

Homework #1

Due: 6 Feb 2009, 23:59 PST

Homework must be submitted electronically to <csci555@usc.edu>. It should have a subject of "Homework 1." 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 Mesa paper[Lampson80], we pointed out a case where using Broadcast gives correct results and Notify gives incorrect ones. It is easy to extrapolate that Broadcast is always the best primitive to use, but that is not the case. Construct an example where Notify is both correct and more efficient than Broadcast and explain why.

Answer: Any time when there is high contention and the processes waiting on the condition want access to interchangeable resources, notify will be correct. For example, if the memory allocator in the Mesa paper only allocated one size block of memory, notify would be correct. It would also be more efficient because when a single block was returned only one process, the one that could

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

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

use the returned memory, would be scheduled. If the block were returned and a broadcast used, every process blocked on the condition would be re-activated, and the majority would return to sleep. Using Notify avoids extraneous process wakeups.

Also give credit for suggesting the dot tuple in the Linda matrix multiplication example, though in practice it may not matter much because the tuple is held for such a short time. You can imagine implementing the dot tuple as a monitor-protected data object that returns the parameters for the next matrix element to calculate. Initially the processes would all be waiting to acquire the dot tuple and would manipulate it one-by-one. This is a high contention situation for the same element, so notify would be correct. The question is whether it would be more efficient than broadcast. The first broadcast might put all the processes on the run queue and each would enter the monitor, manipulate the tuple and release the monitor before another process tried to access the dot tuple and wait. However, it's also possible that the processes might all wind up re-queued. Notify will not be less efficient, though.

Again, the keys are that make notify both more efficient and still correct is that any request may be satisfied by the notification and that the resource has high contention. Any such example is fine.

Grading: 5 points for the example and 5 for the explanation. The explanation should both argue that Notify is correct and that it avoids extraneous process wakeups.

2. In Section 3.5 of Birrell and Nelson's RPC system[Birrell84] they describe a system for dispatching RPC packets to server processes that includes process identifiers provided by the participants. What is the purpose of these identifiers (in your own words)? How important is it that they are completely accurate, and why? In what way are they like Mesa Notify commands?

Answer: The process identifiers simplify the process of looking up the correct receiving process by assuming that the same server process will serve all requests from the same client process. The Ethernet driver on the server can avoid a search for ready server processes if the suggested server is waiting for a packet, thereby enhancing performance.

The suggested identifier need not be accurate at all. If the suggested process doesn't exist, or is busy, the system acts as if no process identifier had been suggested. Because the system must operate without process identifiers in place - for example the first RPC to a server will have no way to guess an accurate one - the correctness of the system does not rely on them.

These identifiers are like Mesa notify commands in that they are hints to enhance the performance of the system. In both cases, the correctness of the system does not rely on them (other than maybe Naked Notifies in Mesa), but performance may be greatly enhanced by providing the (well founded) speculation to the system.

Grading: 3 points for explaining how the identifiers work, 4 points each for explaining that they need not be accurate and why and 3 points for explaining the analogy to Mesa Notifies.

3. If one is going to implement a lock in Linda[Carriero85], one must do it using the `in` command. In CSP[Hoare78] one has the choice of using either the send message or receive message primitive. Explain why.

Here are the outlines of two implementations of a lock using CSP constructs. In both cases each process that may use the lock is identified by a positive integer. The process representing the lock is called the lock manager.

In the first implementation, the lock manager is implemented very much along the lines of the semaphore discussed in class, except that the `val` variable is only allowed to be zero or one. This is accomplished by adding a term to the guard that accepts `V()` messages that only accepts

them when `val` is 0. Processes acquire the lock by sending a `P()` message to the manager. When that message has been received, the process holds the lock. It returns the lock by sending a `V()` message to the manager.

The second implementation also represents the lock as a process. The lock manager uses a two-phase protocol to assign the lock. When a process requests a lock it sends a `request_lock()` message to the manager. The requester then waits for a `lock_granted()` message from the manager. The requester holds the lock when it receives that message. To release the lock the process sends a `release_lock()` message to the manager, and the lock is released when that message is delivered. The manager keeps a variable with the identifier of the process holding the lock (or -1) in it. It loops always accepting `request_lock()` or `release_lock()` messages, and enqueueing the identifiers making the requests. When the lock is available (i.e., is -1) the manager sends a `lock_granted()` message to a process with a pending request. When the lock becomes free (on receipt of a `release_lock()` message) the manager assigns the lock to another process, if a request is waiting.

In both cases I am omitting details of how the manager enforces the requirement that only the lock holder can release the lock, but that is straightforward. You may want to think it through, however.

Explain an advantage of using the local queue in the second implementation. Describe the costs of the second implementation.

Answer: In Linda, only `in` blocks, and blocking is an essential part of acquiring a lock. In CSP, either primitive may block, so either may be the basis for the implementation - as the two examples show.

The advantage of using an explicit queue in the second implementation is that the implementation is free to assign the lock to processes however the implementer sees fit. The implementer can define fairness. In the first case, the queueing is done within the language and the implementer is at the mercy of the language implementer.

The cost of the internal queue is both in additional memory to hold the messages and, more importantly (probably) the additional message in the lock protocol. An extra message implies an extra system call and makes synchronization a more expensive operation.

Grading: 3 points for explaining the blocking, 4 points for explaining the advantage, and 3 for the costs. Split out the cost points as 2 for the overhead and an additional 1 for the memory.

4. JINI[Waldo99] adopts a decentralized layout for both their service location networks and for their authentication (each component decides which keys it trusts to sign code). Athena[Champine90] adopts a centralized model for these choices (Hesiod locates services from a single configuration database, and Kerberos provides central authentication).

Explain the problems that each system was trying to solve that led to that part of the design, and how their design addresses those problems.

Answer: Athena was providing students access to computing hardware for use in instructions. The resources were all controlled by a central entity and a system goal was uniformity of operation. Under those conditions, central configuration is possible - because there is one owner and access granter - and desirable because of the uniformity requirements.

JINI was trying to foster innovation of ideas in a loosely controlled environment where individual users controlled their machines. In this case there was no central authority to make decisions about who was trustworthy - though certainly suggestions may have been made in corporate environments. Furthermore, allowing new services to grow and encouraging incre-

mental upgrades implies that uniformity is less important than agility. A decentralized layout facilitates both possibilities.

Grading: Five points each. There are other possibilities, too, but I'm looking for a consistent argument and grounding in the papers.

References

- [Lampson80] B.W. Lampson and D.D. Ridell, "Experiences with Processors and Monitors in Mesa," *Communications of the ACM*, vol. 23, no. 2, February 1980, 105-117,
- [Birrell84] A. Birrell and B. Nelson, "Implementing Remote Procedure Calls," *ACM Transactions on Computer Systems*, ACM, vol. 2, no. 1, February 1984, 39-59,
- [Carriero85] Nicholas Carriero and David Gelernter, "The S/Net's Linda Kernel," *ACM Transactions on Computer Systems*, ACM Press, vol. 4, no. 2, 1986, 110-129,
- [Hoare78] C. A. Hoare, "Communicating Sequential Processes," *Communications of the ACM*, vol. 21, no. 8, August 1978, 666-677,
- [Waldo99] Jim Waldo, "The Jini Architecture for Network-centric Computing," *Communications of the ACM*, ACM, vol. 42, no. 7, July 1999, 76-82,
- [Champine90] George A. Champine, Daniel E. Geer, Jr., and William N. Ruh, "Project Athena as a Distributed Computer System," *IEEE Computer*, IEEE, vol. 23, September 1990, 40-50,