

Homework #2

Due: 23 Feb 2009, 23:59 PST

Homework must be submitted electronically to <csci555@usc.edu>. It should have a subject of "Homework 2." You must submit ASCII text without embedded formatting commands or markup. 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 ASCII 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 ASCII, 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.¹ 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.² 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. In the Li and Hudak paper (ivy) [Li89], Section 5.5 alludes to an algorithm for distributing copy sets, rather than maintaining a centralized list at the owner. Explain why read fault requests do not need to be forwarded to the owner. Explain what messages the new locks at processors holding a read only copy are intended to serialize.

Answer: A read fault doesn't change ownership of the page, just adds a new member to the copy set. Using the distributed representation of the copy set in section 5.5, any processor holding a read-only copy of the page can return the page and add the requester to the copy set. The page the processor holds is the same as the owner holds, or the page would have been invalidated. Adding the faulter to the local copy set insures that the faulter will receive future invalidations. There is no data that resides at the owner that must be manipulated.

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

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

The locks serialize read fault messages from faulters and invalidate messages from the owner. Faults received before the invalidation imply that this processor is responsible to forward invalidations to them; faults after an invalidation are the responsibility of the new owner. Serializing the invalidations with the faults guarantees that the nodes all agree with the state that fault messages encounter. This is directly analogous to making sure that two write faults each find the owner, even when both are in transit at the same time.

Grading: 5 points for each discussion.

2. Consider a hierarchical naming system that contains information about classical artists. The hierarchy is divided by kind of art (painting, sculpture, music, etc.) and then by artist and finally by the name of the artwork. You can assume that the designers of the namespace have appropriately resolved collisions between artists of the same name and artworks of the same name by the same artist. One would find the paintings by DaVinci in the directory `/painting/DaVinci`. Give an example of a search which is efficient in this naming system and one which is not efficient and explain why. Suggest a naming system that supports more flexible searching in these same categories and explain why that namespace is more flexible.

This question is just asking about searching for names in the name space, not for content in the art.

Answer: Any search that follows the hierarchy is reasonably efficient in that only the path components that need to be traversed are resolved. For example finding all the paintings by DaVinci would basically be listing the files in one directory with little waste. A less efficient search would be one that did not match the hierarchy: find all artworks by DaVinci. That search would have to walk all the top level pathnames looking for a DaVinci pathname underneath. Even worse would be finding all artworks with the same name by any artist, which would have to walk the entire space.

For general searching, an attribute based namespace may be more efficient. The attributes remove the ordering constraints on the names, which potentially removes some of the inefficiencies in searching the name space.

Grading: 3 points a piece for each example and 4 points for the attribute based name space and description.

3. Systems for editing program source often provide search features tuned for that task. For example, by placing the cursor on a function call and pressing a key sequence the editor may reposition the cursor to the definition of that function, even if the function resides in another file. This is an example of a content search system that operates in a narrow context.

Explain how such a system would identify function calls and definitions in the source. Explain how such a system would determine what files to include in its search. Explain how the idea of a closure might come into play in identifying the files to include in the search.

Answer: Identifying function calls and definitions requires the system to be able to parse program source. Such a system may not need to understand the whole of the programming language syntax, but would need to know enough to find those tokens.

Identifying the files to include is more tricky. The system should understand the basics of the development system's naming conventions - where compilation searches for files. For C this might include the location of system headers. For Java and perl this may include an understanding of how package and class names map into the directory structure. Very sophisticated systems may understand ideas like interface descriptions used to create stub routines.

Closure may be important here in that a user's environment may influence how the programming language resolves names. C compilers can take parameters that influence what headers they include or where they search for included sources. Perl and Java can be told where to search for packages. Furthermore such instructions may be encoded in the calls to the compilers or in the environment variables of the user. More "sophisticated" users might even use symbolic links to change their view of the file system. The more that the search system understands these indirect influences, the better its search can be.

Grading: 2 points for the first question, 4 for the second and 4 for the third. Successful answers need not provide all the examples I did. In fact no examples are required. In the closure answer, a full credit answer doesn't have to get into the possibility of locally tweaking the namespace.

4. In class there was a discussion about Grapevine's eventual consistency[Birrell82] and the result that an external administrator could not state precisely when a name was added or removed from a group RName. Describe a system where this ability - the ability to know precisely when a name was added or removed from a group - is required and one where it is not. Your explanation should include why the capacity is critical or why not. For your system that requires group removal, do not use a military or associated system.

Answer: There are certainly many answers to this, but here is an example answer. The key point is that examples where the group semantics are critical cannot be mere performance improvements, but cases where the semantics is critical to the correct operation of the system.

Consider a system that requires three redundant servers to store a piece of data for survivability or reliability. For example three servers to store a financial transaction so that if any two disappear the institution still have up to date records. If the servers being used for this backup are kept in an RName, the changes to that RName have to be serialized and agreed upon by all servers and outsiders to make sure that the system is providing the required reliability constraints.

Consider a system that is using the group names as a hint. For example a file sharing system (used to distribute perfectly legal free operating system images), that keeps nodes hosting a given image in a group. The group is a hint; the requester will contact each name in the group in turn. Hosts that do not respond or that don't have the file are simply ignored. Precision in the content of the hint is not required by definition.

Grading: 5 points for each example. The description should be clear as should why the precision in group membership is essential.

References

[Li89] Kai Li and Paul Hudak, "Memory Coherence in Shared Virtual Memory Systems," *ACM Transactions on Computer Systems*, ACM, vol. 7, no. 4, November 1989, 321-359,

[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,