**Peer-Peer Server**

Programming Assignment 2

Abheek Mondal

A205438046

CS 550

Monday, 06 March 2023

In this assignment, you need to change the server-client architecture from Programming Assignment 1 to a simple P2P architecture file sharing system. This simple P2P system requires two components. Peer node: you need to combine the server and client programs previously made into one single peer node. Each peer node can send and receive files to/from other peer nodes. A central indexing server node: this node could provide the list of all of the peer nodes and the files they hold. Peer node could query the indexing server node to get the peer node lists that hold the request file and then directly go to that node to request for this file.

These are the main program files that I created:

**Peer.py**

The peer node implementation for a file-sharing network. The PeerNode class has several methods to handle different tasks such as registering with the indexing server, hosting files, downloading files, and handling incoming file download requests.

The register\_with\_indexing\_server() method registers the peer node with the indexing server by sending a "register" message containing the node's address (host and port) to the indexing server using a TCP connection.

The get\_file\_list() method retrieves the list of files hosted by the peer node by iterating through the files in the files\_dir directory and creating a dictionary of filename and file size pairs.

The handle\_client() method handles incoming file download requests from other peers. It receives a request from a peer containing the filename to download and sends the contents of the requested file back to the requesting peer.

The unregister\_from\_indexing\_server() method unregisters the peer node from the indexing server by sending an "unregister" message to the indexing server using a TCP connection.

The query\_indexing\_server() method queries the indexing server for the list of peers hosting a specific file by sending a "query" message containing the filename to the indexing server using a TCP connection.

The download\_file() method downloads a specific file from a peer node by sending a "download" message containing the filename to the peer node using a TCP connection and receiving the file contents from the peer node.

The handle\_incoming\_file\_download\_requests() method listens for incoming file download requests on the peer node's server socket and starts a new thread to handle each request.

The register\_files\_with\_indexing\_server() method registers the peer node and its hosted files with the indexing server by sending a "register" message containing the node's address, filename, file hash, and file size to the indexing server using a TCP connection.

The update\_file\_list() method updates the list of hosted files by retrieving the file list from the indexing server using a "get\_files\_list" message and displaying it.

The query\_indexing\_server\_for\_peer\_list() method queries the indexing server for the list of registered peer nodes by sending a "peer\_list" message using a TCP connection.

The delete\_folder() method deletes the folder containing the peer node's hosted files.

The start() method registers the peer node with the indexing server, updates the list of hosted files, and handles incoming user requests through a menu system.

The script uses the argparse module to allow the user to specify the host IP address, port, indexing server IP address, and indexing server port as command-line arguments. If the port argument is not specified, the script finds an unused port.

**Central-Indexing Server**

Peers can register their files with the server, and other users can download those files from the server or from other peers who are hosting those files.

The system consists of two main components: a server and multiple peers. The server maintains a list of registered files and the peer nodes that are hosting those files. The peers register their files with the server and can download files from the server or from other peers.

The server listens for incoming connections from the peers and handles requests from them. The peers can request a list of registered files, register their files with the server, unregister their files, and request to download a file.

When a peer requests to download a file, the server first checks if the file is registered with the server. If it is, the server returns the list of peer nodes that are hosting the file. The peer then randomly selects a node from the list and downloads the file from that node.

Each peer node hosts its own files and listens for incoming requests from other peers. When a peer requests to download a file from a peer node, the node checks if it has the file and sends the file to the requesting peer.

The system uses TCP sockets for communication between the server and the peers, and between the peers themselves. The files are transferred in binary mode. The system also uses a simple protocol for exchanging messages between the server and peers.

**Deployment Script**

The script deploys one server node and five peer nodes. Each peer node is given a unique directory with a prefix, and the PEER\_CODE variable is used to specify the name of the peer code file. The script starts the server process and then deploys each peer node by creating a directory, copying the code file to the directory, and starting the peer process with the appropriate arguments. The script assumes that the peer code file and the server code file are in the same directory.

**Time Analysis**

|  |  |
| --- | --- |
| File size (bits) | Time Taken (Seconds) |
| 128 | 0.00100 |
| 512 | 0.00101 |
| 2000 | 0.00104 |
| 8000 | 0.00099 |
| 32000 | 0.00135 |

This analysis indicates a clear linear correlation between file size and download time, with one exception. Surprisingly, the file with a size of 8000 bits took less time to download than any other file, although this may have been due to chance. I estimate the code to run similarly to the Client-Server Model. As with one peer and one server, acts similar to a Client and Server system.

**Conclusion**

Essentially, my code worked in terms of executing, and connecting to the server. I have problems with listening to sending requests globally to the server. I would have loved to finish the code, but I have 3 midterms and 2 project reports due at the same time as this project, and I did not receive any feedback from the first programming assignment so that I knew how to update this assignment.