CS550 Programming Assignment 1 (PA#1)

A Simple Peer to Peer File Sharing System

Submission:

- ***Due by 11:59pm of 09/23/2018.***
- ***Late penalty: 20% penalty for each day late.***
- ***You may work individually or in a group of two for this assignment.***
- ***For the groups with 2 members, only one submission listing both members is needed. A 1-page document is required to clearly list CONTRIBUTIONS made by each member.***
- ***Please upload your assignment on the Blackboard with the following name: Section_ LastName_FirstName_PA1.***
- ***Please do NOT email your assignment to the instructor and TA!***

1 The problem

This project has two purposes: first to make you familiarize with sockets/RPCs/RMIs, processes, threads; second to learn the design and internals of a peer-to-peer (P2P) file sharing system.

You can be creative with this project. *You are free to use any programming languages (e.g., C/ C++ or Java) and any abstractions such as sockets, RPCs, RMIs, threads, events, etc. that might be needed. Also, you are free to use any machines such as your laptops or PCs.*

In this project, you need to design a simple P2P system that has two components:

1. **A central indexing server**. This server indexes the contents of all of the peers that register with it. It also provides search facility to peers. In our simple version, you don't need to implement sophisticated searching algorithms; an exact match will be fine. Minimally, the server should provide the following interface to the peer clients:
    - *registry(peer id, file name, ...)* -- invoked by a peer to register all its files with the indexing server. The server then builds the index for the peer. Other sophisticated algorithms such as automatic indexing are not required, but feel free to do whatever is reasonable. You may provide optional information to the server to make it more 'real', such as the clients' bandwidth, etc.
    - *search(file name)* -- this procedure should search the index and return all the matching peers to the requestor.
2. **A peer**. A peer is both a client and a server. As a client, the user specifies a file name with the indexing server using "lookup". The indexing server returns a list of all other peers that hold the file. The user can pick one such peer and the client then connects to this peer and downloads the file. As a server, the peer waits for requests from other peers and sends the requested file when receiving a request. Minimally, the peer server should provide the following interface to the peer client:
    - *retrieve(file name)* -- invoked by a peer to download a file from another peer.

Other requirements:

- Both the indexing server and a peer server should be able to accept ***multiple*** client requests at the same time. This could be easily done using threads. Be aware of the thread synchronizing issues to avoid inconsistency or deadlock in your system.
- No GUIs are required. Simple command line interfaces are fine.
- ***Each peer should have an automatic update mechanism***. If a user modifies or deletes some files registered at a server, the effect should be reflected to the server in time. For example, if a user deletes a file on the disk, the server should be notified in time and also remove the corresponding item from the index server.

2 Evaluation and Measurement

Deploy at least 3 peers and 1 indexing server. They can be setup on the same machine (different directories) or different machines. Each peer has in its shared directory (all of which are indexed at the indexing server) at least 10 text files of varying sizes (for example 1k, 2k, ..., 10k). Make sure some files are replicated at more than one peer sites (so your query will give you multiple results to select).

Do a simple experiment study to evaluate the behavior of your system. Compute the average response time per client search request by measuring the response time seen by a client, such as 500 sequential requests. Also, measure the response times when multiple clients are concurrently making requests to the indexing server, for instance, you can vary the number of concurrent clients (N) and observe how the average response time changes, make necessary plots to support your conclusions.

3 What you will submit

When you have finished implementing the complete assignment as described above, you should submit your code on blackboard.

Each program must work correctly and be ***detailed in-line documented***. You should hand in:

1. **Output file**: A copy of the output generated by running your program. When it downloads a file, have your program print a message "display file 'foo'" (don't print the actual file contents if they are large). When a peer issues a query (lookup) to the indexing server, having your program print the returned results in a nicely formatted manner.
2. **Design Doc**: A separate (**typed**) design document (named *design.pdf* or *design.txt*)of approximately 2-4 pages describing the overall program design, , and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made).
3. **Source code and program list**: All of the source code and a program listing containing in-line documentation.
4. **Manual:** A detailed manual describing how the program works. The manual should be able to instruct users other than the developer to run the program step by step. The manual should contain at least a test case which will generate the output matching the content of the output file you provided in 1. The manual ***must*** include screenshots for each step.
5. **Verification**: A separate description (named *test.pdf* or *test.txt*) of the tests you ran on your program to convince yourself that it is indeed correct. Also describe any cases for which your program is known not to work correctly.

6. **Performance results.**
7. **Source code:** All of the source code listed in 3.

Please put all of the above into one .zip or .tar file, and upload it on Blackboard. The name of .zip or .tar should follow this format:

Section_LastName_FirstName_PA1.

Please do **NOT** email your files to the professor and TA!!

4. Grading policy for all programming assignments

- Program Listing
  - works correctly (including the quality or effectiveness of manual)---------- 50%
  - in-line documentation -------- 15%
- Design Document
  - quality of design ---------- 15%
  - understandability of doc ------- 10%
- Thoroughness of test cases ---------- 10%