# CS 550 Programming Assignment #1

## A Simple Napster Style Peer to Peer File Sharing System

***Instructions:***

- ***Due date: 11:59PM on Monday, 09/21/15***
- ***Maximum Points: 100 points***
- ***Maximum Extra Credit Points: 10 points***
- *This is an individual assignment, although you may work in groups to brainstorm on possible solutions; your code implementation, report, and evaluation must be your own work.*
- *Please post your questions to the Piazza forum.*
- *Only a softcopy submission is required; it must be submitted to "Digital Drop Box" on Blackboard; please zip all files (report, source code, compilation scripts, and documentation) and submit it to BB; Name your file as this rule: "PROG#_LASTNAME_FIRSTNAME.{zip|tar|pdf}". E.g. "Prog1_Raicu_Ioan.tar".*
- *Late submission will be penalized at 10% per day (beyond the 7-day late pass).*

### 1 The problem

This project has two purposes: first to get you familiarize with sockets/RPCs/RMIs, processes, threads, makefiles; second to learn the design and internals of a Napster-style peer-to-peer (P2P) file sharing 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. Also, you are free to use any computer as long as it runs Linux. Your assignment will be graded in a Linux environment, and you will loose many points if the TAs cannot compile and run your assignments.

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:
   - obtain(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.

- For full credit, you need to only support text files; for 10 points extra credit, you can add support for binary files.
- No GUIs are required. Simple command line interfaces are fine.

**Extra Credit:**

- **5 points:** Add support for both text and binary files in the transfer
- **10 points:** Add support for data resilience by allowing a configurable replication factor (system wide)

## 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; if you are having trouble running on the same machine due to overlapping ports on your processes, feel free to setup 4 virtual machines (VM) on your system and run each peer and indexing server on a separate VM. 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 1000 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 & Grading

When you have finished implementing the complete assignment as described above, you should submit your solution to 'digital drop box' on blackboard. Each program must work correctly and be detailed in-line documented. You should hand in:

1. **Source Code (30 points):** You must hand in all your source code, including with in-line documentation.
2. **Makefile/Ant (5 points):** You must use Makefiles or Ant to automate your programming assignment compilation
3. **Manual (10 points):** 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.
4. **Compiles Correctly (10 points):** Your code must compile in a Linux environment.
5. **Output file (10 points):** 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. Describe any cases for which your program is known not to work correctly
6. **Design Doc (15 points):** You must write about how your program was designed, what tradeoffs you made, etc. Also describe possible improvements and extensions to your program (and sketch how they might be made).
7. **Performance Evaluation (20 points):** You are to conduct a basic performance evaluation
8. **Extra Credit (15 points):** Binary files support and data replication.

Please put all of the above into one .zip or .tar file, and upload it to 'digital drop box' on blackboard'. The name of .zip or .tar should follow this format: "PROG#_LASTNAME_FIRSTNAME.{zip|tar|pdf}".

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

Grades for late programs will be lowered 10% per day points per day late (beyond the 7-day late pass).