

CS110 - Principles of Computer Systems

Midterm Exam

(Total time = 50 minutes, Total Points = 50)

Name: (please print) SOLUTIONS

In recognition of and in the spirit of the Stanford University Honor Code, I certify that I will neither give nor receive unpermitted aid on this exam.

Signature: _____

This examination is close book and close notes. You may not collaborate in any manner on this exam. You have 50 minutes to complete the exam. Please write your answers on the exam. Note there is one problem per page so the amount of space provided does not necessarily provide an indication of the expected length of the answer. In other words, do not feel compelled to fill every nanoacre of the exam with writing. Before starting, please check to make sure that you have all 8 pages.

Question	Points	Score
1	8	
2	7	
3	7	
4	7	
5	7	
6	7	
7	7	
Total	50	

1. (8 points) Modern processors contain memory mapping units (MMUs) that support paged virtual memory. In the MMU configuration used by most operating systems, virtual addresses generated by the processor are mapped to different physical memory addresses. If you compare the virtual address to the corresponding physical address you will notice that the least significant bits (i.e. low order bits) of the two addresses are the same while the most significant bits differ greatly. (E.g. virtual address `0xbaadf00d` could map to physical address `0xdeadb00d` but not `0xdeadbeef`). Explain why this is the case.

The lower order bits of the virtual address (12 bits in the example) represent the offset into the 4KB page. The higher order bits of the virtual address represent the virtual page number. The memory management unit translates the virtual page number into a physical page number (the internal mechanism here is often more than just a single table-lookup). Since the physical page can lie anywhere in the the physical address space, the physical page number can vary greatly from the virtual page number. The offset (those 12 lower-order bits), however, is carried over as-is into the physical address, so that the offset into the physical page in the physical address is the same as the offset into the virtual page in the virtual address.

*Many students talked about "preventing fragmentation" in their answers. It is true that one of the goals of virtual memory is to *hide* memory fragmentation from user programs -- that is, a user program's data may be scattered across many physical pages, while appearing to be contiguous in virtual memory. However, virtual memory does not prevent fragmentation. In most cases, if an answer involved talking about fragmentation, it was not coherent and did not discuss the address translation mechanism that was at the core of this question.*

2. (7 points) Assume you are given a system that supports concurrent operations using 16 threads on a large one gigabyte shared data structure. The operations are divided into two types: small and large. Small operations only touch a few parts of the shared data structure and run in a short amount of time. Large operations touch most of the share data structure and take a long time to complete. Half of the threads (the *small op threads*) only perform small operations and the other half (*the large op threads*) performs only large operations. Assume there are two synchronization approaches: Method L uses coarse-grained locks to protect the shared data, while Method N protects the data using non-blocking synchronization. For each approach, describe what would happen to the large and small op threads.

Under method L: Large-op threads would acquire the coarse-grained locks and hold on to them for a long time while they process various parts of the data structure. Meanwhile, small-op threads, which only need to touch a small part of the data structure -- and thus could potentially run in parallel with some of the code in the large-op threads -- would be waiting for a disproportionately long time. Similarly, other large-op threads would be blocked waiting for a coarse-grained lock when they could be modifying parts of the data structure that are not being touched by the lock-owning large-op thread at a given point in time. The result of all this is that concurrency/parallelism is severely impaired in the system.

Under method N: Small-op threads would enjoy great concurrency under this approach; they will be able to copy, modify, and atomically-compare-and-swap the relevant parts of the data structure and, since the length of their operations is small, the chance that another thread will have modified the data from under them is relatively low. On the other hand, large-op threads will need to copy larger/more chunks of the data structure and take longer to make their modifications. Most importantly, their atomic-compare-and-swap is very likely to fail since the chance that another thread (large or small) will have modified the data from under them is relatively high. The large op threads would have keep retrying the operation but could potentially never succeed, thereby suffering from starvation.

A number of people said that the large op threads would livelock under method N. In fact, they'd just starve. Livelock is starvation plus preventing other useful work from being accomplished. We didn't take off points for this, since we ended up arguing amongst ourselves for a while before agreeing on the difference between livelock and starvation.

3. (7 points) Assume you have started a new engineering job at a hot Silicon Valley startup and while looking through the source repository you see the following code fragment:

```
/* Pausing this thread for one second seems to make the
system more stable */
sleep(1);
```

Using concepts discussed in class, write an educated guess at what is going on here with the system.

Between the comment and the code we have only a limited amount of information about the system and that information is vague enough that practically anything could be going on with the system. Full credit was given for answers that seemed plausible giving the evidence provided. Points were taken off if the answer was inconsistent (e.g. non-blocking synchronization causing deadlocks) or required unlikely assumptions (e.g. the code is embedded in the network card driver).

