Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033 Computer Systems Engineering: Spring 2001

Handout 29 - Quiz 3 solutions

All problems on this exam are multiple-choice questions. In order to receive credit you must fill in the blank(s) or mark the correct answer or answers for each question. The quiz was designed to be taken in 50 minutes, but you have 90 minutes to answer the questions.

Write your name on this cover sheet AND at the bottom of each page of this booklet.

Some questions may be much harder than others. Read them all through first and attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can't understand your answer, we can't give you credit!

THIS IS AN OPEN BOOK, OPEN NOTES QUIZ.

CIRCLE your recitation section number:

10:00	1. Garland/Freedman	13. Morris+Kaashoek/Tyan	11. Sollins/Furman
11:00	2. Garland/Furman	6. Morris+Kaashoek/Yang	12. Sollins/Tyan
12:00	5. Snoeren/Freedman		
1:00	10. Druschel/Silahtar	3. Jackson/Clarke	8. Saltzer/Dabek
	7. Snoeren/Yang		
2:00	14. Druschel/Dabek	4. Jackson/Silahtar	9. Saltzer/Clarke

Do not write in the boxes below

1-5 (xx/30)	6-13 (xx/38)	14-19 (xx/32)	Total (xx/100)

Name:

I Reading questions

1. [6 points]: According to Prof. Jackson, which of the following observations by Brooks (as stated in *The Mythical Man-Month*) are correct?

(Circle all that apply)

A. A good design has conceptual integrity.

B. A good designer plans to throw a complete implementation away.

C. Information hiding is bad, because all programmers should be able to see all code.

D. The waterfall model is a good design approach.

Brooks himself later admitted that "plan to throw one away" is not quite right; one should use an iterative approach instead of a waterfall approach, and correct mistakes as you go along instead. Information hiding is Parnas's term for data abstraction, now widely regarded (by Brooks as well) as a good idea.

2. [6 points]: During recovery, how does LFS (as described in reading #22) know that it has reached the end of the log?

(Circle best answer)

A. The end of the log is in the last segment (i.e., the one with the highest disk address).

B. The end of the log is in the last segment (i.e., the one with the highest disk address) that has a segment summary with a correct checksum.

C. The end of the log is in the segment that has a segment summary block with a correct checksum and a time-stamp more recent than the checkpoint time-stamp.

D. The end of the log is in the segment on the disk that has a segment summary block with a correct checksum and the most recent time-stamp.

The checksum is needed to ensure that the whole segment was written properly. Since records may be written out of order, the timestamp is necessary. Only the latest is required, since the checkpoint may be written after the last record.

3. [6 points]: The restart manager in System R (as described in reading #23) writes a new checkpoint at the end of a restart. Which transactions are listed in that checkpoint?

(Circle best answer)

A. No transactions will be listed.

B. The transactions that the restart manager has identified as losers.

C. The transactions that the restart manager has identified as winners.

D. The shadowed transactions .

The transactions listed in the checkpoint are those that are outstanding. At the end of a restart, all outstanding transactions have been resolved, so the list is empty.

4. [6 points]: How does Coda (as described in reading #24) save space in the replay log? (Include techniques that the paper says the designers are considering.)

(Circle all that apply)

A. It doesn't write file contents to the log.

B. It appends a single STORE record to the log for a modified file when the file is closed.

C. If a file is updated twice, any records related to the first update are removed from the log.

D. Records related to a temporary file (e.g., one created by a compiler for one of the compilation phases and removed at the end of the compilation) don't appear in the log.

All of the techniques listed are used (A,B, C), or are mentioned as possibilities (D). Of course, file contents must be saved, but they are saved elsewhere: the log contains a pointer to them.

