Stanford University
Computer Science Department
CS 240 Sample Quiz 2 Questions
Winter 2005

February 25, 2005

These were from open-book exams. In general you had 50 minutes to answer 8-10 out of 10-12 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.)

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 **10 of the following 12 questions** (i.e., skip two) and, in a sentence or two, say *why* your answer holds. (5 points each).

1. Livelock: in Figure 6-3 why does "Polling (no quota)" work badly?

2. The livelock paper has a hack to handle the problem of dropping packets on the "screend" queue. What limit does this solution have for multiple applications getting network data?

3. Assume we have a router that does exactly three things: (1) receives packets, (2) processes packets, and (3) transmits packets. If we use a 3-CPU multiprocessor and put each of these on a single dedicated CPU, can we still livelock? Are there any other problems with this approach?

4. Give two ways that the MTV music video award webserver could suffer from livelock despite the fact that everyone connects to it using TCP, which is flow-controlled.

5. Given the following pseudo-code:

```
// runs when there is a network interrupt.
net_interrupt() {
        while eth1 has packets on receive Q
                // process packets;
        while eth2 has packets on receive Q
                // process packets;
        while eth1 has packets to send
                // send packet
        while eth2 has packets to send
                // send packet
}
```

   List several problems with this code from a livelock point of view and suggest how to rewrite it to eliminate these problems.

6. The livelock paper states you can use one of two approaches in solving livelock: "(1) do (almost) everything at high IPL, or do (almost) nothing at high IPL." As they describe these approaches, in what sense are they actually the same?

7. Give two NFS operations that are not stateless. How could you build an NFS implementation to handle this? (Be concrete: don't just give a couple of buzzwords.)

8. In what way does synchronously writing an NFS operation to disk before replying to the sender violate the end-to-end argument?

9. Suppose Jimbo wants to test an NFS server for livelock and runs a client with this loop:

```
while(1){
  send NFS READ RPC;
  wait for response;
}
```

   Is the NFS server probably subject to livelock? What happens as Jimbo increases the number of client machines running this loop?

10. You start sending NFS operations over a reliable, in-order, duplicate-suppressing network stream (e.g., TCP). Jimbo claims that this will fix all the non-idempotent problems of using NFS over UDP. Is this true? Give your intuition or a concrete counter example.

11. To usher in the new millennium, you rewrite NFS to send operations over TCP instead of UDP. Since TCP provides a reliable byte stream, does it solve the problem that some NFS operations are not idempotent? (Hint: you need to think about more than just lost packets.)

12. As a humanitarian gesture, John H. gives you access to Berkeley's network (you can send arbitrary packets and monitor all network traffic). Give one attack you can do to an NFS client and one attack you can do to an NFS server that would allow you to read, write, or provide arbitrary data.

13. Because NFS is stateless it relies on requests being *idempotent* on the persistent file system state (i.e., you can repeat them over and over with the same effect: the statement "x = 4" is an example of an idempotent operation). Which NFS operations are not, in fact, idempotent? Give one example of how this could cause different behavior than if you were running on a local file system.

14. The NFS paper says that because (1) NFS servers are stateless then (2) the server must write data to disk before replying to the client. Give an intuition and example for why (2) is implied by (1). Note, the paper does a poor job in explaining this, so you will have to be more concrete and insightful than they are.

15. (10 points) You notice that leases inherently involve request-response messages. You decide to try to save network traffic by building leases on top of UDP and doing your own retransmission by setting a timer and retransmitting if you didn't get the appropriate reply. When the server grants a read lease, what should it do on retranmission? If a client gets a write revocation for a lease it does not hold, what should it do? Are there other interesting cases?

16. In LBFS, is the "size" returned by `GETHASH` redundant? What about the "count" taken by `CONDWRITE`? Give your reasoning why or why not.

17. LBFS makes a big deal out of saving bandwidth while providing traditional file system semantics. Give two ways you could trade weaker (reasonable) semantics for reduced bandwidth.

18. LBFS lazily checks that the SHA-1 hash in its database matches the actual data contents. At which steps in Figures 2 and 3 does this validation most likely occur and why?

19. When cleaning in LFS how do you tell if an inode is live? If a data block is live?

20. LFS grabs several segments and starts cleaning them. One of the segments contains a block that is at the end of a very large file (i.e., it is reached by indexing off of a triple indirect block). In the worst case, how many other segments will the cleaner have to touch in order to move this block to a new location? Besides extra disk accesses what problem can this cause?

21. For the workloads where LFS does worse than FFS: when is it plausible that the relative difference between the systems decreases on a RAID system?

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

23. Livelock: Why are infinitely large queues not a good idea (even assuming an infinite amount of memory)?

24. Because NFS operations are not idempotent, we want them to be executed exactly once. Jimbo says the way to do this is to send them over TCP just like in LBFS, since TCP provides a reliable, FIFO bytestream. Give an example and intuition for why using TCP does not completely solve the problem. (Hint: think about crashes.)

25. You just brought the Boneh-in-a-box coprocessor, which can do an infinitely fast SHA-1 hash computation. You change the metadata of FFS and LFS so that each pointer to a data block (rather than meta-data block) also stores a SHA-1 hash of the pointed-to data. Before reading a block, you will compare the SHA-1 hash to the blocks in your buffer cache and skip the disk read if it's already there.

    How would SHA-FFS performance compare to old-FFS on reads? How does SHA-FFS compare to SHA-LFS on writes?

26. You change the system in the previous questions to use Rabin-fingerprinting as in the LBFS paper. Why might this be a better idea on SHA-LFS than on SHA-FFS? What other changes do you have to make to the meta-data?

27. LFS: draw the ideal distribution of segments on a 20MB disk with 10MB of live data and state why you think your distribution is the best case. Make sure you label the x- and y-axis with the actual numbers you would see.

28. LBFS: client machine C1 writes to a file that an application on client machine C2 is using. The server sends C2 a message revoking its lease. What exactly does C2 do? (Note, make sure you also say what happens to the application using the file.)