Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033 Computer Systems Engineering: Spring 2002

Handout 41 - Quiz 3

This quiz has 13 multiple-choice questions. In order to receive credit you must mark the correct answer or answers for each question. You have 60 minutes to complete this quiz.

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

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. **Please be neat. Circle your choices clearly and unambiguously!** If we can't understand your answer, we can't give you credit!

There's a copy of the pseudocode for Section II on the last page (page 11). You may tear that page out for easy reference. You don't need to turn that page in with your exam.

THIS IS AN OPEN BOOK, OPEN NOTES QUIZ.

CIRCLE your recitation section:

10:00	1. Balakrishnan/Chambers	13. Morris+Kaashoek/Gnawali	11. Amarasinghe/Bhattacharyya
11:00	2. Balakrishnan/Salz	6. Morris+Kaashoek/Chambers	12. Amarasinghe/Gnawali
12:00	5. Ernst/Salz	14. Witchel/Bauer	
1:00	8. Ernst/Yip	3. Leiserson/Freedman	10. Teller/Bauer
	7. Saltzer/Vandiver		
2:00	9. Saltzer/Freedman	4. Leiserson/Vandiver	15. Teller/Bhattacharyya

Do not write in the boxes below

1-2 (xx/16)	3-5 (xx/24)	6 (xx/4)	7-10 (xx/32)	11-13 (xx/24)	Total (xx/100)

Name:

I Appetizers

1. [8 points]: A system administrator in Tech Square notices that a file server's disk is failing for two unrelated reasons. Once every 30 days, on average, vibration due to nearby construction breaks the disk's arm. Once every 60 days, on average, a power surge destroys the disk's electronics. The system administrator fixes the disk instantly each time it fails. The two failure modes are independent of each other, and independent of the age of the disk.

What's the mean time to failure of the disk?

(Circle the BEST answer)

A. 45 days.

B. 30 days.

C. 22.5 days.

D. 20 days.

2. [8 points]: The checkpoint operation in System R (Reading #21) invokes FILESAVE on files. Which of the following statements about System R's checkpoint and recovery mechanisms are true? Assume that neither the log nor the data on the database disk are lost during a crash.

(Circle ALL that apply)

A. System R requires losers to be undone while recovering from a crash *because* it checkpoints at times that are *not* transaction-consistent.

B. System R requires winners to be redone while recovering from a crash *because* it checkpoints at times that are *not* transaction-consistent.

C. If all System R checkpoints happened at transaction-consistent times, then the recovery process would not have to scan the log further back than the time of the last checkpoint.

D. The System R checkpoint mechanism does not allow running transactions to straddle more than one checkpoint operation, in order to limit the size of the log.

3. [8 points]: Consider the choice of segment size in a log-structured file system (LFS; Reading #18). Suppose that a disk has a latency of 10 milliseconds for seek and rotation combined, and a data transfer rate of 20 Megabytes per second for any non-zero-length disk write. Assume that each segment write requires only one disk seek operation. What's the smallest segment size that would always allow a complete segment to be written at 90% of the disk data transfer rate?

(Circle the BEST answer)

A. 1 Megabyte.

B. 1.8 Megabytes.

C. 3.6 Megabytes.

D. 18 Megabytes.

4. [8 points]: The Coda file system (Reading #22) allows disconnected clients to modify files and reintegrate their changes upon reconnection. Which of these statements are true of the system described in the paper?

(Circle ALL that apply)

A. Upon reintegration, a Coda server detects conflicts by checking the client log and comparing old and new values in the log, using the file storeid as a transaction identifier.

B. If two disconnected clients operate on different parts of a non-directory file and later reconnect to the server, the Coda server automatically merges the two clients' changes.

C. During reintegration, if any non-directory file has a write/write conflict, Coda (as implemented in the paper) aborts the reintegration transaction on the entire volume.

D. Suppose two clients A and B get disconnected at the same time, caching file F. A writes file F, closes F, and successfully reintegrates all its changes with the Coda server upon reconnection. Meanwhile, client B performs several reads on file F. When B reconnects later and reintegrates, the Coda server will inform B of a conflict, warning that it had read a potentially old version of file F while disconnected.

5. [8 points]: According to Prof. Saltzer's lecture, which of these statements about complex systems are true?

(Circle ALL that apply)

A. Complex systems usually fail for complex reasons.

B. Systems that work often do so for reasons not thought of by the designer.

C. The high rate of change of technology makes it easier to design computer systems because one does not need to worry about performance.