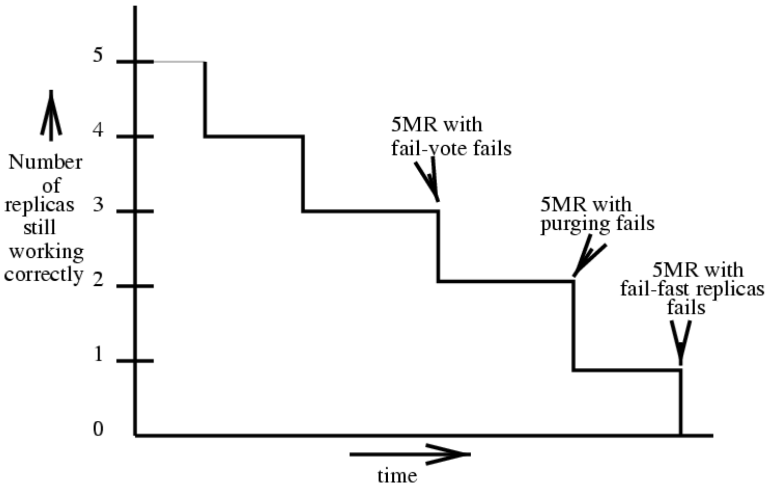

Figure 1: Failure points of three different 5MR supermodule designs, if repair does not happen in time.

5. [6 points]: Figure 1, a reproduction of Figure 7-9 from Chapter 7, shows failure points of three different 5MR supermodule designs. In the same style as Figure 1, plot in Figure 2 the failure points of the three corresponding designs for a triple module redundancy (TMR) supermodule, if repair doesn't happen.

(Complete figure below)

Triple modular redundancy involves 3 replicas. So long as one replica is running, fail-fast operation can continue, since each replica checks its own results. With two replicas running, fail-vote operation can continue, since a majority of the original three replicas is available. Purging allows fewer replicas to determine the outcome of a vote, by regarding a replica known to have failed as if it had never existed. But there must be at least two working replicas for a vote to take place, so purging will fail at the same point—when the second replica goes down.

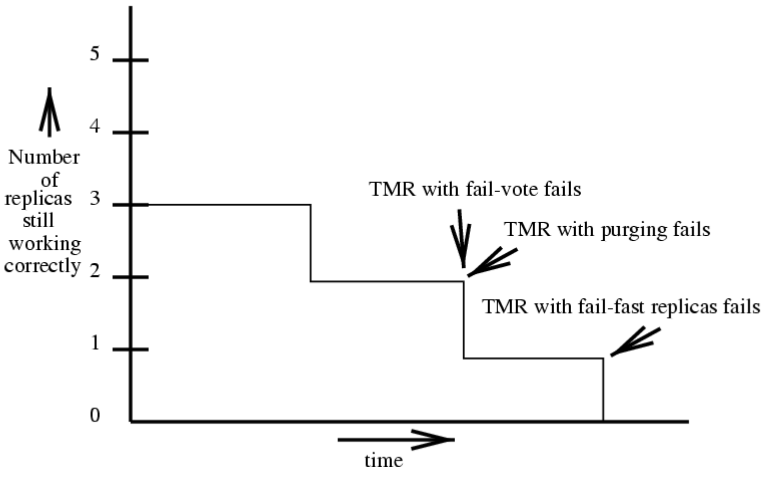

Figure 2: Failure points of three different TMR supermodule designs, if repair does not happen in time.

Name:

II Satellite Broadcasts

Carl Coder decides to quit his job at an e-commerce start-up and go to graduate school. He's curious about the possibility of broadcasting data files through satellites, and decides to build a prototype that does so.

Carl decides to start simple. He launches a satellite into a geosynchronous orbit, so that the satellite is visible from all points in the USA. The satellite listens for incoming bits on a radio up-channel, and re-sends each bit on a separate down-channel. Carl builds two ground stations, a sender and a receiver. The sending station sends on a radio to the satellite's up-channel; the receiving station listens to the satellite's down-channel.

Carl's test application is to send Associated Press (AP) news stories from a sending station, through the satellite, to a receiving station; the receiving station prints each story on a printer. AP stories always happen to consist of 1024 characters. Carl writes this code to run on computers at the sending and receiving stations (Scheme 1):

```
sender() {
  char buffer[1024];
  while(1){
    read next AP story into buffer;  // may wait for next story
    send_buffer(buffer);
  }
}

send_buffer(char buffer[1024]) {
  for(i = 0; i < 1024; i++)
    send_8_bits(buffer[i]);
}

receiver() {
  char buffer[1024];
  while(1){
    ok = recv_buffer(buffer);
    if(ok == true){
      print buffer on a printer;
    }
  }
}

recv_buffer(char buffer[1024]) {
  for(i = 0; i < 1024; i++)
    buffer[i] = recv_8_bits();
  return(true);
}
```

The receiving radio hardware receives a bit if and only if the sending radio sends a bit. This means the receiver receives the same number of bits that the sender sent. However, the receiving radio may receive a bit incorrectly, due to interference from sources near the receiver. The radio doesn't detect such errors; it just hands the incorrect bit to the computer at the receiving ground station with no warning. These incorrect bits are the only potential faults in the system, other than (perhaps) flaws in Carl's design.

