CS 550 – Programming Assignment 1

IN-LINE DESIGN DOCUMENT

PEER TO PEER FILE SHARING SYSTEM

Submitted by

Prashant Bhutani (A20488431) **Rahul Sharma (A20470076)**

under the guidance of

Dr. Zhiling Lan

Department of Computer Science

Illinois Institute of Technology

TABLE OF CONTENTS

INTRODUCTION

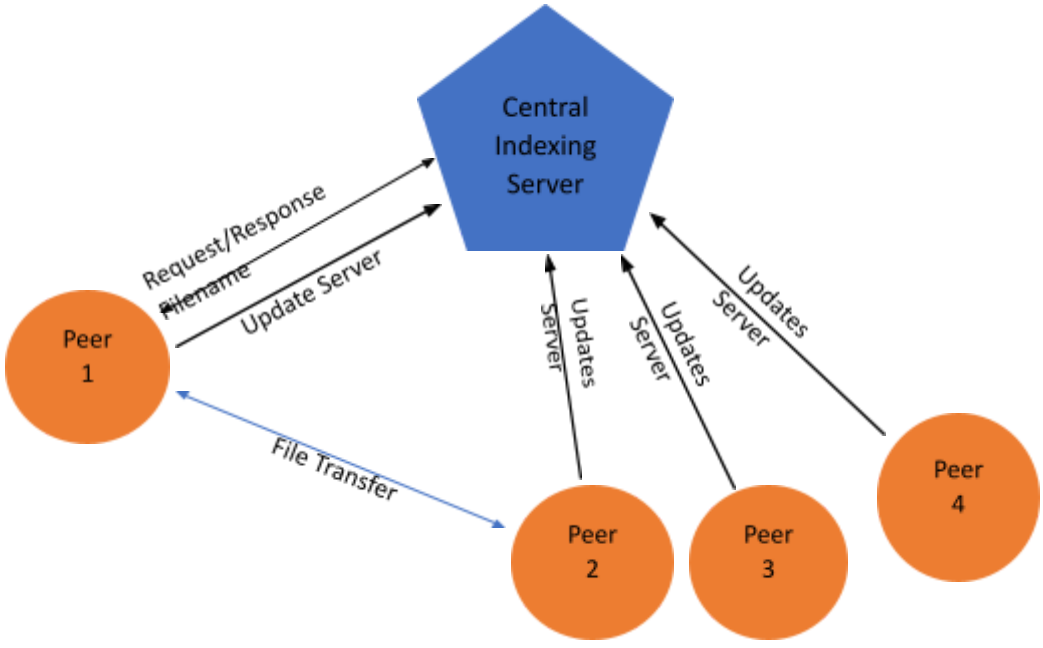
In this project we have utilised Java RMI library to create a Central Indexing Server which contains the information of all the files contained in the file directories of the corresponding peers. The RMI library provides us with stubs which can be registered and deregistered from the indexing server as the status of peer directories change which is handled by the server.

Development Tools used

- Java JDK v11.0.12
- IntelliJ IDEA IDE
- Java RMI Library
- Junit v4.11

Project Structure

As depicted in Fig. 1 we have created a central indexing server which gets updated by the Peers and responds to the requests of peers with the list of peers containing the requested files. Each peer is given an id number when it registers with the server and each peer runs on separate ports at the localhost so they can serve the file download request of other peers and can communicate directly with them.



Once the peer gets a list of peers that contain the needed file(s), the user is asked to choose the peer from which it should retrieve the file and then a new connection between the peers is established to allow the file transfer process which is facilitated by the RMI interface that we have built.

Code Structure and Methods definition

The program structure is divided in 2 major parts, first half contains the definition of server and client functions required by the program to perform required tasks, as there can be multiple scenarios of testing out the implementation of features hence, we have kept the other half of our program structured around building test files for the feature implementations of clients and servers respectively.

1. Models:

We have primarily defined 3 model classes to define the structure of the peers as well as the files contained in them along with the results that we obtain after running the test cases for our project, the models are namely Peer, PeerFile and TestResults

A. **Peer**: This class defines a peer with an Id string and a list of files contained by the peer.
B. **PeerFile**: This class defines the filename and the data contained by the file.
C. **TestResults**: This class captures the test results i.e. it takes in the start and end time for the sequential requests sent by peers to central servers in multiple configurations.

