DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033—Computer System Engineering Handout #35—May 3, 1999

Practice Quiz #3

This handout contains five years worth of quizzes. Have fun. The solutions will be handed out separately. As before, one question will be included on this year's quiz.

 Name ____________________

1994 Quiz #3

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?

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.

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?

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

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?

b. Describe a simple modification to solve this problem.

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.

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?

b. Which atomic operations mentioned in this part of the log are *winners* and which are *losers*?

winners: losers:

c. Is it possible to make the list of losers complete with just the information given?

Yes/No

d. What are the values of x and y immediately after the recovery procedure finishes? Why?

x = Because:
y = Because:

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

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.
_____ 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.

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

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

Name ________________________

1995 Practice Quiz #3

1. *(20 points)* You are designing a computer system to control the power grid for the Northeastern US. If your system goes down, the lights go out and civil disorder —riots, looting, fires, etc.—will ensue. Thus, you have set a goal of having a system MTTF of at least 100 years (about 10^6 hours).

2. For hardware you are constrained to use a building block computer that has a MTTF of 1000 hours and a MTTR of 1 hour. Assuming that the building block is self checking (i.e., fail stop), how can you arrange to meet your goal? (Briefly explain your answer.)

3. Your software is 10^6 lines of code. Even after rigorous testing, the software can be expected to have about one bug per 10^3 lines or about 10^3 bugs. You may assume that these bugs are unrelated. Discuss briefly a strategy for how to arrange your software to meet the goal of a MTTF of at least 100 years.

4. *(30 points)* The registrar's office is upgrading its scheduling program for limited-enrollment humanities subjects to make it a network service that anyone can use by RPC. But there is a concern that having multiple threads trying to update the database at the same time could possibly cause trouble.

   The program operates as follows; the interface originally had just two entries:

   Status = Register(subject-name)

   and

   Drop(subject-name)

   where subject-name is a string such as "21W471". The Register procedure checks to see if there is any space left in the subject, and if there is, it increments the class size by one and returns the status value zero. If there is no space, it does not change the class size; instead it returns the status value -1. (This is a very primitive registration system—it just keeps counts!)

   As part of the upgrade, two new procedures have been added:

   Seize(subject-name)

   and

   Release(subject-name)

   Seize sets a lock on the given subject, waiting if necessary for the lock to be released. Release releases the lock on the given subject.

   Here is a typical application program, which registers the caller for two subjects, H1 and H2:

```
integer procedure Register-two(H1, H2)
    status = Register(H1)
    if (status== 0) then
        status = Register(H2)
        if (status == -1)
            Drop(H1)
        endif
    endif
    return status
```

 Name ____________________

4a. The goal is that the entire procedure Register-two should be an atomic transaction. Add calls for Seize and Release to the Register-two procedure that obey the Simple Locking Rule (i.e., Seizes before the first read/write, and Releases after the last read/write).

```
integer procedure Register-two(H1, H2)
        status = Register(H1)
        if (status== 0) then
                status = Register(H2)
                if (status == -1)
                        Drop(H1)
                endif
        endif
        return status
```

4b. Add calls for Seize and Release that obey the Two-phase Locking Rule, and in addition postpone all Seizes as late as possible and do all Releases as early as possible.

```
integer procedure Register-two(H1, H2)
        status = Register(H1)
        if (status== 0) then
                status = Register(H2)
                if (status == -1)
                        Drop(H1)
                endif
        endif
        return status
```

4c. Louis Reasoner has come up with a suggestion that he thinks could simplify the job of programmers creating atomic application programs such as Register-two. His idea is to revise the two programs Register and Drop by moving the Seize and Release statements into them. That is, the procedure:

```
integer procedure Register(subject-name)
        { current code }
        return status
```

would become:

```
integer procedure Register(subject-name)
        Seize(subject-name)
        { current code }
        Release(subject-name)
        return status
```

As usual, Louis has misunderstood part of the problem. Give a brief argument of what is wrong with this idea.

5. *(20 points)* Consider a database system in which all transactions are atomic with respect to system failures and other, concurrent, transactions. Locks on data records along with a log in nonvolatile memory are used to provide these properties. The database itself is stored on disk and cached in main memory by the cache manager. The database system is able to write the following records to the log:

- BEGIN N: start of transaction number N
- COMMIT N: commit of transaction N
- ABORT N: abort of transaction N
- UNDO N: a record to be able to undo part of transaction N
- REDO N: a record to be able to redo part of transaction N

In addition, the system writes DATA records to the on-disk database.

5a. Suppose the cache manager implements a write-through policy. That is, whenever a transaction writes a record to disk to update the database, the record is synchronously forced to disk. Explain what minimum records need to be written to the database and the log, and in what order for the following database operations: start transaction, abort transaction, commit transaction, read of variable, and write of a variable?