If the computer tells the printer to print an unprintable character, the printer prints a question mark instead.

After running the system for a while, Carl observes that it doesn't always work correctly. He compares the stories that are sent by the sender with the stories printed at the receiver.

6. [1 points]: What kind of errors might Carl see at the receiver's printer?
(Circle all that apply)

A. Sometimes one or more characters in a printed story are incorrect.

B. Sometimes a story is repeated.

C. Sometimes stories are printed out of order.

D. Sometimes a story is entirely missing.

A. Any time a bit is corrupted by interference, the corresponding character will print as something different from that intended.

B. As described, there are no mechanisms in the communication system or in the programs that retransmit, so there is no way for a story to be repeated.

C. As described, the transmission system has constant delay, and there are no retransmissions, so there is no way that one story could be received out of order.

D. We are told that a bit is received for every bit sent, so the receiver will receive a (possibly incorrect) complete copy of every story.

7. [6 points]: The receiver radio manufacturer claims that the probability of receiving a bit incorrectly is one in a million, and that such errors are independent. If these claims are true, what fraction of stories are likely to be printed correctly?

(Circle best answer)

A. $1/1,000,000$

B. $1 - 1/1,000,000$

C. $(1/1,000,000)^{1024*8}$

D. $(1 - 1/1,000,000)^{1024*8}$

*For a story to be received correctly, every one of the 1024*8 bits must be received correctly. The chance of receiving a single bit correctly is (1 - 1/1,000,000). Since each bit is independent of all the others, the probability that all of the bits are received correctly is that expression raised to the 1024*8 power, which is answer D.*

Carl wants to make his system more reliable. He modifies his sender code to calculate the sum of the bytes in each story, and append the low 8 bits of that sum to the story. He modifies the receiver to check whether the low 8 bits of the sum of the received bytes match the received sum. Here is his new code (Scheme 2):

```
send_buffer(char buffer[1024]) {
  char sum = 0;
  for(i = 0; i < 1024; i++){
    send_8_bits(buffer[i]);
    sum = sum + buffer[i];
  }
  send_8_bits(sum);
}

recv_buffer(char buffer[1024]) {
  char sum1, sum2;
  sum1 = 0;
  for(i = 0; i < 1024; i++){
    buffer[i] = recv_8_bits();
    sum1 = sum1 + buffer[i];
  }
  sum2 = recv_8_bits();
  if (sum1 == sum2) return true;
  else return false;
}
```

8. [6 points]: What kind of errors might Carl see at the receiver's printer with this new system?
(Circle all that apply)

A. Sometimes one or more characters in a printed story are incorrect.

B. Sometimes a story is repeated.

C. Sometimes stories are printed out of order.

D. Sometimes a story is entirely missing.

A. Even though there is a checksum, if there are two transmission errors in the same story their effect on the checksum may cancel, so stories with errors will still slip through and be printed.

B. Carl hasn't added any retransmission mechanism, so we can't have duplicate stories.

C. Carl hasn't added any mechanism for delay, so stories can't arrive out of order.

D. The new system discards incoming stories when the checksum doesn't verify, so some stories can be completely missing.

9. [6 points]: Suppose the sender sends 1000 stories. Which scheme is likely to print a higher number of these 1000 stories correctly?
(Circle best answer)

A. The first scheme is likely to print more correctly.

B. The second scheme is likely to print more correctly.

C. None of the above.

This problem was intended to illustrate the same principle as the three-engine airplane example of chapter 7: when you add redundancy to a system, the additions can fail and thereby lead to new modes of system failure.

The first scheme prints every story, whether or not it is correct, so it is assured that it prints all of the correct stories. The second scheme doesn't print stories that fail to pass its error detection filter. As a result, it will occasionally throw away a perfectly correct story because of a failure in its error detection system–a reception error in a bit of just the checksum that the sender appended to the story. Conclusion: the first scheme would, on average, print more correct stories.

But there is one problem with this conclusion: if we calculate the probability that scheme 2 suppresses a correct story, we find that it is only 8 in a million. In a batch of just 1,000 stories, the chance that we would find an actual loss is negligible. Technically, the answer is A, but pragmatically, the answer is C.

