Programming Project 2

CS5352 Advanced Operating Systems Design
Fall 2018

Due Date: 11/2, 2 p.m., soft copy via Blackboard.
Late submissions are accepted till 11/9, 2 p.m., with 10% penalty each day.
No submissions accepted after 11/9, 2 p.m.

1. The Problem

This project has two purposes: first, to get you familiarize with sockets/RPCs/RMIs, processes, threads; second, to let you 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) and any abstractions such as sockets, RPCs, RMIs, threads, events, etc. that might be needed.*

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

1. **A central indexing server**. This server indexes the content 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:
    - *register(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 done using threads.
- No GUIs are required. Simple command line interfaces are fine.
- You should develop a Makefile to automate the compilation.

2. Evaluation and Measurement

Deploy at least 3 peers and 1 indexing server. 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 1,000 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 solution on the Blackboard.

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

1. 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. A design document of approximately 3-4 pages describing the overall program design, a description of "how it works", and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made). This document should include a section describing the tests you performed and performance results.
3. All of the source codes including the Makefile containing in-line documentation

Please put all of the above into one .zip or .tar file, and upload it to the Blackboard.

4 Grading criteria

- Program (all source codes including the Makefile)
    - Works correctly ------------- 50%
    - In-line documentation -------- 15%
- Design Document
    - Quality of design ------------ 15%
    - Readability of the documentation ------- 10%
- Thoroughness of test cases ---------- 10%