As we can create and add more files and peers to our system so the above classes serve to be essential to define the structure of above

2. Server Nodes:

The server is split in 2 different components namely CentralIndexServer and a PeerServer as the peer clients also behave as servers when other peer approaches them to retrieve a file

A. **Central Indexing Server:** Central Index Server has 3 main tasks i.e., to register and deregister the peers from the index as well as to monitor and update the changes that occur at the peer directories. It also needs to serve the search request from peers and respond to them with the list of other peers. Most importantly all the implementations performed in the class are done in synchronization i.e., serving multiple clients at once and avoiding deadlocks. Let's find below the variables utilised and the methods implemented by the Central Indexing Server class.

1. ***fileNamePeerIdsMap:*** This HashMap variable contains the IDs of peers and maps the files that are associated with the peer in a list datatype.
2. ***peerIdObjectMap:*** This HashMap variable maps a peer to its Id whenever a new peer is formed and registers it with the indexing server.
3. ***Registry:*** This method takes in the parameters which define a peer as mentioned in the Peer model and creates a new peer object and gives it an id which is randomly generated, it then maps its files to a directory. Upon this the method also maps its details in the above hashmaps to register it on the indexing server and returns the ID of the peer.

4. ***Deregistry:*** This method takes in the ID of an already registered peer and then looks up in the fileNamePeerIdsMap hashmap for its details, when invoked it first deletes the mapping of files that come along with the peer from the fileNamePeerIdsMap and then removes the details of the peer from peerIdObjectMap so as to unregister the given peer from the indexing server.
5. ***Search***: This method takes in the name of the required file and checks with the fileNamePeerIdsMap if it contains any peer mapped with the exact matching file name, it then returns the list of such peers.

B. Peer Server: A peer acts as a server when it gets a request of a file retrieval by the other peer (client), the peer server implements the PeerServerinterface to perform the action.

1. ***PeerServer:*** This method is invoked for creating a new server, this method takes in the peerId and directory as an input and uses the Peer model to create and define a peer which acts as a server

2. ***Retrieve:*** This method takes in the peer id as well as filename as requested by the other peer, upon checking if the file exists at its directory the retrieve method copies the file from existing server (peer) to the client directory by creating another file and reading data from the requested file, if the requested file already exists at the client's directory or if there's no data in the requested file, the retrieve function terminates immediately.

3. Interfaces:

Interfaces form an essential part of our project as we have used RMI library which relies on the same interface to be utilised at the server and client side in order to allow us to exchange the data between them. As we have 2 separate server files, we need to create separate interfaces for both of them which extend the Remote class of Java and contain their respective method declarations. The interfaces are implemented by both the central index server as well as the peer server. The interfaces namely are:

A. CentralIndexingServerInterface: Facilitates the implementation of register, deregister and the search methods as utilised by the peers
B. PeerServerInterface: This interface serves peer to peer communication utilised for file download requests and transfer.

4. Logger:

For constantly updating the changes that take place in the file directory of each peer, we have created a directory watcher class, which runs the watch service provided by the Java file system for the path of each peer directory path and keeps a track of any creation of new file as well as any deletion or modification in the already existing files.

This class contains 3 methods which are as follows:

A. **DirectoryWatcher**: This method takes in the Directory path as an input and runs a watch service over this path.

B. **Beginlogin:** This method watches the given path till the server is running and creates an entry whenever a file is updated or deleted and also for the creation of a new file.

C. **Endlogging**: This method is invoked for all the peers when our central server stops running or either for a given peer when it de-registers from the central index server

5. Test Classes:

Similar to the server nodes, we have separate test files to test the implementation of all the required features based on different configurations of peers and files contained with them, which are as follows:

A. Central Indexing Server Test:

Unlike the peers, our central index server has a pretty much well-defined set of tasks to be performed hence this test class is responsible to start the central index server upon reading the environment variables from the ConstantsUtil file that we have provided under the utility directory.

B. Peer Server Test:

This test class comprehensively tests the functioning of peers as a Server, i.e., it accepts the requests from client peers and then reads the file from server directory, creates a new file and then copies the data from the required file to a new file and copies it to the client directory.

C. **Peer Test:** This class tests the working of clients concurrently i.e. multiple clients generate concurrent requests to the server. It creates multiple threads for the peer by utilising the PeerTestThreads class.

6. Threads:

We have created thread classes for testing the capability of servers in handling multiple concurrent requests and synchronising these requests. Following are the classes available for implementing multiple threads in our application

A. **CentralIndexingServerThread:** Creates multiple threads for the indexing server to enable handling the requests from multiple clients at the same time. Needs the address and port to run a new thread of server parallelly.