Carl decides his new system is good enough to test on a larger scale, and sets up 3 new receive stations scattered around the USA, for a total of 4. All of the stations can hear the AP stories from his satellite. Users at each of the receivers call him up periodically with a list of articles that don't appear in their printer output so Carl can have the system *re-send* it. Users can recognize which stories don't appear, because the Associated Press includes a number in each story, and assigns numbers sequentially to successive stories.

10. [6 points]: Carl visits the sites after the system has been in operation for a week, and looks at the accumulated printouts (in the order they were printed) at each site. (Carl notes that the last story is received by all sites and all sites have received all retransmissions they have requested.) What kind of errors might he see in these printouts?

(Circle all that apply)

- **A.** Sometimes one or more characters in a printed story are incorrect.
- **B.** Sometimes a story is repeated.
- **C.** Sometimes stories are printed out of order.
- **D.** Sometimes a story is entirely missing.

A. As before, stories with errors can still slip through.

B. Carl has added a retransmission mechanism, and since this is a broadcast system, all the receivers hear the retransmissions. So if site A misses story 9 and requests a retransmission, site B will see story 9 arrive twice.

C. The only way that a receiver knows to request retransmission of a missed story is that it receives a story with a higher sequence number. Therefore, the retransmission is certain to arrive out of order.

D. We are told that the last story has been received by everyone, and that everyone has stopped asking for retransmissions, which means that all stories have been received. No stories are missing.

11. [1 points]: Suppose Carl sends out four AP stories. Site 1 detects an error in just the first story; site 2 detects an error in just the second story; site 3 detects an error in just the third story; and site 4 receives all 4 stories correctly. How many stories will Carl have to re-send? Assume any re-sent stories are received and printed correctly.

(Circle best answer)

- **A.** None.
- **B.** One.
- **C.** Two.
- **D.** Three.
- **E.** Four.

Exactly three stories have been requested, so Carl will resend exactly those three.

After reading the RAID paper, Carl realizes he could improve his system even more. He modifies his sender to send an extra "parity story" after every four AP stories; the parity story consists of the exclusive or of the four real stories. If one of the four stories is damaged, Carl's new receiver reconstructs it as the exclusive or of the parity and the other three stories. His new pseudo-code uses the checksumming send_buffer() and recv_buffer() to detect damaged stories.

```
sender() {
  char buffer[1024];
  char parity[1024];
  while(1){
    clear parity[] to all zeroes;
    for(i = 0; i < 4; i++){
      read next AP story into buffer;
      send_buffer(buffer);
      parity = parity xor buffer;  // XOR two buffers of 1024 elements
    }
    send_buffer(parity);
  }
}

receiver() {
  char buffers[5][1024];    // holds the 4 stories and the parity
  boolean ok[5];          // records which ones have been received correctly
  int n;          // count buffers received correctly

  while(1){
    n = 0;
    for(i = 0; i < 5; i++) {
      ok[i] = recv_buffer(buffers[i]);
      if (ok[i]) n = n + 1;
    }
    for(i = 0; i < 4; i++) {
      if (ok[i]) print buffers[i];   // buffers[i] is correct
      else if (n == 4) {  // reconstruct buffers[i]
        clear buffers[i] to all zeroes;
        for(j = 0; j < 5; j++) {
          if (i != j)
              buffers[i] = buffers[i] xor buffers[j];   // XOR two buffers
        }
        print buffers[i];
      } // don't print if you cannot reconstruct
    }
  }
}
```

12. [6 points]: Suppose Carl sends out four AP stories with his new system, followed by a parity story. Site 1 is just missing the first story; site 2 is just missing the second story; site 3 is just missing the third story; and site 4 receives all stories correctly. How many stories will Carl have to re-send? Assume any re-sent stories are received and printed correctly.

(Circle best answer)

$\boxed{\textbf{A.}}$ None.

B. One.

C. Two.

D. Three.

E. Four.

When only one story has a checksum error, the receiving site can reconstruct the damaged story by exor'ing the other three stories and the parity story. Since each site is missing at most one story, all the damaged stories can be reconstructed and no retransmissions are needed.

Carrie, Carl's younger sister, points out that Carl is using two forms of redundancy: the parity story and the checksum for each story. Carrie claims that Carl could do just as well with the parity alone, and that the checksum serves no useful function.