Operation	**Write the following record(s) to the log**	**Write the following record(s) to the database**	**Write log or database first?**
Start transaction			
Abort transaction			
Commit transaction			
Read variable			
Write variable			

5b. In the system from part (a), how is the state of the system recovered from the log after a system failure (i.e., contents of main memory are lost)?

6. *(30 points)* Ben Bitdiddle has proposed the simple neat and robust file system (SNARFS). Ben's system has no on-disk data structures other than the disk blocks themselves. Each 4K-byte disk block starts with the following 24 bytes of information:

- FID (File-ID): a 64-bit number that uniquely defines a file. A FID of zero implies that the disk block is free.
- SN (Sequence Number): a 64-bit number that identifies which block of a file this disk block contains.
- T (Time): The time of last update.

In addition, the first block of a file contains the file name (string), version number, and the FID of its parent directory. The rest of the first block is filled with data. Setting the directory FID to zero marks the entire file free.

Directories are just files. Each directory need contain only the FID of its parent directory. However as a 'hint' directories may also include a table giving the mapping from name to FID and the mapping from FID to blocks for some of the files in the directory.

To allow fast access, three in-memory (virtual memory) structures are created each time the system is booted:

- MAP: an in-memory hash-table that associates a (FID,SN) pair with the disk block containing that block of that file.
- FREE: a "buddy-system" free list that represents all of the free blocks on disk in a compact manner. (A buddy-system maintains a list of contiguous regions of free disk space.)
- RECYCLE: a list of blocks that are available for reuse but have not yet been written with an FID of 0.

6a. What is the minimum number of disk I/Os required in Ben's SNARFS system to create a file in an existing directory, and write a 2K-byte string into the file? If the system crashes (i.e., the contents of virtual memory are lost) after these I/Os are completed, the file should be present in the appropriate directory after recovery. Each read or write of a disk block results in one disk I/O. (Briefly explain your answer.)

6b. Ben argues that the in-memory structures can be rebuilt after a crash. Explain briefly what actions are required to rebuild MAP, FREE, and RECYCLE at boot time.

6c. Ben also argues that with appropriate free list management, the write performance of SNARFS can be as fast as for a log-structured file system. How can this be?

Practice Quiz #3 1996

1. *(20 points)* Here is a list of 5 different storage systems.

   (1) RSS, the System R storage system
   (2) A log-structured file system (LFS)
   (3) xFS
   (4) AFS (Athena's file system)
   (5) The Coda file system

   For each situation below, choose the storage system that would be best suited for that situation, and briefly explain your answer. For each situation, you should consider the workload placed on the storage system; consider any special requirements for scalability, security, or reliability; and maximize speed. You will use each storage system only once.

1a. *(4 points)* A file server with a write-dominated workload

1b. *(4 points)* Transaction processing

1c. *(4 points)* A file server for a large network where clients are often disconnected

1d. *(4 points)* A very fast file server for a small, fast network with trusted clients

2. *(4 points)* A file server for a large, untrusted, and relatively slow network*(18 points)* Match each "hint for system design" (from Lampson's paper) or effect (from *The Mythical Man-Month*) with a system or argument that exhibits that effect, or uses that hint.

_____	Second system effect	(1) The Java bytecode verifier
_____	Leave it to the client	(2) The end-to-end argument
_____	Keep it simple	(3) Brinch Hansen's RC 4000 (a "microkernel")
_____	Log updates	(4) 6.033 design project 1
_____	Static analysis	(5) Multics
_____	Cache answers	(6) System R

3. *(28 points)* Alyssa P. Hacker needs a transaction-processing system for remotely updating information about her collection of 97 cockroaches. Having taken 6.033, she remembers System R. However, she thinks that System R was much too complex, and she can do better.

3a. *(4 points)* First, Alyssa stores her database on disk. When a transaction commits, it simply goes to the disk and writes its changes in place over the old data.

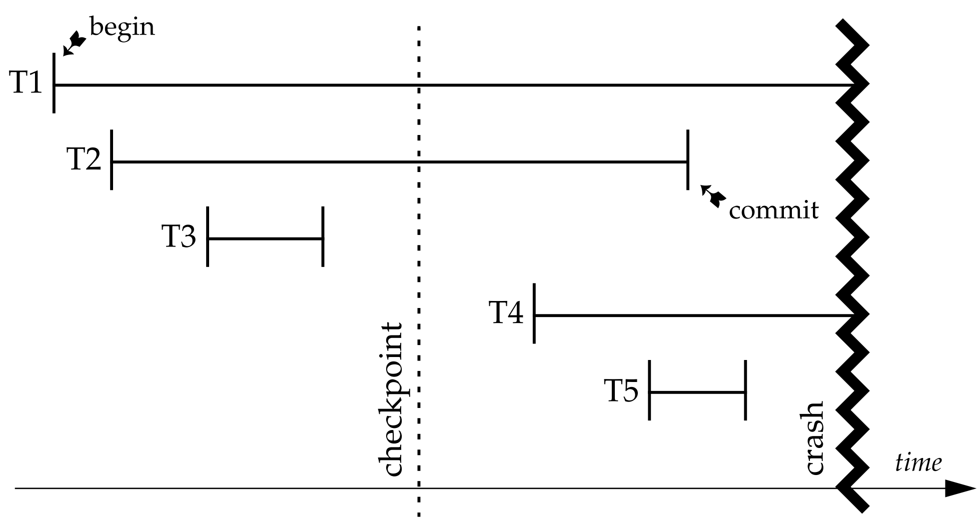
Give one major problem with Alyssa's system.

