Stanford University
Computer Science Department
CS 240 Sample Quiz 3 Answers
Spring 2004

May 31, 2004

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.)

1. *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 diagnos (e.g., hardware bit flipping is a real pain).*

2. *A race condition in the procedural would be a violation of the monitor invariant, which corresponds to violating the loop invariant in MPP. In the procedural system this could happen because you have external procedures that access data without locks or because multiple entry procedures are not "big enough" to guarantee the invariants you need. The MPP system did not have external procedures, so there doesn't seem to be an analogue for this case, but the second case should hold.*

3. *(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).*

4. *5 best, 1 worst: 5 points 5 best, 3 worst: 4 pts depending on explanation. 3 best, 1 worst: 3 points others, 2pts (+/- depending on explanation)*

5. *If the checkpoint is written early in the log, we will essentially examine the disk twice.*

6. *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 it doesn't know how big regions 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 it can be a bit easier since the machine pages are not as scattered, but it shouldn't matter that much.*

7. *Need a way to tell the beginning/end of log entries if they exceed a single sector. Or, have a checksum for entries. Commit needs to be within a single sector and cannot straddle sectors.*

8. *You will have to recompute all the parity blocks. We took off two points if you said it mattered which order they were written out in. If a data*

disk fails while reconstructing then we cannot recover (these are two faults: invalid parity, and lost data).

9. *Need to recompute all of the parity information – additional crashes are not a problem because only the parity info is written and it will be recomputed anyway. If "crash" is assumed to mean a disk crash, then need to replace disk and recompute its contents using the other disks + parity information. Additional disk crashes will lose data.*

10. *P1 = A xor B xor C and P2 = A xor B xor D.*

11. *Yes. No: 5 pts. -2 for bogus explanation (some get lucky) No. Yes: 0 pts. No. No. / Yes. Yes: 3pts, depending on explanation*

12. *Something about a mix of applications; heavy paging, light paging, varying memory sizes, varying the parameters used to control the lists.*

13. *They claim that "the execution of the operating system in general and critical section in particular is slower on Disco, which increases the contention for semaphors and spinlocks." The intuition behind this is that (1) contention means one or more thread is trying to acquire a held lock so (2) the longer you hold a lock, the more likely there is that someone else tries to acquire it.*

14. *ESX will only probablistically hash memory. It may miss cases. On the other hand Disco will always share memory when two applications read the same block from the same shared disk or when they send page aligned messages. The downside of this is that you had to do something special in both cases: (1) mount a disk as shared or (2) modify the OS to call remap.*

15. *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);
```

16. *(1) slower because contig 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.*

17. *RAID has problems with small writes since you may have to read the parity block and potentially the old data before doing the write. The best case for RAID are writes across and entire "stripe" — in the example, a write that would hit A,B,C,D and P. LFS is perfect for writing large chunks of data. LFS will probabaly get better in some ways. The large RAID has larger capacity, which may well make cleaning easier — more free capacity (less cleaning needed) and larger amounts of data, which allow better seperation. Also, LFS has trouble with sequential reads after contig writes — the parallel disk heads can eliminate many seeks that would be required (as long as the data is smeared across different disks).*