Introduction

In your undergraduate education, you took a course in operating systems. Part of this course covered file systems. In this course, the file system was usually on your local computer. File systems can have different features. For redundancy, a file system might have a file in two data centers. If one file is damaged or not available, you still have access to the other. Suppose there is a read request, either copy of the file can be accessed. When a file is updated, both copies need to be updated. If one copy is not available, your implementation must wait until it is available. Thus, you must queue the requests. Another feature might be security using encryption. This is what you will be doing. The details are below.

The project is to build a defensive distributed file system and add security. After week 4, you will need to submit your project design doc. At the end of this semester, you need to submit a write-up and code for your project. A demo presentation will be held during the last two weeks of the semester.

You will work in groups of 3–4 for the final project. Your group is required to turn in the code and a short write-up describing the design and implementation of your project, and to make a short in-class demo presentation about your work.

Each group will submit **one** copy of the project report and code. At the end of the semester, we will ask group members to evaluate their team members' performance. We want each group member to have equal contributions to the project. Usually, each member will get the same grade for a project.

Deliverables (One submission per group. DO NOT submit individually)

1. Form a Group (before the end of 2nd week)

You need to form your project group of three or four students in the first week. If you can ‘t find a team member, we will assign you to a group randomly. After the first week, you need to submit your group member list via blackboard.

2. Working on the project consistently throughout the semester is critical. As in professional software development, you will perform an **Update Standup** weekly:

https://docs.google.com/document/d/1NkntjzQFMx7ooU5ND59n4LYN6LuNlE44OisOiyIvik8/edit?usp=sharing

Give a three-point summary of both your work from last week and your plans for this week. This is due each Friday at midnight.

3. Design doc (before the end of the 4th week)

You must submit a design doc describing:

- The design pattern : UML graph.
- Toy example to illustrate how your design works.
  - In this example, please work with just one request, one thread. So you can understand if your design works or not.

4. Code (2 weeks before the final)

- You will maintain your code in the git repository and submit your git repository via blackboard.
- Benchmark your result with 100K requests: random read, random write, random read and write. Clearly define your evaluation criteria. You should at least measure the runtime.

5. Write-up wiki (2 weeks before the final)

Write a document of your project in all detail (benchmark should be put into the experiment part), and turn it in along with your project's code by the final deadline. Take a look at the list of write ups from past years (2012): http://css.csail.mit.edu/6.858/2012/projects.html and 2013:

- http://css.csail.mit.edu/6.858/2013/projects.html)

to get a sense of what this writeup should look like.

6. Presentation (the last week (tentative))

Prepare a 10 mins in-class demo presentation of your project.

- 2 slides about background,
- 2 slides about your design and
- 2 slides about your benchmark.

The presentation will be held online. 5 mins for Q&A.

Project: Requirements for the Encrypted File System Project

At this point, you have a working distributed file system. Your next requirement is to allow users to store data on untrusted distributed file servers (P2P). At a minimum, your file system should meet the following requirements:

Requirements for the distributed file system

- This is a P2P file system (https://en.wikipedia.org/wiki/Peer-to-peer).
- Users can create, delete, read, write, restore files.
    - Bonus is you can do async read and write.
- A client should always see the latest version of a file, or at least that a client should never see an older version of a file after it sees a newer one.
- Users should be able to set permissions on files and directories, which also requires that your file system be able to name users.
- The system should be able to deal with concurrent write and read.
- File names (and directory names) should be treated as confidential. The data stored in each Peer should be encrypted.
- Users should not be able to modify files or directories without being detected, unless they are authorized to do so.
- The communication between Peer to Peer should be encrypted.
- A malicious file server should not be able to create or delete files or directories without being detected.

The final result of these projects should be a functional file system implementation that meets the above requirements. You can implement your prototype in any language you want, such as Python or C++ or Go. You can decide how the file system client and server should be run. One reasonable design would be to have the file system client provide a minimal shell environment that allows users to perform the operations described in the above requirements.