3b. *(4 points)* In Alyssa's second design, the *only* structure she keeps on disk is a log. This log records every change made to the database, along with the transaction which the change was a part of. Commit records, also stored in the log, indicate when a transaction commits. When the system crashes and recovers, it replays the log, redoing each committed transaction.

Give one disadvantage of Alyssa's system.

Now, Alyssa decides to change her system by storing both the log, and a snapshot of her database. She remembers that System R called this snapshot mechanism a *checkpoint*. To checkpoint, Alyssa just writes the entire state of the database into the log. When she crashes and recovers, she starts from the last checkpointed state, and then redoes or undoes some transactions to restore her database.

The other parts of this question refer to these five transactions:



So T2, T3, and T5 commit before the crash, but T1 and T4 have not committed by that time.

 Name ________________

3c. *(10 points)* When the system recovers, after the checkpointed state is loaded, some transactions will need to be undone or redone using the log. For each transaction, mark off in the table whether that transaction needs to be undone, redone, or neither.

	Undone	*Redone*	*Neither*
T1			
T2			
T3			
T4			
T5			

3d. *(10 points)* Now, assume that transactions T2 and T3 were actually *nested* transactions: T2 was nested in T1, and T3 was nested in T2. Again, fill in the table:

	Undone	*Redone*	*Neither*
T1			
T2			
T3			
T4			
T5			

4. *(34 points)* ODC (an acronym for Odd Disk Company) has just invented a new kind of stable storage, the Whisk. A Whisk is unlike a disk in the following ways:

- Whisks have very low read and write latencies. (Disk latency is high for both operations.)
- On the other hand, a Whisk's bandwidth is much less than that of a disk.
- Whisks are *associative.* A block is named with a pair of items: an address and a *tag*. (Disks just use addresses.) We write these pairs as A/t, where A is the address and t is the tag. Thus, for example, there might be three blocks on the Whisk with address 49, if those three blocks had different tags (e.g., 49/1, 49/2, and 49/97).

The Whisk provides four important operations:

- *read*(A/t) $\rightarrow$ *data*. This is the normal read operation.
- *write*(A/t, *data*). Just like a normal disk, if the system crashes during a *write*, a partially written block may result.
- *exists*(A/t) $\rightarrow$ *boolean*. This returns true if block A/t exists on the Whisk.
- *change-tag*(A/m, n). This operation is new: *change-tag*(A/m, n) **atomically** changes the tag m of block A/m to n (deleting any previous block A/n in the process). If the system crashes during a *change-tag*, the disk will not be left in a partially-changed state.

Ben Bitdiddle is very excited about the properties of Whisks! Help him develop different storage systems using Whisks as the medium.

4a. *(7 points)* At first, Ben emulates a normal disk by writing all blocks with tag 0. He wants to add an atomic write operation to this system. Give the operations required to atomically write a block on Ben's Whisk, and circle the operation that is the commit point. (In class, we gave an atomic-write algorithm for disks which required four operations; you should be able to do it with less.)

4b. *(5 points)* Now, Ben is considering implementing a log-structured file system with the Whisk. Is this a good idea? Consider the properties of the disk LFS was designed for.

Now, Ben is working on a Whisk transaction system; he'd like you to help him finish it. Looking through his notes, you see that Ben's system *will use no caches or logs*: all writes go straight to the Whisk. One sentence particularly catches your eye: a joyfully scrawled "***Transaction IDs Are Tags!!***"

Ben's basic idea is this. The current state of the database will be stored in blocks with tag 0. When a transaction t writes a block, the data is stored in the separate block A/t until the transaction commits.

Ben has set aside a special disk address, **ComRec**, to hold commit records for all running transactions. For a valid transaction ID t, the contents of **ComRec**/t is either "committed", "aborted", or "in progress", depending on what state t is in.