B. **PeerTestThread**: Creates multiple threads for peers and registers them on the central indexing server to enable them in handling multiple file retrieve requests from the other peer clients by replicating multiple peer server threads. It takes the details of the peer server to be replicated as input parameters including its Id, Directory, the files that have been shared and the files it contains in the directory.

C. **Deregistry Thread**: This class lets deregistering all the threads of a peer from the central indexing server concurrently, takes in the peerId, it's files and centralIndexingServerInterface as input parameters.

D. **DirectoryLogsThread:** Runs the directory watcher on multiple threads to keep the track of changes in the file directory of each peer concurrently.

7. Utility:

These classes serve the other files with the environment variables that are required by our application to work, including but not limited to Host Address, Port number, console messages. Common file functions utilised by the peer systems etc. We have split the utilities in the following classes:

A. **ConstantsUtil:** This class contains all the environment variables and console messages which are needed and utilised by other classes to host our environment.

B. **FileUtil:** Provides commons file methods to other classes which are as follows:

1.retrieveFile: This method is used by the peer server to transfer a file from a peer server's directory to the client server's directory. The method takes in the client peer's and server peer's details as an input parameter as well as the peerServerInterface.

2.readSharedDirectory: This method is used to read the contents of a given directory, i.e. it creates a list of all the files available in the directory which is obtained as a parameter and returns the list.

3.startDirectoryLogging: This method is used to create instances for directory watcher class which implements the logging functionality in the application and also creates the threads for the watch logger.

4. createFile: This method is used to create files for the directory paths of peers, we can specify the filename, the directory path of peer as well the size of file in Kbs in the parameters the file of given filename and size gets created at the specified directory.

5.createFiles: This method utilizes the createFile method and when called creates multiple files of varying sizes at the peers, by running simple for-loops and can be invoked by the peer server test cases as specified earlier.

C. **TestResultsUtil:** This class defines the output for the tests that we run, which are divided into 2 parts:

1. getSearchResults: This method takes in the test results which are yielded when the clients get their search requests served by the central server a i.e. the begin and end time of the request to calculate the average time taken for all the requests (which can be sequential or parallel depending on the option that user selects)

2. getRetrievalResults: This method takes in the test results which are yielded when the clients get their file retrieval requests are served by the other peer servers i.e. the begin and end time of the request to calculate the average time taken for all the requests (which can be sequential or parallel depending on the option that user selects)

8. Code Flow

The PeerToPeerFileSharing class: As our application is going to start from the main class, we have covered all the possible scenarios in this class. When the program executes, it prompts the user to input the number of peers between 3 to 10, upon which in the "./shared" path relative to the code Directory, a separate folder for each of the peers is created and gets registered with the central server. After that the program asks to select one of the 4 options which correspond to the test which user wants to execute which are as follows:

1. **Central Indexing Server Test:** This option runs the instance of central indexing server test which in turn tests if the methods that are defined in the central indexing server i.e. the central indexing server should be able to serve the peers with the list of peers containing the file which is requested to it. The test also takes into consideration if concurrent requests are handled by the server effectively.
2. **Peer Server Test:** This test option calls the Peer server test class and comprehensively tests the functionality of peers when they act as server, which means they must be able to handle the request from peer clients as well as provide the file download functionality by implementing the retrieve method defined in the peer server class. The test also considers if a peer is able to handle requests from multiple clients at the same time.
3. **Client Sequential Call Test:** This option tests the performance of a peer as a client and invokes the sequential testing method of the peer test class, which makes multiple requests by peers to the central indexing server in a sequential manner, i.e. one request is followed by another one.
4. **Client Parallel Call Test:** This option tests the parallel running of peers by calling the parallel peer testing method available in the peer test class, which after creating multiple peers and registering them with the central indexing server, makes many parallel requests to the central server.

As all the files' directories get registered and the user selects a test method, the required outputs are obtained in the CLI which are explained in the Output documentation.

9. Assumptions :

We have designed all our peers and the central index server to run on one machine hence we have provided the relative paths of the peer Directories and the central index server only searches those directories to serve the peer client requests

Scope of Enhancements:

We have already made minor enhancements to the requirements for example: calculating both the retrieval as well as search average time, making sequential as well as parallel requests from the client to the server, the above modes can be set by the user and are very user friendly. However we can further improve the user experience by adding a layer of GUI to our application, this can be done by technologies like Java servlets, applets etc.