

Solutions are due at the beginning of class on the due date and must be **typed** and neatly organized. Late homeworks will not be accepted. You are permitted to discuss these problems with your classmates; however, your work and answers must be your own. All questions should be directed to the teaching assistant.

Problem 1 : Storage security.

Unix filesystems use a form of restricted access control lists (ACLs) in order to protect files. In this limited version 3 fields of 3 bits each are used to classify protection. The 3 fields indicate access for either the owner of the file, the group of the owner, or anybody and the 3 bits in each file indicate whether the member specified in the field has r - read, w - write, or x - execute permissions.

Suppose we have the following scenerio: Greg is writing a new book and has asked for some help from some (potentially questionable) colleagues: Larry, Curly, and Moe.

- Greg should be able to invoke all operations on the file.
 - Larry, Curly, and Moe should be only able to read the file (Greg just wants suggestions, not corrections).
 - Everyone else should have no access.
- (a) How would one accomplish this with the standard UNIX permissions?
- (b) What would change (from part a) if one had a true, unrestricted access control list system?
- (c) If one had a capability system how would one accomplish this?
- (d) Now Greg decides to open up access to read only for everyone, and upgrade Larry's, Curly's, and Moe's permissions to read and write. But he also now wants to disallow a fourth colleague, Shemp, from having read access. How would one do this with the standard UNIX FS ACLs? What about with unrestricted ACLs? and capabilities?

Problem 2 : Disconnected operation.

You may have noticed Greg frequently brings his laptop to class. This is because he is busy working on important professor stuff and not because he is playing games in the back of the room. When he writes papers with your friendly TA, they use **CVS** (Concurrent Versions System) to resolve conflicts when multiple people update the same file. This can be *highly irritating* because “resolving conflicts” is a nontrivial task requiring manual intervention.

Your friendly TA also sometimes takes his laptop home (a TA’s work is never done, unlike a professor’s) and his CVS updates late at night are usually hampered by his low-bandwidth modem connection to CMU. It would be much more pleasant if your friendly TA could just use a file system designed for offline operation and automatic conflict resolution. Unfortunately, your friendly TA apparently lives under a rock because he has never heard of Coda, Bayou, Ficus, etc. Instead, he assigns you to design your own simple disconnected operation file system: **FTAFS**, the Friendly TA File System.

- (a) Write two or three sentences (each) on how Froese and Bunt’s design axes (caching unit, replacement policies, etc.) apply to the problem at hand—specifically, what values/parameters you think are appropriate for FTAFS.
- (b) Now change your assumptions: this summer, your friendly TA’s compatriots are leaving for internships in geographically disparate areas of the country. They’ll still be working on papers together; however, they’ll only be connected to CMU’s network about once a month. How will your FTAFS design need to change to accommodate this?

Problem 3 : Alternate storage technologies.

FLASH memory is a non-volatile semiconductor memory which is used in many portable devices as the main stable storage. Its access speed for reads is equivalent to that of regular RAM, but writing speed is limited by the fact that regions of memory must be erased before they can be written. These regions are called erasure units. Clearing erasure units is very slow (on the order of a second) and they can only be erased a finite number of times before they fail (around 10,000–1,000,000 times). Therefore, in order to update data in place, the entire erasure unit must first be read, then erased, then the modified unit re-written.

- (i) Why does this limitation pose a problem for normal UNIX file systems?

Log-structured file systems were invented to mitigate the cost of small writes on disk drives. Writes are coalesced in memory into segments and then written out to disk into a clean segment.

- (ii) Could LFS be used to manage files on FLASH devices? How would LFS solve the problems of FLASH?
- (iii) In LFS, segments are sized such that the time to read or write a segment is larger than the time to seek to the segment. What would determine the segment size in a FLASH device?
- (iv) Fragmentation occurs in two ways in LFS for disks. First, there can be unused space within a segment. Second, as the disk becomes full, there are fewer long extents of free segments for large files to be written into sequentially. Fragmentation in this way hurts sequential bandwidth. How would the use of FLASH memory affect fragmentation?