D. Brooks argues that the rationalism approach to system design leads to better systems than the empiricism approach because it is more systematic.

E. Systems that have theme songs are more likely to succeed.

Name:

II ANTS: Advanced "Nonce-ensical" Transaction System

Sara Bellum, forever searching for elegance, sets out to design a new transaction system called ANTS based on the idea of nonces from 6.033. She observes that the locking schemes she learned in 6.033 cause transactions to wait for locks held by other transactions. She observes that it is possible for a transaction to simply abort and retry, instead of waiting for a lock. A little bit more work convinces her that this idea may allow her to design a system in which transactions don't need to use locks for isolation.

Sara sets out to write pseudocode for the following functions: BEGIN(), READ(), WRITE(), COMMIT(), ABORT(), and RECOVER(). She intends that, together, these functions will provide transaction semantics: isolation, recoverability, and durability. *You may assume that once any of these functions starts, it runs to completion without preemption or failure, and that no other thread is running any of the functions at the same time.* The system may interleave the execution of multiple transactions, however.

Sara's implementation assigns a transaction identifier (TID) to a transaction when it calls BEGIN(). The TIDs are integers, and ANTS assigns them in numerically increasing order.

Sara's plan for the transaction system's storage is to maintain cell storage for variables, and a write-ahead log for recovery. Sara implements both the cell storage and the log using stable storage. The log contains the following types of records:

- BEGIN TID
- COMMIT TID
- ABORT TID
- WRITE TID, Variable Name, Old Value

Sara implements BEGIN(), COMMIT(), ABORT(), and RECOVER() as follows:

- BEGIN() allocates the next TID, appends a BEGIN TID record to the log, and then returns the TID.
- COMMIT(TID) appends a COMMIT TID record to the log before returning.
- ABORT(TID) undoes all of transaction TID's WRITE() operations by scanning the log backwards and writing the old values from the transaction's WRITE records back to the cell storage. After completing the undo, ABORT() appends an ABORT TID entry to the log, and returns.
- RECOVER() is called after a crash and restart, before starting any more transactions. It scans the log backwards, undoing each WRITE record of each transaction that had neither committed nor aborted at the time of the crash. RECOVER() appends an ABORT record for each such transaction.

Sara's **isolation intention** *is that the result of executing multiple transactions in parallel is the same as executing those same transactions one at a time, in increasing transaction ID order.*

Name:

Sara wants her READ() and WRITE() implementations to provide isolation by adhering to the following rule. Suppose a transaction with TID t executes READ() on the variable X. Let u be the highest transaction ID $< t$ that calls WRITE() on X and commits. The READ() executed by t should return the value that u writes.

Sara observes that this rule does not require her system to execute transactions in strict TID order; for example, the fact that two transactions call READ() on the same variable does not (by itself) constrain the order in which the transactions must execute.

To see how Sara intends ANTS to work, consider the following two transactions:

	TRANSACTION T_A	TRANSACTION T_B
1	$t_a \leftarrow$ BEGIN() *(returns 15)*	
2		$t_b \leftarrow$ BEGIN() *(returns 16)*
3	$v_a \leftarrow$ READ(t_a, X);	
4	$v_a \leftarrow v_a + 1$;	
5		$v_b \leftarrow$ READ(t_b, X);
6		$v_b \leftarrow v_b + 1$;
α	WRITE(t_a, X, v_a);	WRITE(t_b, X, v_b);
β	COMMIT(t_a);	COMMIT(t_b);

Each transaction marks its start with a call to BEGIN(), then reads the variable X from the cell store and stores it in a local variable, then adds one to that local variable, then writes the local variable to X in the cell store, and then commits. Each transaction passes its transaction ID (t_a and t_b respectively) to the READ(), WRITE(), and COMMIT() functions.

These transactions both read and write the same piece of data, X. Suppose that T_A starts just before T_B, and Sara's BEGIN() allocates TIDs 15 and 16 to T_A and T_B, respectively. Suppose that ANTS interleaves the execution of the transactions as shown through line 6, but that ANTS has not yet executed lines α and β. You can assume that no other transactions are executing, and that no failures occur.

6. [4 points]: In this situation, which of the following actions can ANTS take in order to ensure isolation?

