Stanford University
Computer Science Department
CS 240 Quiz 3
Winter 2003

March 21, 2003

This is an open-book exam. You have 50 minutes to answer eight out of ten questions. Write all of your answers directly on the paper. Make your answers as concise as possible. Sentence fragments ok.

NOTE: We will take off points if a correct answer also includes incorrect or irrelevant information. (I.e., don't put in everything you know in hopes of saying the correct buzzword.)

Question	Score
1 - 2	
3 - 4	
5 - 6	
7 - 8	
9 - 10	
total	

Stanford University Honor Code

In accordance with both the letter and the spirit of the Honor Code, I did not cheat on this exam nor will I assist someone else cheating.

Name and Stanford ID:

Signature:

Answer **8 of the following 10 questions** (i.e., skip two) and, in a sentence or two, say *why* your answer holds. (5 points each).

1. Eraser: give a *short* code example (and the intuition behind it) showing how we can get a race if only one thread ever writes to a shared variable.
   Intuition: second guy depends on the value of the first to make a decision and then do a calculation that assumes the value was the same.

thread 1	thread 2
	x = 10 (initialization)
acquire(xlock);	
if $x > 5$	
	x = 2
y = x*3;	
else	
y = 0;	
release(xlock);	

2. IRIX does not have superpages. Jimbo wants superpages. *Briefly*: how could you trick IRIX into using superpages using disco? Give a concrete example of doing so is worse than doing it in the OS itself. If you statically partitioned memory across guest OSes, might you actually see performance improvements?
   Disco would try to allocate contiguous machine pages to both the virtual pages and "physical" pages used by the guest OS. A complication is that Disco doesn't know about memory regions (e.g, segments, memory-mapped objects that the OS is using), or how big they will be; they could either hack the OS or try walking its page tables to get this information. Not really: If you statically partition physical memory, supporting superpages can be a bit easier since the machine pages are not as scattered which makes getting contiguity easier, but it shouldn't matter that much.

3. Last time I checked, Jerry Saltzer and Dave Clark believed that CPUs should not have errors. Can you reconcile their stance with the end-to-end argument?
   The naive end-to-end tells us that it may make little sense to reduce the residual error rate of any system component much below that of the others. However, we often (rightly) want the lower levels of a system to be more reliable than the higher levels (e.g., applications). This is true because: (1) more different systems include them so an error affects more people; (2) errors may harder to diagnose (e.g., hardware bit flipping is a real pain).

4. If you were an OS, give two ways to tell if you were running on a virtual machine?
   It is very difficult for VMs to virtualize time. Thus, one thing you could do it run specific operations and see if they took much longer than they should (which would indicate that something else was running underneath you). Another would be to send a packet that contains the IP address that you thought you had over a network and snoop all traffic to see if that matched the address in the packet.

5. Now that you've had to wade through a bunch of experimental evaluations, what experiments would you use to evaluate the VMS approach to virtual memory?
   Different types of workloads to test different VM features: (1) Microbenchmarks to test specific features: e.g., programs with small/large/mixed resident set sizes. Many microbenchmarks are listed in the superpage paper, e.g., Tree. (2) Realistic workloads such as a shared compute server workload like the Stanford Elaine machines workload with multiple selfish user processes. Either use real deployment or playback collected traces. (3) Varying the parameters used in VMS to test effect on performance.

6. From Figure 6-1 in the livelock paper: Why does the line go up? What determines how high the peak is? What determines how fast it goes down?
   (1) because you are forwarding packets: 1-1 line that passes through the origin. (2) MLFRR: the rate that it can process beofre it loses stuff. (3) starts losing stuff after doing wasted work — this makes the system less efficient (friction in a sense).

7. Your SPED webserver serves files remotely mounted using an NFS file system. What performance changes would you expect to see?
   Hard to say. Some things get better, others worse. Better: server buffer cache utilization— going to server's memory should be better than going to disk; async I/O will now work, since you're just listening on a network connection. Worse: you could be pulling data across the same link you are using for web traffic; additionally, the latency goes up — you will try to hide this with more outstanding connections, but it can be more costly.

8. Concretely, what would happen if the exokernel allowed a libFS to write out metadata the held a pointer to another disk block before that block was initialized?
   If a crash happened between the write and initialization, then after the system rebooted, the data in the disk block could give the application control of disk blocks it should not have.

9. Jimbo likes everything, including LBFS. It's his birthday so you implement LBFS as a local file system on his disk using the Boneh-in-a-Box co-processor, which makes both fingerprint and signature computations infinitely fast. Compared to a traditional local file system, give one way that local-LBFS (1) reads will be slower, (2) reads will be faster, (3) writes will be slower, (4) writes will be faster.
   (1) slower because contiguous data may get scattered thoughout the disk. (2) faster because you share different data blocks, which can allow your buffer cache to appear bigger. (3) slower because if a hash matches you have to read the data from disk and compare it (all write hits now require reads). (4) faster because if the data you want to write is already somewhere on disk and in your cache so you can compare, you can just write out a pointer.

10. Assume you wanted to implement the systems we've read about in a libFS. Give one concrete example of how you would benefit if the exokernel: (1) exposed allocation; (2) exposed names; and (3) exposed revocation. (I.e., pick three examples, one for each principle, from whichever systems you prefer, and say how you could implement them better given the principle.)

(1) exposed allocation lets you make decisions about what to allocate (e.g., disk blocks);

(2) physical names let you ask for what suits you best, e.g., you can pick the sequential disk blocks you want for your log segments;

(3) revocation allows you to make your own decisions when resources allocated to you are revoked. E.g., application can choose which pages to page out under memory pressure.