Design Document

Basic Theory:

We use Java RMI to implement the communication between Index Server and Peers. Since the RMI has already realized the thread for each client communication with server, we only need to create threads in Peer, one is used for keeping monitoring files in directory and the other is used for showing menu for the user.

There we list each code with design description.

1. **IndexI.java**

The Remote interface for Index Server. It lists two methods: registry() and lookup().
The Peer need this interface to receive and know the object from Index Server.

2. **IndexServer.java**

The code for Index Server.
It implements those two methods and main function listed in IndexI.java:

- registry(): This method receive IP and files names from Peer to register. If the files is not null, it will put IP and files names into a ConcurrentHashMap, which will safely save data from multiple Peer synchronously. If the files is null, it means the peer is exiting and so the Index Server will remove this IP from this ConcurrentHashMap.
- lookup(): This method receive a file name from Peer, then Index Server will search the data in ConcurrentHashMap one by one to find exact same file name. When it find a same name under a peer, the peer IP will be added into an ArrayList. Finally, this ArrayList will be returned to Peer.
- main(): It creates a IndexServer object, registers RMI on default port 1099, and bind the server on “rmi://127.0.0.1/IndexServer”. Then it is waiting for Peer to connect.

3. **PeerI.java**

The Remote interface for Index Server. It lists one methods: retrieve().
Other Peer need this interface to receive and know the object from this Peer.

4. **Peer.java**

The code for one Peer.
It implements these methods and classes:

- retrieve():
    - This method receive the name of file wanted to be downloaded. Then this peer will search the files on the default path ”files/”. If the required file does not exists, return null. If has, it create an array of bytes to read all bytes from the required file, then return this bytes array.
- registerMonitor:
    - This thread class will automatically update the files in the default path. When anything changed, it will automatically call registry() to Index Server. In the run method, we set an infinite while loop with a sleep method to realize this function. In my setting, it will sleep

2s per loop.

- In the loop, it will pick up all the flies names which contains(.txt) and save them to ArrayList “newfiles”. Also when pick one name, it will be compared to the old ArrayList “files”. If “files” does not contain this name, we know something changed. In addition, if the files number in “newfiles”, is different with that in “files”, we also know something changed. Under these situations, we save the “newfiles” to “files” and call registry with this new “files”. In this way, the data updated.

- peerClient:
  - This thread class will show the menu for users and supply corresponding methods.
  - Firstly it will create thread registerMonitor and run it for automatically registry. Secondly it will get object from Index Server by Naming.lookup().
  - It provide four options to user.
    - Option 1: Set loop time. It used for testing the performance as a large sequence requests. It will affect the loop time on Option 2 and 3.
    - Option 2: Lookup and download. Firstly, it prompt user to input the file name. Secondly, it call executeLoopup() to call lookup function on Index Server and get the IP list. Then it call executeDownload() to ask user choose one IP and download the file to this peer’s default file path.
    - Option 3: Set a timer to call Lookup. We can appoint a time to start the lookup function on that time.
    - Option 4: Exit. Firstly it will interrupt the registerMonitor thread to close the update. Then registry null to the Index Server to remove the IP of this Peer in Server’s CurrentHashMap. Finally, call Sysetm.exit(0) to quit the program.
  - It has four methods to support those options.
    - getSelection(): This method get a integer from user with the limit that the number should between lower bound and upper bound. First, it prompt user to input an integer and check whether it is a integer. If it is, then check whether it is be between lower bound and upper bound. If all these are right, return the number. Otherwise it requires user to input again.
    - executeLookUp(): This method call lookup() to Index Server in appointed loop times. It will calculate the total execution time and calculate the average time for one lookup. Finally it return the IP list which was got from Index Server.
    - executeDownload(): This method will check whether the IP list contains IP. If it has, it will require user to choose one IP and get the peer object from the selected Peer. Then call retrieve to get the file data and write it to the new local file.
    - TimerLookup(): This method will require user to set a time(HH:MM:SS) and create a timer based on this time with today’s year, month and day. In addition, the millisecond will set as 000 uniformly. Then prompt user to input a file name. When the system time reach the set time, timer will call run method wicth will call executelooptime(). In this way, we can run multiple server on same time and get the performance (average time).
- main(): It creates a Peer object, registers RMI on specific port (e.g. 1001, each peer keep different port), and bind the server on specific URL (e.g. rmi://127.0.1.1:1001/Peer). Then it is waiting for other Peers to connect.

Possible Improvement:

1) In IndexServer, we can also create an CurrentHashMap to store <filename, IP>, which can be used for lookup() rapidly since the key is filename. However, it will incur more cost on registry() period. Therefore, it’s a trade-off between lookup() and registry().
2) This time we only distinguish the files by its name, since the functions registry and lookup are also only based on file name. However, in the real situation, there may exist different files with same name. Maybe we can calculate MD5 to compare whether different.