Stanford University
Computer Science Department
CS 240 Quiz 3 Answers
Spring 2006

June 14, 2006

This is an open-book exam. You have 50 minutes to answer 8 out of 10 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 - 5	
6 - 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 eight of the following ten questions and, in a sentence or two, say *why* your answer holds (5 points each).

1. Initial versions of BSD disabled and re-enabled interrupts as the sole means of providing mutual exclusion. Using your knowledge of Mesa: what must BSD do (and why) if a kernel thread running with interrupts disabled sleeps (i.e., puts itself on a block queue and allows another thread to run)? Give an example of a problem this can cause. (Note: you only need to understand Mesa to answer this question, the details of BSD are irrelevant.)

   Interrupt disable = monitor lock. If a monitor sleeps holding the lock waiting on a condition it must release it otherwise no other thread can enter the monitor to make the condition true. This is also true in BSD; if it does not re-enable interrupts this will wreak havoc with all sorts of stuff. So at each sleep point, the kernel re-enables interrupts and then when the thread wakes up, re-disables them. The problem this causes is that this breaks the critical section, which can cause a race condition.

2. You run on an Alpha which only has loads and stores of 32-bit and 64-bit values (i.e., has no way to store and load bytes or shorts directly). What is the complete set of values could you possibly see after running the following code and why?

```
char x = 0;
char y = 0;

  Thread 1          Thread 2
  ------------------------------
x = 1;                y = 1;
```

   If T1 runs then T2 runs non-overlapping we can get (1,1). If they do overlap, then the RMW cycle will ensure that last writer wins, giving (0,1) or (1,0) (this occurs if `x` *and* `y` *are placed in the same word in memory; this assumption needed to be stated).*

3. Describe how to do Eraser-style detection of file race conditions in an NFS file server. In particular explain: (1) what the states in the Eraser state diagram correspond to and (2) what information you would need besides just the RPC arguments for `read` and `write`. Note: You will almost certainly need to augment the NFS protocol and you should use the Eraser definition of what a race is.

   You'll have to add locking. You probably want to track when they open, close file so can filter out when a given access cannot propagate forward (past close) or backward (past open).

   (a) Virgin: just got written.

(b) *Exclusive: one process just reading writing file (emacs).*

(c) *Shared-modified: get a write from another process.*

(d) *Shared: read of another thread. If firest thread closed, then go to exclusive state.*

4. Assume you could classify packets with what application they were intended for infinitely fast. What could you improve in the livelock paper and how?

   Traffic differentiation: One big improvement would be to track which application a packet was for and if its queue was full or it had used too much CPU discard the packet. Right now if any application has a full queue all receive processing is shut down.

5. Explain how you would use the Xen "reorder barrier" when writing out LFS checkpoints.

   You would use it to force all segment sectors previous to the checkpoint to be written out. Otherwise the checkpoint state won't make much sense since you can be missing data that it refers to. Note that forcing checkpoint to be written before subsequent sectors is not necessary.

6. Unlike FFS, Linux `ext2` requires the user `fsync` both a newly created file and its containing directory to guarantee that both the name and file will be persistent. Assume: Linux `ext2` does no synchronous writes to disk, but does implement `fsync` correctly. You `creat` a brand-new file, and successfully `fsync` both it and its directory entry to disk. The system crashes. After a crash the `fsck` repair program discovers that the inode for your file is also claimed by an entirely different directory from a different user. What happened? (This basic bug currently ships with Linux, BTW.)

   If the file reuses an inode from a deleted directory but the deletion has not been written to disk, you can come back and have two pointers to the same thing. Worse: use an indirect block from other file that was not fsynced. Your fsync'd file will use it, as may the other.

7. Map/reduce skips log records that cause problems. Give a justification similar to one from the end-to-end paper for for why this is ok.

   It doesn't make sense to make one component of a system have much lower residual error rate than the others: the raw data map/reduce works with already has lots of errors — since it is stale (out of date) or machines were down when it crawled — thus skipping a small number of records isn't such a big deal.

8. Assume LFS has split a 100MB disk that has 50% utilization into 1MB. Draw the a distribution of segment utilization that will give a write cost of 1.

   write cost of 1 = we never read any data for cleaning. This requires that the distribution is completely bimodal: 50 segments are completely empty (and can be written to w/o cleaning) and 50 are completely full.

9. Lampson talks about using batching for speed. Give four examples from our reading where systems have done this.

   There were a bunch of examples: (1) livelock (quotas), LFS batches writes into segments, Nooks batches XRPC calls, XEN batches page table updates.

10. Give four examples of Lampson hints from the exokernel paper.

    Many possible answers.