13. [6 points]: Is Carrie right?

(Circle best answer)

A. Yes, because the parity allows erroneous bits to be detected.

B. Yes, because the parity indicates which bits must be corrected.

C. No, because without the checksum, the system can't detect 100% of the incorrect bits.

$\boxed{\textbf{D.}}$ No, because the checksum indicates which story must be corrected.

With just the parity story, the receiver could discover that one of the five stories contained an error, but would have no idea which one, so it would not have enough information to correct the error. Carl's system has lost a useful feature, so Carrie (and answers A and B) must be wrong.

C. With or without the checksum, multibit errors can slip through, so either way we can't detect 100% of the incorrect bits; this answer can't be right.

D. Since each story carries its own checksum, a bad checksum indicates something is wrong with that particular story, so the receiver knows which one needs to be reconstructed. This is the only correct answer.

III Replicated Calendar

Ben Bitdiddle has just been promoted to Engineering Manager. He quickly notices two facts about his new job. First, keeping an accurate appointment calendar is crucial. Second, he no longer has any programming responsibilities. He decides to address both problems at once by building his own highly available replicated calendar system.

Ben runs a client user interface on his workstation. The client talks over the network to one of three replicated servers. Ben places the three servers, called S1, S2, and S3, in three different cities to try to ensure independent failure. Ben only runs one client at a time.

Each server keeps a simple database of Ben's appointments. The database holds a string for every hour of every day, describing the appointment for that hour. The string for each hour starts out empty. A server can perform just two operations on its own database:

- DBRead(day, hour) returns the appointments for a particular day and hour. Day indicates the desired day, where 0 means January 1st, 2000. Hour is an integer between 0 and 23, inclusive.
- DBWrite(day, hour, string) changes the string for a particular hour. Writing an empty string to an hour effectively deletes any existing appointment for that hour.

Each server allows Ben's client to invoke these operations by RPC. The RPC system uses a powerful checksum that detects all errors and discards any corrupted message. If the RPC client implementation doesn't receive a response from the server within a few seconds, it times out, sets the variable rpc_OK to false, and returns nil. If the client does receive a reply from the server, the RPC implementation sets rpc_OK to true and returns the result from the server, if any. The RPC system does not re-send requests. Thus, for example, if the network discards or corrupts the request or response message, the RPC call returns with rpc_OK set to false.

Ben's client user interface can display the appointments for a day and also change an appointment. To support these operations, Ben writes client software based on this pseudo-code (the notation S[i].f indicates an RPC call to function f on server i):

```
ClientRead(day, hour) {
  String s;
  for(i = 1; i <= 3; i++) { // try each server one by one
    s = S[i].DBRead(day, hour);
    if(rpc_OK)
      return s;                  // return with the first reply
  }
  return "Error";
}

ClientWrite(day, hour, what) {
  for(i = 1; i <= 3; i++) { // write to all three servers
    boolean done = false;
    while(done == false){
      S[i].DBWrite(day, hour, what);
      if(rpc_OK)
        done = true;
    }
  }
}
```

14. [6 points]: Suppose the network connecting Ben's client to servers S1 and S2 is fast and reliable, but the network between the client and S3 often stops working for a few minutes at a time. How will Ben's system behave in this situation?

(Circle all that apply)

A. ClientWrite will often take a few minutes or more to complete.

B. ClientRead will often take a few minutes or more to complete.

C. ClientWrite will often fail to update all of the servers.

D. ClientRead will often fail, returning "Error".

ClientRead is affected only by S1, which is fast and reliable, so choices B and D are incorrect. S3 stops working for a few minutes a time, since ClientWrite doesn't give up until it succeeds, this merely slows down ClientWrite; therefore, answer A is the only right answer.

Ben tests his system by reading and writing the entry for January 1st, 2000, 10am: he calls:

```
ClientWrite(0, 10, "Staff Meeting");
ClientWrite(0, 10, "Breakfast");
ClientRead(0, 10);
```

15. [2 points]: Suppose there are no faults. What string will the ClientRead call return? **(Circle best answer)**

A. Staff Meeting

B. Breakfast

C. 6.033 Recitation

D. Error

Since, the code executes strictly sequentially (ClientWrite doesn't return until it is done), answer B is the right answer.