So far, begin, read, and write have been implemented. (Below, *t* is a transaction ID, A an address, and *data* a data block.)

```
trans-begin(t) =
  write(ComRec/t,
        "in progress")

trans-read(t, A) =
  if exists(A/t) then
    return read(A/t)
  else /* haven't written yet */
    return read(A/0)

trans-write(t, A, data) =
  write(A/t, data)
```

In parts c and d, you are asked to write pseudocode for commit and abort, and design a system for recovery after a crash. You may want to look through both parts before beginning. **Do not worry about concurrent transactions:** we are only concerned with atomicity in the face of failure.

4c. *(12 points)* Write pseudocode for *trans-commit*(*t*) and *trans-abort*(*t*), and circle the commit point in *trans-commit*. You can assume that the variable *dirty*, an array with *num-dirty* elements, holds all the addresses to which *t* has written. (Don't worry about any garbage an aborted transaction might leave on disk, and assume transaction IDs are never reused.)

4d. *(10 points)* How would you implement recovery after a crash? Ben has already done some of the work: his code examines the **ComRec** blocks and determines which transactions are committed, aborted, or in progress. When your pseudocode is called, he has already set 6 variables for you (you might not need them all):

num-committed	the number of committed transactions
committed	an array holding the committed transactions' transaction IDs
num-aborted	the number of aborted transactions
aborted	an array holding the aborted transactions' transaction IDs
num-in-progress	the number of transactions in progress
in-progress	an array holding the in-progress transactions' transaction IDs

(*Hint:* You may assume that addresses, A, run from 0 to *N*.)

Practice Quiz #3 --- 1997

1. [8 points]: RAID 5 differs from RAID 4 in that parity is spread among drives, with the advantage that

    A. Single-drive failures are tolerated.
    B. Read and write performance in the absence of errors is enhanced.
    C. Write performance in the absence of errors is enhanced.
    D. Locating data on the drives is easier.
    E. There's no real advantage, its just another stupid fad.

(Circle letter to left of best answer)

2. [8 points]: A certain disk drive takes an average of 10 ms to move its head to an arbitrary track, rotates one full revolution every 8 ms, and has 10 sectors per track. What is the average time necessary for an arbitrary 1-sector read?

    A. 10 ms.
    B. 10.8 ms.
    C. 14 ms.
    D. 14.8 ms.
    E. 18 ms.
    F. 18.8 ms

(Circle letter to left of best answer)

3. [8 points]: Two-phase locking within transactions assures that

    A. No deadlocks will occur.
    B. Results will correspond to some serial execution of the transactions.
    C. Resources will be locked for the minimum possible interval.
    D. Neither gas nor liquid will escape.
    E. Transactions will succeed even if one lock attempt fails.

(Circle letter to left of best answer)

 Name ________________________________

4. [8 points]: The ATM of a dishonest bank in the Cirrus system might cheat by claiming to have dispensed money when in fact it hadn't. Cirrus detects this by

A. Multi-key hyperbolic affine encryption techniques.

B. Smart cards.

C. Batch posting.

D. Four-message protocol.

E. Customer complaints.

(Circle letter to left of best answer)

Sick Transit

Gloria Mundi, an MIT alumna who dropped 6.033 immediately following the second quiz, is undertaking to resurrect the failed London Ambulance Service as a new streamlined company called Sick Transit. She has built a new computer she intends to use for processing ST's transactions. Although she never understood the System R paper, she is sure that a much simpler approach can be used for reliable transaction processing.

A key component in Gloria's machine is a highly reliable sequential-access infinite tape, which she plans to use as an **append-only** transaction log. Records can be appended to the tape, but once written are immutable and stable. Records on the tape can be read any number of times, from front-to-back or from back-to-front. There is no disk in the ST system; the tape is the only non-volatile storage.

Because of the high cost of the infinite tape, Gloria compromised on the quality of more conventional components like RAM and CPU, which fail frequently but are fail-fast in the sense that every error causes an immediate system crash. Gloria plans to ensure that, after a crash, a consistent state is recoverable from the log on the infinite tape.

Gloria's code uses transactions, each identified by a unique transaction ID. The visible effect of a completed transaction is confined to changes in global variables, whose WRITE operations are logged. The log will contain entries recording the following operations:

BEGIN(tid) start a new transaction, whose unique ID is *tid*

COMMIT(tid) commit a transaction

ABORT(tid) abort a transaction

WRITE(tid,variable,old-value,new-value)

write a global variable, specifying previous & new values.

To keep the system simple, Gloria plans to use the above forms as the application-code interface to the transaction system, in addition to a *READ(tid,variable)* call which returns the current value of *variable*. Each of the calls will perform the indicated operation and write a log entry as appropriate. Reading an unwritten variable is to return ZERO.

Gloria begins by considering the single-threaded case (only one transaction is active at any time). She stores values of global variables in a symbol table in RAM. Gloria is now trying to figure out how to reset variables to committed values following a crash, using the log tape.

5. [8 points]: In the single-threaded case, what value should variable v be restored to following a crash?