(Circle ALL that apply)

A. Force just T_A to abort, and let T_B proceed.

B. Force just T_B to abort, and let T_A proceed.

C. Force neither T_A nor T_B to abort, and let both proceed.

D. Force both T_A and T_B to abort.

Name:

To help enforce the isolation intention, Sara's implementation of ANTS maintains the following two pieces of information for each variable:

- ReadID — the TID of the highest-numbered transaction that has successfully read this variable using READ().
- WriteID — the TID of the highest-numbered transaction that has successfully written this variable using WRITE().

Sara defines the following utility functions in her implementation of ANTS:

- INPROGRESS(TID) returns false if TID has committed or aborted, and otherwise true. (All transactions interrupted by a crash are aborted by the RECOVER function.)
- exit() terminates the current thread immediately.
- LOG() appends a record to the log and waits for the write to the log to complete.
- read_data(x) reads cell storage and returns the corresponding value.
- write_data(x, v) writes value v into cell storage x.

Name:

Sara now sets out to write pseudocode for READ() and WRITE(). (The last page of this quiz has a copy you can tear out for easy reference.)

Lines are numbered 1–13 in READ().

```
1 FUNCTION READ(TID tid, Cell x)
2    // Return the value stored in cell x
3    if (tid < WriteID(x)) {
4        ABORT(tid); exit();
5    }
6    if ( (tid > WriteID(x)) AND INPROGRESS(WriteID(x)) ) {
7        // Last transaction to have written x is still in progress
8        ABORT(tid); exit();
9    }
10   // In all other cases execute the read
11   v ← read_data(x);
12   ReadID(x) ← max(tid, ReadID(x)); // Update ReadID(x)
13   return v;
```

Lines are numbered A–N in WRITE().

```
A FUNCTION WRITE(TID tid, Cell x, DataValue v)
B    // Store value v in cell storage x
C    if (tid < ReadID(x)) {
D        ABORT(tid); exit();
E    } else if (tid < WriteID(x)) {
F        Statement (I); // See Question 8
G    } else if ( (tid > WriteID(x)) AND INPROGRESS(WriteID(x)) ) {
H        ABORT(tid); exit();
I    }
J    LOG(WRITE, tid, x, read_data(x));
K    write_data(x, v);
L    // Now update WriteID(x)
M    Statement (II); // See Question 10
N    return;
```

Name:

Help Sara complete the design above by answering the following questions.

7. [8 points]: Consider lines 6–9 of the READ pseudocode. Sara is not sure if these lines are necessary. If lines 6–9 are removed, will the implementation preserve Sara's isolation intention?

(Circle ALL that apply)

- **A.** Yes, the lines can be removed. Because the previous WRITE to x (by the transaction with TID WriteID(x)) cannot be affected by transaction tid, read_data(x) can execute.
- **B.** Yes, the lines can be removed. Suppose transaction T_1 successfully executes WRITE x, and then transaction T_2 executes READ x before T_1 commits. After this, T_1 cannot execute WRITE x successfully, so T_2 would have correctly read the last written value of x from T_1.
- **C.** No, the lines cannot be removed. One reason is: The only transaction that can correctly execute read_data(x) is the transaction with TID equal to WriteID(x). Therefore, the condition on line 6 of READ should simply read: "if (tid $>$ WriteID(x))".
- **D.** No, the lines cannot be removed. One reason is: isolation might not be preserved when transactions abort.

8. [8 points]: Consider **Statement (I)** on line F in the pseudocode for WRITE. Which of the following operations for this statement preserve Sara's isolation intention?

(Circle ALL that apply)

- **A.** ABORT(tid); exit();
- **B.** return; (*without aborting* tid)
- **C.** Find the higher-numbered transaction T' corresponding to WriteID(x); ABORT(T') and terminate thread T'; execute write_data(x, v) in transaction tid; and return.
- **D.** All of the above choices.

Name:

9. [8 points]: Consider lines G–H in the pseudocode for WRITE. Sara is not sure if these lines are necessary. If lines G–H are removed, will Sara's implementation preserve her isolation intention? Why or why not?

(Circle ALL that apply)

A. Yes, the lines can be removed. We can always recover the correct values from the log.

B. Yes, the lines can be removed since this is the WRITE() call; it's only on a READ() call that we need to be worried about the partial results from a previous transaction being visible to another running transaction.

C. No, the lines cannot be removed. One reason is: If transaction T_1 writes to cell x and then transaction T_2 writes to cell x, then an abort of T_2 followed by an abort of T_1 may leave x in an incorrect state.

