then send the complete file. Finally, the function is in charge of closing the connection with the other peer.

At the same time, the client's socket is created and connected to the indexing server and another loop that runs without stop is created for the client, to be able to call one command after another.

Hence, the first thing the client does is read the input of the shell for new commands and when one is written, split it up by blank spaces.

If the command is register, the server port is added to the message before it is sent to the indexing server. Then, the function update_files_list is called to get all the files that the client hosts and send their names to the indexing server.

If the command is unregister, the client just sends the command to the indexing server so that it can handle it.

If the command is get_files_list, the client sends the command and receives a list of all the files that the registered peer nodes are hosting at that moment.

If the command is delete, the client removes one of its files and notifies the indexing server calling the update_files_list.

Finally, if the command is download, the first thing the client does is sending the file it wants to download to the indexing server. Then, it receives a list of registered peer nodes that host the file requested. The client chooses one of all peer nodes at random and launches a thread with the function download and the IP address and port of that peer node.

The first thing that the function download does is creating a socket and connecting it with the other peer node server. Then, the same download protocol as the one created in the Programming Assignment 1 is used, in which the name of the file is sent to the server, the file specifications are received and then, the file. If the hash of the file matches with the one received in the file specifications, the client tells the indexing server that a new file has been downloaded to update the files list the client has if it is registered for P2P downloads.

Finally, it is important to note that, both for the client and server two functions have been implemented: send_all and recv_all.

Every time that something wants to be sent or received, these functions are called and ensure that all the data is correctly sent and received.

The send_all function implements a loop that goes through the data and sends it in buffer (1024 bytes) size chunks until all the data has been sent. (Note that the last chunk can be less than buffer)

The recv_all function also implements a loop to make sure that it returns the data when all of it has been received. At the same time, in case that the length of the data is unknown but less than the buffer size, the loop is terminated after the first iteration.

Some of the possible improvements and extensions could be:

- Implementing a thread that is constantly checking the files in the client folder, to notify the indexing server every time one file gets removed or added automatically.
- Implementing a login system for the peers.
- Changing that instead of randomly select the peer node to download a file, choose the one that is the less congested or the one that will provide the better performance.