   A. 37
   B. *new-value* from the last logged *WRITE(tid,v,old-value,new-value)* to v, or ZERO if unwritten.
   C. *new-value* from the last logged *WRITE(tid,v,old-value,new-value)* not followed by an *ABORT(tid)*, or ZERO if unwritten.
   D. *new-value* from the last logged *WRITE(tid,v,old-value,new-value)* followed by a *COMMIT(tid)*, or ZERO if unwritten.
   E. Either *old-value* or *new-value* from the last logged *WRITE(tid,v,old-value,new-value)* to *v*, depending on whether that *WRITE* is followed by a *COMMIT* on the same *tid*, or ZERO if unwritten.

(Circle letter to left of best answer)

Gloria now tries running concurrent transactions on her system. Of course, accesses to the log continue to be serialized by the sequential-access tape drive.

Her first trial involves parallel execution of the transactions:

```
BEGIN(t1);                      BEGIN(t2);
{  int t1x=READ(x);             {  int t2x=READ(x);
   int t1y=READ(y);                int t2y=READ(y);
   WRITE(t1,x,t1x,t1y+1);          WRITE(t2,y,t2y,t2x+2);
   COMMIT(t1);                     COMMIT(t2);
}                               }
```

Note that the initial values of x and y are ZERO, as are all uninitialized variables in her system. Here the READ primitive simply returns the most recently written value of a variable from the RAM table, ignoring commits.

6. [8 points]: In the absence of locks or other synchronization mechanism, will the result necessarily correspond to some serial execution of atomic transactions?

   A. Yes.
   B. No, since the execution might result in x=3, y=3.
   C. No, since the execution might result in x=1, y=3.
   D. No, since the execution might result in x=1, y=2.

(Circle letter to left of best answer)

Gloria is considering using locks, and automatically adding code to each transaction so as to guarantee serial consistency of transactions. She would like to maximize concurrency; she is, however anxious to avoid deadlocks. For each of the following proposals, identify whether the approach (1) yields semantics consistent with some sequential execution of atomic transactions and (2) introduces potential deadlocks.

7. [4 points]: A single, global lock which is locked at the start of each transaction and released at COMMIT.

Transaction Atomicity (Y/N)? _____; Introduces potential Deadlocks (Y/N)? ____

8. [4 points]: A lock for each variable. Every READ or WRITE operation is immediately surrounded by a LOCK/RELEASE of that variable's lock.

Transaction Atomicity (Y/N)? _____; Introduces potential Deadlocks (Y/N)? ____

9. [4 points]: A lock for each variable. Every variable READ or WRITTEN within a transaction is locked immediately prior to the first reference to that variable in the transaction; all locks are released at COMMIT.

Transaction Atomicity (Y/N)? _____; Introduces potential Deadlocks (Y/N)? ____

10. [4 points]: A lock for each variable. All variables READ or WRITTEN within a transaction are locked, in alphabetical order, immediately following the BEGIN. They are all released at COMMIT.

Transaction Atomicity (Y/N)? _____; Introduces potential Deadlocks (Y/N)? ____

11. [4 points]: A lock for each variable. All variables WRITTEN within a transaction are locked, in alphabetical order, immediately following the BEGIN. They are all released at COMMIT.

Transaction Atomicity (Y/N)? _____; Introduces potential Deadlocks (Y/N)? ____

In the general case (concurrent transactions) Gloria would like to avoid having to read the entire log during crash recovery. She proposes periodically adding a CHECKPOINT entry to the log, and reading the log backwards from the end restoring committed values to RAM. The backwards scan should end as soon as committed values have been restored to all variables. Each CHECKPOINT entry in the log contains current values of all variables and a list of open (uncommitted) transactions at the time of the CHECKPOINT.

12. [10 points]: What portion of the tape must be read to properly restore values committed at the time of the crash?

    A. All of the tape; checkpoints don't help.

    B. Enough to include the BEGIN from each transaction which was uncommitted at the time of the crash.

    C. Enough to include the last CHECKPOINT, as well as the BEGIN from each transaction which was uncommitted at the time of the crash.

    D. Enough to include the last CHECKPOINT, as well as the BEGIN from each transaction which was uncommitted at the time of the last checkpoint.

(Circle letter to left of best answer)

Simplicity Winns, Gloria's onetime MIT classmate, observes that since global variable values are recoverable from the log their storage in RAM is redundant. She proposes eliminating the RAM as well as all of Gloria's proposed locks, and implementing a ***READ(tid,var)*** primitive which returns an appropriate value of ***var*** by examining the log. Note that, in the following description, the arguments ***tid*** and ***var*** to the ***READ*** call are in bold type.

