

# Homework #2

Due: 4 Mar 2011, 23:59 PST

Homework must be submitted electronically to <csci555@usc.edu>. It should have a subject of "Homework 2." You must submit plain text without embedded formatting commands or markup (ASCII or UTF-8 are acceptable). That means, among other things, no postscript, no Microsoft Word, no PDF, no FrameMaker, no TeX, no groff, no DocBook, no HTML, no XML, and no JavaDoc. Do not submit your homework as an attachment to your e-mail. Do not base 64 encode it. Do not Rot13 encode it. Plain text. You may PGP sign it, but are not required to do so. Do not PGP encrypt it. If you submit something that is not unmarked-up text, it is functionally the same as turning nothing in.

Homework turned in on the due date is not penalized, one day late is 25% off (that is the grade will be multiplied by 0.75), 2 days late is 50% off, and 3 days late is 75% off. No work will be accepted more than 3 days late. I will generally use the Date: line of the mail, but should the situation merit it, I am not above looking through mail system logs to confirm the submission time. I should not have to mention it, but forging a Date: line to avoid a late deduction is grounds for an F.

Do the work yourself. Computer science is a collaborative science, and I encourage you to talk over the ideas in the homework with other students. *However*, the final submission, that is, *the text of the homework*, must be composed individually by each student. If you hand in homework that is identical to another student, you risk failing the class. (In fact the only way that you would not fail the class in such a circumstance would be if one student had copied another student without the knowledge of the copied student; the copied student would not be penalized.) That is an awfully large risk for 10% of your total grade. Do the work yourself.

As with all work for csci555, this work is subject to the USC code of Student Conduct.<sup>1</sup> Read it, learn it, live it. Should you have any questions on how to apply the code, do not hesitate to contact me or the Office of Student Judicial Affairs and Community Standards.<sup>2</sup> Should it prove possible, do not plagiarize work from sources outside the class. Plagiarizing homework is grounds for *failing the class*. It is perfectly all right to properly cite external sources, should you find some that are useful.

Answers will not be graded on their beauty of expression. Answers will be graded on whether they show a logical approach and sensible explanation. Short, simple sentences are fine. What is important is that your ideas are clear to the reader, and that they answer the question. Of course, no answer will be penalized because it is beautifully expressed, either.

Each question has equal weight.

## Homework

1. Grapevine [Birrell82] and the Domain Name System[Mockapetris87] are structured very similarly, yet the DNS provides an upper bound on how old data can be in the system. Consider the two systems in the case of a long term network partition where some name servers for a given domain/repository are in each partition. [You may assume that all if two nameservers cannot communicate, no client can communicate with both of them.] Explain how clients see the namespace in each system. What fundamental design decision is embodied in this behavior?

**Answer:** In the DNS case, after the stale data timer goes off, all partitions except the one containing the primary will lose the ability to resolve names in this domain.

---

<sup>1</sup> <http://web-app.usc.edu/scampus/university-student-conduct-code/>

<sup>2</sup> <http://www.usc.edu/student-affairs/SJACS/>

Under Grapevine, all partitions continue to be able to resolve and update names, though when the partitions are re-merged, there may be significant changes to the name space as multiple updates are applied. Similarly, clients in different partitions will see different histories of the namespace.

The Grapevine designers would rather have all partitions continue indefinitely and risk an inconsistent history of the namespace: they prefer availability to consistency. The DNS designers would rather halt service than have long time inconsistency, making the opposite choice.

**Grading:** Four points for each description, 2 for the decision.

2. Consider creating a namespace for works of art that are characterized by the creator's name, the kind of art, and the year the art was created. [For the purposes of this question, assume that these attributes all have single values - that is no collaboration, multi-year projects, or mixed media art.] Another namespace is being created for the same works of art, but this one is characterized by the location of the art, by country, museum, building and room. [In this case each work of art is indivisible and fits in a single room; similarly museums and buildings are not shared across countries or museums.]