Just to be sure, Ben tries a different test, involving moving a meeting from 10am to 11am, and scheduling breakfast:

```
ClientWrite(0, 10, "Free at 10");
ClientWrite(0, 11, "Free at 11");
ClientWrite(0, 10, "Talk to Frans at 10");
ClientWrite(0, 11, "Talk to Frans at 11");
ClientWrite(0, 10, "Breakfast at 10");
```

Ben starts the test, but trips over the power cord of his client computer while the test is running, causing the client to reboot. The client forgets that it was executing the test after the reboot; it doesn't re-start or continue the test. After the reboot Ben calls ClientRead(0, 10) and ClientRead(0, 11). Other than Ben's client crashing, the only faults that might occur during the test are lost messages (and thus RPC failures).

16. [6 points]: What results might Ben see?

(Circle all that apply)

A. Breakfast at 10, Talk to Frans at 11

B. Talk to Frans at 10, Talk to Frans at 11

C. Breakfast at 10, Free at 11

D. Free at 10, Talk to Frans at 11

Note that ClientWrite recovers from RPC failurs. Thus, if a ClientWrite fails, then S1, (S1 and S2), or (S1, S2, and S3) are updated by that write. Therefore, a possible outcome is choice A, since it could happen when the last ClientWrite updated S1 (and then failed). Another possible outcome is choice B, since it could happen after the third ClientWrote updated S1, and then failed. Choice C is not possible, since it implies that the last ClientWrite updated S1, which means that 4th ClientWrite succeeded (i.e., updating S1, S2, and S3); thus, "Free at 11" could not have returned by ClientRead(0, 11). A similar line of reasoning rules out choice D.

Ben is getting a little paranoid, so he calls ClientRead(0, 10) twice, to see how consistent his database is.

17. [6 points]: What results might Ben see?

(Circle all that apply)

A. Breakfast at 10, Breakfast at 10

B. Talk to Frans at 10, Talk to Frans at 10

C. Free at 10, Breakfast at 10

D. Talk to Frans at 10, Free at 10

A and B are obviously possible outcomes. D is also a possible outcome; Ben's test could have failed during the 3rd ClientWrite updating say S1 with "Talk to Frans at 10", leaving the other servers with "Free at 10". The first read might contact S1 and get "Talk to Frans at 10"; the second read could contact S2, if the first RPC to S1 fails. Choice C is not a possible outcome; if "Breakfast at 10" is possible, then the write with "Talk to Frans at 10" must have updated S1, S2, and S3; thus, "Free at 10" could not have been returned.

Ben feels this behavior is acceptable. Before he starts to use the system, however, his younger brother Mark points out that Ben's system won't be able to complete updates if one of the servers is down. Mark says that if a server is down, a DBWrite RPC to that server will time out, so ClientWrite will have higher availability if it ignores RPC timeouts. Mark suggests the following changed ClientWrite:

```
ClientWrite(day, hour, what) {
  for(i = 1; i <= 3; i++) {
    S[i].DBWrite(day, hour, what);
    // Ignore RPC failure.
  }
}
```

Ben adopts this change, and starts using the system to keep his appointments. However, his co-workers start to complain that he is missing meetings. Suspicious of Mark's change, Ben tests the system by manually clearing all database entries on all servers to empty strings, then executing the following code on his client:

```
ClientWrite(0, 10, "X");
v1 = ClientRead(0, 10);
ClientWrite(0, 10, "Y");
v2 = ClientRead(0, 10);
ClientWrite(0, 10, "Z");
v3 = ClientRead(0, 10);
```

Assume that the only possible faults are losses of RPC messages, and that RPC messages are delivered instantly.

18. [6 points]: With Mark's change, what sequences of v1, v2, and v3 are possible?
(Circle all that apply)

A. X, Y, Z

B. X, Z, Y

C. X, X, Z

D. X, Y, X

With this change, a ClientWrite might return after having updated 0, 1, 2, or all three of servers. Thus, choices A, C, and D are possible outcomes. Choice B is not possible, because returning "Z" before writing "Z" is impossible.

19. [6 points]: In Ben's original system, what sequences of v1, v2, and v3 would have been possible? **(Circle all that apply)**

A. X, Y, Z

B. X, Z, Y

C. X, X, Z

D. X, Y, X

Only choice A could have been possible, because a successful write in Ben's original system would have updated all copies.

End of Quiz 3 solutions