Simplicity's plan is to implement ***READ*** so that each transaction *t* "sees" the values of global values at the time of *BEGIN(t)*, as well as changes made within *t*. She quickly sketches an implementation of ***READ*** which she claims gives appropriate transaction atomicity semantics:

1. Set variables *winners* to empty list and *prior* to FALSE;
2. Loop over each entry *E* in log, starting at last and working backward:
    - 2.1: if E is *BEGIN(**tid**)* then set *prior* to TRUE;
    - 2.2: if E is *COMMIT(etid)* and *prior* is TRUE, add e*tid* to *winners* list;
    - 2.3: if E is *WRITE(etid,**var**,old-val,new-val)* then
        - 2.3.1: if *etid* is ***tid***, then return *new-val*;
        - 2.3.2: if *etid* is on *winners* list, then return *new-val*;
    - 2.4: continue looping to next earlier entry. If list is exhausted, return 0.

Gloria is a little dazed by Simplicity's quick synopsis, but thinks that Simplicity, who aced 6.033, is likely correct. Gloria asks your help in figuring out what Simplicity's algorithm actually does.

13. [10 points]: Suppose transaction t *READ*s variable x but does not write it. Will each READ of x in *t* see the same value? If so, concisely describe the value returned by each *READ*; if not, explain.

(Explain Briefly)

14. [8 points]: Does Simplicity's scheme REALLY offer atomic transaction semantics yet avoid deadlocks?

    A. Yup. Read it and weep, Gloria.
    B. It doesn't introduce deadlocks, but doesn't guarantee serialized atomic transaction semantics either.
    C. It gives serialized atomic transactions, but introduces possible deadlocks.
    D. Simplicity's approach doesn't work even when there's no concurrency -- it gives wrong values for variables. Maybe she aced 6.033 because her partners on the second project were 6-sigma hackers.
    E. Isn't it about time for this quiz to be over?

(Circle letter to left of all applicable answers)

Gift Department

You all realize that the real motivation of the *Sick Transit* problem is a stupid pun. Let's see if you can identify it:

15. [2 points]: What does *sic transit gloria mundi* actually mean?

A. Thus passes a glorious Monday.

B. Thus pass the earthly glories.

C. Gloria threw up on the T Monday.

D. This bus for First Class and Coach.

E. This is the last straw! If I wanted to take @#%!*% *Latin* tests, I'd have gone to Oxford. I'm reporting this course to the committee on curricular improprieties.

(Circle letter to left of best answer)

Finally, you of course recall the sage advice of our guest lecturer, Bob Metcalfe:

16. [2 points]: Metcalfe suggests you start your writing with:

A. "Bob Metcalfe, visionary inventor of the Ethernet, ..."

B. "You"

C. "The World Wide Web is rapidly gaining popularity"

D. "I"

E. "Howdy, pardner."

Practice Quiz #3 --- 1998

1. A RAID level-5 system

A. Requires more correction disks per data disk than a RAID level-1 system

B. Has a single dedicated correction disk, unlike a RAID level-3 system

C. Requires more correction disks per data disk than a RAID level-3 system

D. Is faster than a RAID level-4 system for workloads that are mostly small writes

E. None of the above.

(circle letter for best answer on answer sheet)

2. The goal of N-version programming is to reduce the probability that
   - A. A software bug will cause a failure
   - B. The last version of a program will be lost
   - C. A communications protocol will have a security flaw
   - D. Secret information will be stolen by malicious code
   - E. None of the above

(circle letter for best answer on answer sheet)

3. Shadow pages are used in System R
   - A. To recover to a consistent state after a system crash
   - B. To ensure that the log is not corrupted in a system crash
   - C. To provide continuous operation even when one data disk is offline
   - D. To avoid taking a checkpoint in the middle of a transaction
   - E. None of the above.

(circle letter for best answer on answer sheet)

4. Which of the following are changes that would occur if Coda were changed to use pessimistic rather than optimistic replica control?
   - A. Users would no longer be able to change any file on their laptops while disconnected from the servers.
   - B. Reads would no longer be guaranteed to return the most up-to-date version of the data.
   - C. Users would no longer have to manually reconcile conflicting changes to the same file.
   - D. A file might be unavailable to a user even though the server is reachable and functioning normally, unlike in the current design where files are always available if the server is reachable and functioning.
   - E. Files cached on a client machine would continue to be usable after it is rebooted while disconnected, unlike in the current design.

(circle letters for ALL THAT APPLY on answer sheet)

5. What was the primary design goal of the log-structured file system? When compared to the UNIX fast file system, using the log structure

   A. Improves the performance of workloads dominated by small writes

   B. Improves the performance of workloads dominated by small reads

   C. Improves the performance of segment cleaning

   D. Enables recovery to a consistent disk state after a crash

   E. Enables atomic updates to multiple files in a single transaction

(circle letter for best answer on answer sheet)

II. BCPL

Ben Bitdittle graduates from MIT and doesn't get a good programming job because he didn't do a UROP. He decides to go into business. He bids $1 at a Resolution Trust Corporation auction and becomes the owner of the Bank of Central Peoria, Limited (BCPL).