D. No, the lines cannot be removed. One reason is: If transaction T_1 writes to cell x and then transaction T_2 writes to cell x, then an abort of T_1 followed by an abort of T_2 may leave x in an incorrect state.

10. [8 points]: Which of these operations for **Statement (II)** on line M preserves Sara's isolation intention?

(Circle ALL that apply)

A. WriteID(x) $\leftarrow$ tid

B. WriteID(x) $\leftarrow \min$(WriteID(x), tid)

C. WriteID(x) $\leftarrow \max$(WriteID(x), tid)

D. WriteID(x) $\leftarrow \max$(WriteID(x), ReadID(x))

Ben Bitdiddle looks at the READ and WRITE pseudocode shown before for Sara's system and concludes that her system is in fact nonsensical! To make his case, he constructs the following concurrent transactions:

TRANSACTION T_1	TRANSACTION T_2
1 ID1 $\leftarrow$ BEGIN();	
2	ID2 $\leftarrow$ BEGIN();
3 WRITE(ID1, A, v_1);	
4	$v_2 \leftarrow$ READ(ID2, A);
5	WRITE(ID2, B, v_2);
6	COMMIT(ID2);
7 $v_1 \leftarrow$ READ(ID1, B);	
8 COMMIT(ID1);	

The two transactions are interleaved in the order shown above. Note that T_1 begins before T_2. Ben argues that this leads to a deadlock.

Name:

11. [8 points]: Why is Ben's argument incorrect?

(Circle ALL that apply)

- **A.** Both transactions will abort, but they can both retry if they like.
- **B.** Only T_2 will abort on line 4. So T_1 can proceed.
- **C.** Only T_1 will abort on line 7. So T_2 can proceed.
- **D.** Sara's system does not suffer from deadlocks, though concurrent transactions may repeatedly abort and never commit.

Recall that Sara uses a write-ahead log for crash recovery. The log format is described on page 4.

12. [8 points]: Which of these statements is true about log entries in Sara's ANTS implementation?

(Circle ALL that apply)

- **A.** The order of BEGIN entries in the log is in increasing TID order.
- **B.** The order of COMMIT entries in the log is in increasing TID order.
- **C.** The order of ABORT entries in the log is in increasing TID order.
- **D.** The order of WRITE entries in the log for any given variable is in increasing TID order.

13. [8 points]: The WRITE function appends the WRITE record to the log before it writes the cell storage. Sara wants to improve performance by caching the stable cell storage in main memory (which is not stable). She changes read_data() to read the value from the cache if it is there; if it isn't, read_data() reads from stable cell storage. She changes write_data() to update just the cache; ANTS will update stable cell storage later. Can ANTS delay the write to the stable cell storage until after the COMMIT record has been written to the log, and still ensure transaction semantics?

(Circle ALL that apply)

- **A.** No, because if the system crashed between the commit and the write to stable storage, RECOVER() would not recover cell storage correctly.
- **B.** Yes, because the log contains enough information to undo uncommitted transactions after a crash.
- **C.** Yes, because line 3 of READ() won't let another transaction read the data until after the write to stable storage completes.
- **D.** None of the above.

End of Quiz 3
Have a great summer!

Name:

For quick reference. You may tear this page out.

Lines are numbered 1–13 in READ().

```
1 FUNCTION READ(TID tid, Cell x)
2    // Return the value stored in cell x
3    if (tid < WriteID(x)) {
4        ABORT(tid); exit();
5    }
6    if ( (tid > WriteID(x)) AND INPROGRESS(WriteID(x)) ) {
7        // Last transaction to have written x is still in progress
8        ABORT(tid); exit();
9    }
10   // In all other cases execute the read
11   v ← read_data(x);
12   ReadID(x) ← max(tid, ReadID(x)); // Update ReadID(x)
13   return v;
```

Lines are numbered A–N in WRITE().

```
A FUNCTION WRITE(TID tid, Cell x, DataValue v)
B    // Store value v in cell storage x
C    if (tid < ReadID(x)) {
D        ABORT(tid); exit();
E    } else if (tid < WriteID(x)) {
F        Statement (I); // See Question 8
G    } else if ( (tid > WriteID(x)) AND INPROGRESS(WriteID(x)) ) {
H        ABORT(tid); exit();
I    }
J    LOG(WRITE, tid, x, read_data(x));
K    write_data(x, v);
L    // Now update WriteID(x)
M    Statement (II); // See Question 10
N    return;
```

Name: