EEL-4930/EEL-5934
Principles of Computer System Design
Final Project - Assigned: 10/23/2013

Due dates:
10/30/13 5pm (Definition of group+topic)
11/6/13 5pm (Problem definition and proposed approach)
12/6/13 5pm (Final report+code)

This is the final project for this class. You may work individually, or in a group of 2-3 students for this project.

In the "recommended track" for this project, you will take the FUSE filesystem that has been developed over the semester and extend it to support an advanced feature. These features can be based upon concepts already discussed in class or others that will either be covered in lectures, or additional systems design concepts that we may not have time to cover in class but addressed in the textbook, including reliability, performance, privacy, or consistency. This project will consist of the following subcomponents:

1. Declaration of group, topic, and general problem. Due **10/30/2013 5pm via Sakai** as an attachment submission. This should be 1-2 paragraphs long and identify what problem you plan to study in your project.
2. Problem definition. This includes a short background study, the specific problem you will be solving, and proposed solution to the problem. Due **11/6/2013 via Sakai submission**. This should be approximately 1 to 2 pages long.
3. Final submission. This includes your software and a 10 to 20 page document describing your system, modeled after a typical technical report. Due **12/6/2013 via Sakai submission**. Requirements are as follows:
   a. Code – Good use of modularization, limiting complexity, and readability; examples of these can be shown by looking at the source code and relationship of distht → simpleht → python's dictionary and of hw2 → htproxy → xmlrpc and the associated documentation and testing code.
   b. Documentation (submitted as a PDF):
      i. Background – Describe your problem, why it is interesting – cite papers, web sites, or other documentation. Remember your solution does not need to be novel.
      ii. Implementation – Your implementation documentation should make it so that your code does not need to be read to appreciate your work. It should also not be so overloaded with information that reading the source code would be easier than reading the documentation itself.
      iii. Testing mechanisms – When implementing new projects, tests must be established that focus on ensuring that no bugs were created. Identify potential problem areas and describe the tests used to verify that those problems do not exist. The purpose of a testing mechanism is to verify that there are no functional regressions. Example: test.py.

iv. Evaluation mechanisms – What are the benefits of your work? Design a scheme that evaluates your work with the previous approach. In some cases, this may have overlap with the testing mechanism. The point of this portion, though, is to show the enhancement, whereas a testing mechanism is used to verify there are no regressions. Example: time for very_long_io to execute in hw3.
v. Evaluation – Present your evaluation, showing graphs and describing the interesting results from your work
vi. Potential issues – List any potential issues that exist with the addition of your new feature as well as any components that were left undone

Submit your final work via Sakai as follows:

- Submit only PDF and .py files for your documentation and code
- Zip all your files
- If submitting as a group, only one person needs to submit
- Non-conformant submissions will lose significant points

Notes: You are allowed to modify and extend distht.py, simpleht.py, and hw3.py (for example, distht/simpleht may not have primitives required to support your project). You should discuss these revisions in your overview and include them as diff files with the final submission (diff -cB old new). If you have questions, comments, or thoughts, you are encouraged to visit Professor Figueiredo or David during office hours.

Suggested project topics – these are based on ideas that have been discussed in class:

- Supporting multiple clients and concurrent operations: up to homework #3, you have focused on a single client, but a more interesting use case of the file system design needs to support multiple clients. To do so, it is important to support atomic operations on objects stored in the file system, and provide support for locking files (e.g. using the POSIX API supported by FUSE).
- Improving performance: We have overviewed several approaches to identify and address performance bottlenecks in a system. This topic seeks to identify performance bottlenecks and seek to implement and evaluate performance improvements.
- Fault tolerance – if the file system supports multiple servers and nodes fail, how can you prevent data from being lost?

Other possible topics – these may not have yet been covered in class, but may be in an area of your interest and provide an opportunity for you to investigate them in practice:

- Integration with other object storage frameworks – e.g. memcached is a widely-used key value object store. One topic would be to port the FUSE file system to use memcached as storage back-end, and extend the file system with additional functionality (e.g. support a directory hierarchy rather than a simple flat model) and characterize performance.
- Project proposals in other areas that are not in the area of file systems may be entertained, but you **must** talk to the instructor prior to submitting your definition of group/topic.