When he arrives at BCPL, Ben is shocked to learn that the only programmer who understood BCPL's database has left the company to work on new animation techniques for South Park. Hiring programmers is difficult in Peoria, so Ben decides to take over the database code himself.

Ben learns that an account is represented as a structure with the following information.

```
struct account {
        int number;      /* account number */
        int balance;     /* account balance */
}
```

The BCPL system implements a standard transaction interface for accessing accounts.

```
tid = Begin();
```
Start a new transaction that will be identified as number `tid`.

```
balance = Read(tid, account number);
```
Return the balance of an account (called by transaction number `tid`).

```
Write(tid, account number, newbalance);
```
Update the balance of an account (called by transaction number `tid`).

```
Commit(tid);
```
Make the updates written by transaction number `tid` visible to other transactions.

The BCPL system uses two disks, both accessed synchronously (i.e., read and write operations on the disks won't return until the data is read from the disk or has safely been written to the disk, respectively). One disk contains nothing but the account balances, indexed by number. This disk is called the *database disk*. The other disk is called the *log disk* and exclusively stores, in chronological order, a sequence of records of the form:

```
struct logrecord {
        int op;           /* write, commit, or completed */
        int tid;          /* transaction number */
        int account;      /* account number */
        int new_balance;  /* new balance for account "account" */
}
```

where the meaning of each record is given by the `op` field:

```
op=WRITE        Update of an account to a new balance by
                transaction tid
op=COMMIT       Transaction tid's updates are now visible to
                other transactions and stable across crashes
op=COMPLETED    Transaction tid's writes have been
                applied to the balances on the database disk
```

For each active transaction, the BCPL system keeps a list in main memory called `intentions` containing pairs `(account, newbalance)`. The implementation of `Read` is

```
int Read(tid, account) {
   if (account is in intentions list for tid) {
      get last pair for that account from intentions list for tid
      return newbalance field of that pair
   } else {
      fetch account structure for that account from database disk
      return balance field of that structure
   }
}
```

A. Recovery

Ben asks whether the database computer has ever crashed and learns that it crashed frequently due to intense sound vibrations from the jail next door. Ben decides he had better understand how recovery works in the BCPL system.

In Section A, you can assume that transactions are run serially (i.e., no concurrent transactions).

Ben examines the implementation of the recovery procedure. He finds the following code:

```
recovery()
   winners = null
   for each record L of the log from oldest to newest:
      if (L.op == COMMIT) add L.tid to end of winners list
      if (L.op == COMPLETED) remove L.tid from winners list
   for each record L of the log from oldest to newest
      if (L.op == WRITE and L.tid is in winners list)
         update database disk record for account L.account
            with balance L.new_balance
 • for each tid in winners list from first to last
 •    append a COMPLETED record for tid to the end of the log
```

6. What would happen if the two lines marked with a bullet were omitted?

   A. The system might fail to recover correctly from the first crash that occurs.

   B. The system would recover correctly from the first crash but the log would be corrupt so it might fail to recover correctly from the second crash.

   C. The system would recover correctly from multiple crashes but would have to do more work when recovering from the second and subsequent crashes.

   D. None of the above.

(circle letter for best answer on answer sheet)

7. For the `recovery` and `read` procedures to be correct, which of the following could be correct implementations of the `commit` function? **:**

```
A. commit(tid) {}

B. commit(tid) {
      apply updates in intentions list for tid
              to database disk and clear list
      append COMMIT record to log
   }

C. commit(tid) {
      append COMPLETED record to log
      apply updates in intentions list for tid
              to database disk and clear list
      append COMMIT record to log
   }

D. commit(tid) {
      append COMMIT record to log
      apply updates in intentions list for tid
              to database disk and clear list
      append COMPLETED record to log
   }
```

(circle letters for ALL THAT APPLY on answer sheet)

8. For the `recovery` and `read` procedures to be correct, which of the following could be correct implementations of the `write` function?**:**

```
A. write(tid, account, newbalance) {
      append WRITE record to log
   }

B. write(tid, account, newbalance) {
      add an entry to the end of intentions list for tid
      append WRITE record to log
   }

C. write(tid, account, newbalance) {
      append WRITE record to log
      add an entry to the end of intentions list for tid
   }

D. write(tid, account, newbalance) {
      append WRITE record to log
      add an entry to the end of intentions list for tid
      update database disk record for account
   }
```

(circle letters for ALL THAT APPLY on answer sheet)

Ben is rather surprised to see there is no `abort(tid)` procedure that terminates a transaction and erases its database updates. He calls up the database developer who says it should be easy to add. Ben figures he might as well add the function now, and adds a new log record type ABORT.

