DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033—Computer System Engineering Handout #33—May 3, 1994

QUIZ #3—Solutions.

Quiz 3 median score: 86

Problem 1. *(20 points)* You are examining a 3.5-inch magnetic disk that spins at 7500 RPM, with an average seek time of 6 milliseconds and a data transfer rate of 2 Megabytes/second.

a. How long does it take to read a block of 4 Kilobytes chosen at random?

Seek time:	6 ms.	
1/2 revolution rotation time:	4 ms.	7500/60 = 125 rps = 8 ms/revolution
Read time:	2 ms.	(4 Kilobytes)/(2 Megabytes/sec.)
Total time:	12 ms.	

Suppose that the operating system maintains a one-Megabyte cache in RAM that hold disk blocks. The access time to this cache is 125 nanoseconds and the data transfer rate is 20 Megabytes/second.

b. Give the formula that tells the average time to read a 4 Kilobyte disk block, assuming that the hit-ratio of the disk block cache is h.

The size of the cache is irrelevant and the access time to the first byte is negligible. Assuming that the system first reads a missing disk block into the cache, and then copies it into the user's read area,

$T_{av} = [(1-h)*12 + 0.2]$ milliseconds

If the system somehow reads the disk block in parallel into both the cache and the user's read area, the second term should be multiplied by h.

c. Ben Bitdiddle's workstation has 24 Megabytes of RAM. To increase h, Ben reconfigures the disk block cache to be much larger than one Megabyte. To his surprise he discovers that many of his applications now run slower rather than faster. What has Ben probably overlooked?

In order to increase the size of the block cache, Ben must have reduced the memory available for other things such as virtual memory, heap space, or application storage. These things are now working harder to utilize their smaller allocation. If the cache is really big, the time spent searching it may also now be significant.

d. Louis Reasoner has gone over to the Lab for Computer Science and has disassembled your disk unit to see how it works. Remembering that the centrifuge in the biology lab runs at 36,000 RPM, he has come up with a bright idea on how to reduce the rotational latency of the disk. He suggests speeding it up to 96,000 revolutions per minute. He calculates that the rotation time will now be 625 microseconds.

One of the other UROP's in the lab says this idea is crazy. Explain the line of reasoning that the other student probably has in mind.

This is fundamentally a problem about incommensurate scaling...

1. Don't visit the lab when Louis turns it on for the first time, because it will probably explode. The force required to keep the edges from flying away probably will exceed the tensile strength of the disk platter.
2. Assuming Louis switches to titanium platters to avoid the first problem, the edge of the disk is moving at Mach 1.6, while the innermost track is moving at about Mach 0.5. The seek arm may have some trouble moving across the region of the platter where it crosses the sound barrier.
3. The data will be passing under the read heads an order of magnitude faster than it used to, and they probably will require a complete redesign for the higher bandwidth.

Note also that since the change didn't do anything to improve the seek time, the overall performance may not improve nearly as much as Louis hopes.

Problem 2. (18 points)

Your company has asked you to take the responsibility for developing a storage system for a very demanding application, so you have decided to use a triplexed stable-storage scheme. You will use three ordinary disk drives D1, D2, and D3, and arrange that D2 and D3 store identical copies of each block stored on D1. The disk drives are of a very simple design that does not report read errors. That is, they just return data, whether or not it is valid.

a. You initially construct the application so that it writes a block of data to all three drives concurrently. After a crash occurs during the middle of a write, you are unable to reconstruct the correct data for that block. What is the problem?

All three copies contain a muddle of old data partially overwritten by an unknown amount of new data. There is no assured-to-be-correct source for any of the data.

b. Describe a simple modification to solve this problem.

There are several possibilities. Here is one simple one:

Write procedure: write D1, then write D2, then write D3.

Read procedure:
- Read the same block from D1 and D2 and D3
- If all three copies are different, copy D1 to D2 and D3 (completion)
- If the first two copies are the same, copy that value to D3 (completion)
- If the last two copies are the same, copy that value to D1 (backout)
- Return the now-unanimous value

c. One day there is a really awful power glitch that crashes all three disks in such a way that each disk corrupts one random track. Fortunately, the system wasn't writing any data at the time. Describe a procedure for reconstructing the data. Explain any cases that your procedure cannot handle.

For each disk block N
- read block N from D1 and D2 and D3
- If all three copies are different, report unfixable (intolerable error)
- If any two are the same, copy the agreed-on value over the odd one
- Return the now-unanimous value

The detected intolerable error occurs if we are unlucky enough that the same block N is corrupted on two or more different drives. Another (undetectable) intolerable error occurs if we are unlucky enough that the same block N is corrupted in exactly the same way on two or three different drives. It is interesting that the response to the "all three copies different" case is not the same as in the answer to part b; the reconstruction procedure of part c is designed to tolerate a different set of expected events.

3. *(20 points)* A system that implements atomic operations writes the following types of records to a log maintained on stable storage:

<AO_i ***start***>	Atomic operation *i* starts.
<AO_i, *x, old, new*>	Atomic operation *i* writes the value *new* over the value *old* for the variable named *x*.
<AO_i ***commit***>	Atomic operation i commits.
<AO_i ***abort***>	Atomic operation i aborts.
<AO_i, AO_j, … AO_k; ***checkpoint***>	At this instant, atomic operations i, j,…k are uncommitted.

Atomic operations start in numerical order. A crash occurs, and the recovery procedure scans the log backwards to the latest checkpoint, finding the following records:

<AO_7, AO_{41}, AO_{42}; ***checkpoint***>
<AO_{43} ***start***>
<AO_{44} ***start***>
<AO_{43}, y, 5, 6>
<AO_{43}, x, 5, 9>
<AO_{43}, ***commit***>
<AO_{44}, y, 6, 4>
<AO_{45} ***start***>
<AO_{45}, z, 3, 4>
<AO_7 ***abort***>
<AO_{41}, q, 1, 9>
<AO_{46} ***start***>
<AO_{45}, y, 4, 3>
<AO_{44} ***commit***>
<AO_{45}, y, 3, 7>
<AO_{41} ***commit***>
<AO_{47} ***start***>
<AO_{46} x, 9, 2>
<AO_{46} w, 0, 1>
<AO_{46} ***commit***>
<AO_{47}, u, 2, 1>
(***** crash happened here *****)

a. How much farther back in the log should the recovery procedure scan?
Assuming that the system is using a backout-list log protocol, to the start record of AO42, so that it can be undone. If it is using an intentions-list log (unlikely, because there is no mention of completion records) it would be necessary to go back to the start record of AO7 (which aborted after the checkpoint), to make sure that everything it has changed has been undone.

b. Which atomic operations mentioned in this part of the log are *winners* and which are *losers*?
winners: 7, 41, 44, 43, and 46 losers: 42, 45, and 47

c. Is it possible to make the list of losers complete with just the information given? <u>Yes</u>/No

d. What are the values of x and y immediately after the recovery procedure finishes? Why?
x = 2 Because: AO46 committed that value
y = 4 Because: AO44 committed that value. AO45's attempt to change it has been backed out.

4. *(5 points)* The log of problem 3 contains (ambiguous) evidence that someone didn't follow coordination rules. What is the evidence?

The only shared variables are X and Y, so they must be the clue. Reconstructing their version history up to the point where AO_{45} writes Y, we run across the following remarkable state:

AO	7	41	42	43	44	45	46
X			5->9				
Y			5->6	6->4	4->3		
commit state	A	U	U	C	U	U	U

AO_{45} is using a value for Y that AO_{44} has changed, but has not yet committed. If AO_{44} aborts, AO_{45} will also have to abort. The evidence is ambiguous because there might be an incredibly complicated, optimistic coordination rule that includes checking when AO45 tries to commit that AO44 didn't change Y again and that AO44 actually committed.

5. *(5 points)* The intentions-list logging protocol requires that the commit (or abort) record should be written to the log BEFORE any updates to cell storage are made. Identify the best reason for this requirement.

_____ 1. So that the recovery manager knows what to undo.
__X__ 2. So that the recovery manager knows what to redo.
_____ 3. Because the log is less likely to fail than the cell storage.
_____ 4. To minimize the number of disk seeks required.

6. *(12 points)* Brooks describes several interesting effects, three of which are listed below:

a. The mythical man-month.
b. The second-system effect.
c. Plan to throw one away.

Following are several root causes. Indicate beside each one the letter of the one effect it is most likely to cause.

__a__ 1. training time
__b__ 2. the first system is likely to be quite spartan
__a__ 3. additional communication
__c__ 4. a second system is likely to have better modularity
__a__ 5. repartitioning effort
__b__ 6. each idea looks good in isolation

Some of these causes can underly several effects; the "most likely" ambiguity can be resolved by noting that Brooks identifies certain causes as the principal ones.

7. Pat, Diane, and Quincy are having trouble using e-mail to schedule meetings of their 6.033 case study group. Pat suggests that they take inspiration from the 2-phase commit protocol.

(15 points) Which of the following most closely resembles 2-phase commit? (Circle the appropriate Roman numeral.)

I. a. Pat requests everyone's schedule openings.
   b. Everyone replies with a list but does not guarantee to hold all the times available.
   c. Pat inspects the lists and looks for an open time.
      If there is a time,
         Pat chooses a meeting time and sends it to everyone.
      Otherwise
         Pat sends a message canceling the meeting.

II. a–c, as in protocol I.
   d. Everyone, if they received the second message,
         acknowledges receipt.
      Otherwise
         sends a message to Pat asking what happened.

III. a–c, as in protocol I.
   d. Everyone, if their calendar is still open at the chosen time
         Send Pat an acknowledgement.
      Otherwise
         Send Pat apologies.
   e. Pat collects the acknowledgements. If all are positive
         Send a message to everyone saying the meeting is ON.
      Otherwise
         Send a message to everyone saying the meeting is OFF.
   f. Everyone, if they received the ON/OFF message,
         acknowledges receipt.
      Otherwise
         sends a message to Pat asking what happened.

IV. a–f, as in protocol III.
   g. Pat sends a message telling everyone that everyone has confirmed.
   h. Everyone acknowledges the confirmation.

(5 points) Below, circle the letter of the step that commits the meeting time.

a b c d e f g h

Step d tentatively commits the individual participants, but because some participants may send apologies rather than commitments the operation as a whole cannot commit until Pat knows that everyone has accepted. Step e is therefore required, and the sending of Pat's ON/OFF message is the commitment step for the operation.

Steps g and h are a refinement that allow Pat to discard her notes after the last acknowledgement comes back, but they contribute nothing to the 2-phase commit protocol itself.