In either case you can use a hierarchical or a non-hierarchical namespace design. Explain which kind of namespace you prefer for each space. (Note: you may choose the same kind of namespace for each kind, or different ones)

**Answer:** In the first case the various attributes in the name are all equally important. One can imagine scholars who naturally group art by creator and those that group them by time. Putting them into a hierarchy implies making one the most important. Consider the "top level directory" of the namespace. If you put creator on top, browsing the namespace requires looking for making a decision about that first (or looking across all authors to make a decision about a lower category). This kind of information is more easily organized non-hierarchically.

The reverse is true of the second, location-based, namespace. These names reflect a containment hierarchy. When searching a museum, knowing the country it is located in is reasonable, and probably narrows the scope of the name resolution intuitively. In fact, without knowing which museum is being considered, the building probably makes little sense. This namespace is better represented hierarchically.

**Grading:** 1 point each for picking the kind of name space and another 4 for each explanation.

3. When we discussed Weighted Voting[Gifford79], we had a lively discussion about overheads of various operations on files, and on the synchronization behavior that would be used to set read and write quorums. Give an example of a shared file that exhibits the following overheads:

- A file more often read than written where access overhead is dominated by consistency control
- A file more often read than written where access overhead is dominated by data transfer
- A file more often written than read where access overhead is dominated by consistency control
- A file more often written than read where access overhead is dominated by data transfer
- A file read and written equally often

I am expecting a description of the file, not a file name.

- Answer:**
- A file more often read than written where access overhead is dominated by consistency control

A small configuration file, for example a shell configuration. This is small enough for a single RPC to deliver the data.

- A file more often read than written where access overhead is dominated by data transfer

A large executable file meets this criteria. OpenOffice in the Linux world, for example, or other large complex program.

- A file more often written than read where access overhead is dominated by consistency control

There are various small state files that are written by applications. For example, editors often write files containing changes since the last save in case they are interrupted (autosave files). Some applications write small or zero-length files that are used for application synchronization as well. They are written often, say once per arriving mail message, but only accessed when there is contention for a resource.

- A file more often written than read where access overhead is dominated by data transfer

A large shared log file, such as a system error log, often meets this criteria. Such files can easily grow to the size of executables, and assuming no regular parsing or monitoring, they are more often read than written.

Similarly, some applications and system services regularly log transactions from which they can recover lost work. These logs can be much larger than the small editor recovery files - e.g. Coda's transaction logs.

- A file read and written equally often

A backing file for a database can meet this criteria. So can large log files if the files are regularly parsed for unusual entries. That is sometimes done by intrusion detectors.

**Grading:** 2 points each. Describe the file and say why it meets the criteria.

4. Two events are causally related if one is the direct or indirect result of the other occurring. Events can be causally ordered by the Lamport Logical Clocks algorithm[Lamport78].

Give an example of causally related events. Give an example of causally ordered, but not causally related events. Explain the relationship between the sets of causally ordered and causally related events in a system running the Lamport Logical Clocks algorithm.

- Answer:** If I get e-mail from my brother and then send him a check because of the message, those events are causally related.

If I get an e-mail from my brother, and then one from my father, and send my father a check based on the contents of his e-mail, the events of my brother sending an e-mail and my father getting a check are causally ordered (my brother's mail came before my father getting the check) but are not causally related.

If the clock algorithm is running, the set of causally related events is a subset of the causally ordered ones. All events can be ordered by the algorithm, but only some will directly cause others.

**Grading:** Three points for each example and 4 points for the explanation.

# References

- [Birrell82] Andrew D. Birrell, Roy Levin, Roger M. Needham, and Michael D. Schroeder, "Grapevine: An Exercise in Distributed Computing," *Communications of the ACM*, ACM, vol. 25, no. 4, April 1982, 260-274,
- [Mockapetris87] Paul Mockapetris, "Domain Names - Concepts And Facilities," *RFC-1034*, November 1987,
- [Gifford79] David K. Gifford, "Weighted Voting for Replicated Data," Seventh Symposium on Operating Systems Principles, ACM, December 1979, 150-162,
- [Lamport78] Leslie Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM*, vol. 21, no. 7, July 1978, 558-565,