9. Which of the following could be correct implementations of the `abort` function? You may assume the `recovery` function is changed correspondingly.**:**

```
A. abort(tid) {
       clear intentions list for tid
   }
B. abort(tid) {
       append ABORT record to log
   }
C. abort(tid) {
       append ABORT record to log
       clear intentions list for tid
   }
D. abort(tid) {
       clear intentions list for tid
       append ABORT record to log
   }
```

(circle letters for ALL THAT APPLY on answer sheet)

B. Buffer cache

BCPL is in intense competition with the nearby branch of Peoria Authorized Savings, Credit and Loan. BCPL's competitive edge is lower account fees. Ben decides to save the cost of upgrading the computer system hardware by adding a main memory buffer cache, which will make the database much more efficient on the current hardware. The buffer cache is used for reads and writes to the database disk only; reads and writes to the log disk are still synchronous.

The buffer cache uses LRU replacement. Each account record on the database disk is cached or replaced separately. In other words, the cache block size, disk block size, and account record sizes are identical.

In Section B, you can assume that transactions are run serially (i.e., no concurrent transactions).

10. Why will adding a buffer cache for reads to the database disk make the system more efficient?**:**

A. It is faster to read the buffer cache than to read the disk.

B. If common access patterns can be identified, performance can be improved by prefetching multiple account balances into the cache.

C. It reduces the total number of disk reads when one transaction reads the same account balance multiple times without updating it.

D. It reduces the total number of disk reads when multiple consecutive transactions read the same account balance.

(circle letters for ALL THAT APPLY on answer sheet)

Ben then makes a mistake (after all, he didn't do a UROP). He reasons that the intentions list described in Section A is now unneccessary, since the list just keeps in-memory copies of database disk data, which is the same thing done by the buffer cache. He deletes the intentions list code and modifies `write` so it updates the copy of the account balance in the buffer cache. He also modifies the system to delay writing the COMPLETED record until all buffered accounts modified by that transaction have been written back to the database disk. Much to his horror, the next time the inmates next door plan an escape and cause the BCPL system to crash, the database does not recover to a consistent state.

17. [9 points]: What might have caused recovery to fail?

A. The system crashed when only some of the modifications made by a committed transaction had reached the database disk.

B. The LRU replacement policy updated the database disk with data modified by an uncommitted transaction, which later committed before the crash.

C. The LRU replacement policy updated the database disk with data modified by an uncommitted transaction, which failed to commit before the crash.

D. The LRU replacement policy updated the database disk with data modified by a committed transaction, which later completed before the crash.

E. The LRU replacement policy updated the database disk with data modified by a committed transaction, which did not complete before the crash.

(circle letter for best answer on answer sheet)

C. Concurrency

Ben deletes the buffer cache code and goes back to the simpler system described in Section A (i.e., the one with intention lists).

Ben is finally ready to investigate how the BCPL system manages multiple concurrent transactions. He calls up the developer and she tells him that there is a mutex lock stored in main memory for each account in the database, used by the `concurrent_begin` and `concurrent_commit` functions. Since BCPL runs concurrent transactions, all its applications use these functions instead of the lower-level `begin` and `commit` functions described earlier.

An application doing a concurrent transaction must declare the list of accounts it will access as an argument to the `concurrent_begin` function.

```
concurrent_begin(list-of-accounts) {
   atomically seize mutexes for all accounts in list-of-accounts
   tid = begin();
   record list-of-accounts in the in-memory data structure for tid
   return tid
}

concurrent_commit(tid) {
1. commit(tid)
2. release mutexes for each account in list-of-accounts for tid
}
```

Ben runs two transactions in parallel. The two transactions are:

```
tida = concurrent_begin(LIST(2));      tidb = concurrent_begin(LIST(2))
   tmpa = read(tida,2);                   tmpb = read(tidb, 2);
   write(tida, 2, tmpa+1);                write(tidb, 2, tmpb+2);
concurrent_commit(tida);               concurrent_commit(tidb);
```

`LIST()` creates a list from its arguments. The initial balance of account 2 before these transactions start is 0.

18. [6 points]: What possible values can account 2 have after completing these two transactions (assuming no crashes)? <Bold>(circle letters for all that apply on answer sheet)

    A. 0
    B. 1
    C. 2
    D. 3
    E. 4

(circle letters for ALL THAT APPLY on answer sheet)

Ben is surprised by the order of the operations in `concurrent_commit`, since `commit` is expensive (requiring synchronous writes to the log disk). It would be faster to release the mutexes first.

19. [9 points]: What possible values can account 2 have after completing these two transactions (assuming no crashes) if lines (1) and (2) of `concurrent_commit` were swapped? Again, the initial balance of account 2 before these transactions start is 0. <Bold>(circle letters for all that apply on answer sheet)**:**

    A. 0
    B. 1
    C. 2
    D. 3
    E. 4

(circle letters for ALL THAT APPLY on answer sheet)