CS2510 Project 1
Simple File Sharing System

Instructions:

- *Due date: oct 25, 2019 (3 weeks)*
- *You should work in a group of two for this project. If you cannot find a teammate, please let the professor know.*
- *Design doc should be sent to your TA (lol16@pitt.edu) for review by the end of the first week the project released, that is, oct 11, 2019.*
- *Name your file as "Student1_Student2_P1" with student name as "LastName_FirstName".*
- *Grades for late programs will be lowered 10 points per day late.*
- *All questions about the project should be directed to the TA: lol16@pitt.edu*

The problem

This project has two purposes:

1) Learn the design and internals of a peer-to-peer (P2P) file sharing systems
2) Understand how different design choice influence the performance of the system

You can be creative with this project. You are free to use any programming languages (C, C++, Java, etc.) and any abstractions such as sockets, RPCs, RMIs, threads, events, etc. that might be needed. The program should be able to execute in the elements.cs.pitt.edu machines.

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

- **An indexing server.** This server indexes the contents of all of the peers that register with it. You can assume the indexing server is at a well-known location. It also provides search facility to peers. It is you design decision for the search algorithm. 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. It is your design decision for the indexing algorithm.
    - search (file name) -- this procedure should search the index and return the matching peers to the requestor. It is your design decision which matched peer or peers to return.
- **A peer.** A peer is both a client and a server.
  As a client, it can request a file from the indexing server using lookup(filename). The indexing server returns other peers that hold the file. The client then connects to a (set of) peers 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:

- obtain (filename) -- invoked by a peer to download a file from another peer.

Other requirements:

- You should come up with two designs of the system for comparison. That is, implement two algorithms that achieve the same functionality, but have different characteristics.
- 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 thread synchronization issues to avoid inconsistencies or deadlock in your system.
- One of your design decisions is the structure of the network. An indexing server can be part of a peer or can be a stand-alone server.
- No GUIs are required. Simple command line interfaces are sufficient.

Evaluation and Measurement

Peers and servers can be setup on the same machine (different directories) or different machines. Each peer maintains a directory with at least 10 files of varying sizes, all of which are indexed at the indexing server. Make sure some files are replicated at more than one peer sites (so your query can give you multiple results).

Do a simple experiment to evaluate the behavior of your designs with following metrics.

1. Number of messages exchanged
2. Number of bytes transferred
3. Compute the average response time per client for search and obtain request by measuring the response time seen by a client

- Measure the systems with M files, N number of sequential requests with frequency f. the size of M, the values N and f should be able to specified as a command line arguments. Make sure that the arguments are big enough to make a difference in experiments (if they're very small, there will be very little difference in response time, for example).
- Also, measure the systems when multiple clients are concurrently making requests to the indexing server, for instance, you can vary the number of concurrent clients (C) and observe how the metrics change, make necessary plots to support your conclusions.

What you will submit

When you have finished implementing the complete assignment as described above, you should submit your project by email to your TA. Each program must work correctly and be documented. You should hand in:

1. Design Doc: A separate (typed) design document of approximately 2-3 pages (including diagrams) describing the two programs' design, and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made).
2. Source code containing in-line documentation.
3. Manual: A 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.
4. Performance results and discussion, including a table with arguments and the values you experimented with.
5. A Report, showing the metrics you collected, and an analysis (why did this curve go up/down here or there)

Please put all of the above into one .zip or .tar file, and email it to your TA. The name of .zip or .tar should follow this format:

Student1_Student2_P1.{zip | tar}; note that Student should be substituted with LastName_FirstName.

Grading policy for all programming assignments

- Program
  - works correctly (including the quality or effectiveness of manual) ------- 50%
  - specify evaluation parameter as command line argument -------- 5%
  - in-line documentation -------- 5%
  - Thoroughness of evaluation and quality of report ---------- 25%
- Design Document
  - quality of design ---------- 10%
  - understandability of doc ------- 5%

Recall that grades for late programs will be lowered 10 points per day late.