Problem 4 : Wide-area caching.

Buzzwordsgalore.com is about to go out of business. Your “combination handheld personal video recorder/potato masher” failed miserably in the marketplace. Incidentally, because of that incident your CEO fled the country in the company’s yacht—in his absence the board has named *you* the new CEO.

It’s up to you to save the company. Last night you had a brilliant, marketable idea: a product for caching static web content in proxy servers. Your new service, “BWGalore.net”, will be a distributed cooperative caching network—that is, you’ll set up a collection of proxy caches that communicate with each other to share hot objects.

- (a) Your first major design decision is to either **cache near the content** (placing caches close to the major Internet backbones and connection points) or **cache near the clients** (putting one cache at each of your client’s sites). What are the advantages and disadvantages of each? Which would you choose?
- (b) Your proxy will have a large memory (1 GB) for caching “hot” pages, plus disk space (100 GB) for caching “warm” pages. There are several ways to store objects on disk for rapid lookup:
 - (1) You could create N cache directories (where N is large) and always place new objects in the emptiest directory, keeping a “object-to-directory” lookup table on disk.
 - (2) You could create N cache directories and hash each object name to a directory.
 - (3) You could create a directory structure based on object names. For example, the URL “http://host.subdomain.domain.tld/dir1/obj2.html” would go in /cachedir/tld/domain/subdomain/host/dir1/obj2.html.
 - (4) You could create a single cache directory and store all objects there based on their full object name (converting “/” to “-”, for example).

Compare each of these schemes in the areas of (i) number of disk accesses needed to access an object; (ii) ability to co-locate related objects (e.g., a web page and the JPEG images on that page); (iii) “fair sizing” among directories (e.g., all directories are about the same size). Choose any file system that you’ve learned about this semester and base your answers on that file system.

- (c) Why might it be useful to relax consistency requirements for the disks on your web proxies—that is, *not* guarantee that data are in a recoverable state on disk after a crash? Would you recommend doing this for web *servers* for the same reasons? Why or why not?
- (d) When the disk is very busy, and you run out of memory to store “hot” pages, you can free memory by **throwing out “unwritten” objects** (not yet written to disk) or **throwing out “written” objects** (already on the disk). What are the advantages and disadvantages of each? (E.g., which scheme reduces the load on the disk, etc.?)

Problem 5 : Backups.

A Storabyte company offers for their top line of storage systems (100s of disks, GBs of cache, tens of thousands of I/Os per second) a solution, called RemoteSite, which protects against data loss due to a natural disaster such as an earthquake. In this setup, another storage system is installed at a different physical location and the two are connected together (the speed of the link can vary from FibreChannel to a T3 line - 45 Mb/s).

The protocol that keeps the two storage systems connected and up-to-date works as follows. The primary system logs all writes to a special location in memory. This log contains the addresses and the new data written out to the disks and coalesces multiple writes to the same location into one record. Every 15 seconds, the log is sent asynchronously over the link to the backup system. The log is replayed there effectively propagating the changes to the disk of the backup system. When the replay finishes, the primary system is notified and the memory that held that portion of the log is freed.

- (a) Why did the designers come up with this complicated logging scheme? Specifically, what if they chose a conventional mirroring approach, where every write goes to both systems?
- (b) Does this system completely prevent against data loss? If yes, say why, if not, justify your answer and state what the window of vulnerability is (i.e., how long are data unprotected).

Recall the snapshot mechanism in Network Appliance's File Server. Although it is used to recover from accidental power failure, it could be used in the RemoteSite backup solution with a primary and secondary File Servers.

- (c) List what changes, if any, are needed in the snapshot mechanism, and briefly describe how the RemoteSite protocol would be implemented using your (modified) snapshots. Try to retain as much of the design of the RemoteSite protocol as possible.