Common strong answer:

It appears that changing the timing of a thread makes the system more stable. Race conditions are timing dependent errors so a likely scenario is a race condition in the system that the coders couldn't figure out how to fix so they perturbed the timing with the sleep statement to avoid the error more often. Some students pointed how this was a scary sign of the engineering talent of the company.

Other answers that got most credit (-2 points) were claiming the sleep changed the execution to avoid some livelock or deadlock. One would hope if the code was really doing this it would have been reflected in the comment. The comment seems to indicate the coder didn't understand why the system was more stable and hence didn't understand what was going wrong. Putting a sleep to avoid a livelock situation shows an understanding of the problem.

4. (7 points) What is the purpose of intermediaries in client/services architectures? Give an example.

The main point of intermediaries in general is to provide a buffer for messages between the client and server. Full credit was given for stating this. The example straight from class was email - an intermediary allows one end point to be off, and the email to be delivered when it is turned on.

Partial credit between 1 and 5 points was given for answers ranging from a clear statement of the main point with a weak supporting example (or vice versa) to statements that apply only to specific types of intermediaries (e.g., trusted intermediaries, DNS, etc.). Examples of other partial credit statements include load balancing and adding reliability.

5. (7 points) Would it ever make sense to have a client/service architecture (complete with RPC communication) where all the clients and services are processes on the same machine? Justify your answer.

I accepted a number of different answers.

Most people compared a client-service architecture (CSA) running on one machine to a single process implementing the same functionality. In that case, CSA has the advantage of increased modularity. Hopefully when one process misbehaves or crashes, it doesn't take down the whole system. The downside is that RPC overhead may slow down the system.

Some people compared CSA running on one machine to CSA running on multiple machines. In this case, the overhead of RPC is decreased, but there's more fate-sharing --- now all processes in the architecture are tied to the fate of the machine.

Some people suggested running a CSA application on a single machine using virtualization --- i.e. each process of the application would run in a separate VM. This is acceptable.

*Using virtualization as an example of CSA where all the clients and services are on the same machine (the clients are the guest OSes and the services are provided by the hypervisor) is acceptable. But it's not correct to say that CSA on a single machine *is* virtualization, or that virtualization is the only way to do CSA on a single machine.*

Other good examples of when you might use CSA on a single machine are microkernels and X11. I didn't give full credit for suggesting that communication between user-level programs and the kernel is an example of CSA, since syscalls aren't really RPCs.

I accepted answers which said that you might want to run a CSA application all on one machine for testing.

*I didn't give full credit for answers which simply told me about the benefits of RPC/CSA -- you needed to say something specifically about the benefits of running CSA on a *single machine*.*

I gave very little credit to people who said that it never makes sense to use CSA on a single machine.

It's important for this question that you understand the difference between "processes" and "threads". A number of people lost credit for this.

6. (7 points) In a naming scheme, would it ever make sense to have more *names* than possible values in the *universe of values*? Justify your answer.

Yes, it would. Consider the case of booleans: we can have 3 variables a, b, and c, and then we now have more names than the number of possible values in the universe of values (true, false).

A number of students got the right answer, but didn't do a great job justifying it. I typically took 2 points off for that. Other students also lost some (typically 2-3) points for mixing up namespaces and/or contexts with the idea of a universe of values. Students answering "no" got very few points.

7. (7 points) Explain why a complex software project may not be able to bring the number of bugs in the system to zero even if they had a large engineering team focused solely on a goal of getting rid of all the bugs.

A complex software project has a large number of modules, and a number of interactions between modules that is quadratic in the number of modules. It is difficult for a single individual or group to know all the modules and the interactions between them well enough to trace all the bugs in the code. Race conditions and other concurrency-related bugs might further add to the difficulty of finding all the bugs.

However, there is a large engineering team dedicated to fixing all the bug, so presumably the team will be able to track them down. But even if the source of the bugs can be determined, it is likely that in such a complex software project, fixing one bug changes the system in a way that introduces one or more bugs. And some bugs may not be fixable given the current architecture of the system. Thus the number of bugs in the system will never get to zero.

I typically didn't require students to have all of this for an answer. Students tended to answer more along one line of reasoning than the other. The real crux of the answer is that fixing a bug in a large system often causes more bugs to appear, so students who didn't mention that typically lost 2 points. Some students mentioned "emergent properties" as a reason that the number of bugs will never go to zero, and those students typically got full credit as long as there weren't other problems with the answer.