6.033 Spring, 1998 **ANSWER SHEET — TEAR OFF AND HAND IN** Handout 30: May 13, 1998

Name:

Circle your recitation section number:

10:00	1. Gifford/Candea	13. Chapin/Almeida	
11:00	2. Gifford/Almeida	7. Chapin/Candea	12. Karger/Fu
12:00	6. Rinard/Rusnak		
1:00	3. Rivest/Fu	9. Rinard/Sapuntzakis	11. Karger/Kwon
	8. Saltzer/Mazieres	14. Ward/Rusnak	
2:00	4. Rivest/Sapuntzakis	5. Ward/Mazieres	10. Saltzer/Kwon

Instructions: When you mark an answer, circle the letter(s). If you make a mistake and need to change it, circle the Oops field and write your intended answer on the back of this sheet. You should also circle the Oops field and write on the back if you want to explain your answer. Put your name on any extra pages you attach to this answer sheet.

Circle one letter for questions labeled *best*; circle multiple letters for questions labeled *all*.

									Do not write in this box
1	RAID	best	A	B	C	D	E	Oops	1 /6
2	N-version programming	best	A	B	C	D	E	Oops	2 /6
3	Shadow pages	best	A	B	C	D	E	Oops	3 /6
4	Coda	**all**	A	B	C	D	E	Oops	4 /6
5	Log-structured file system	best	A	B	C	D	E	Oops	5 /6
6	System R recovery	best	A	B	C	D	E	Oops	6 /6
7	Removing lines from recovery	best	A	B	C	D		Oops	7 /6
8	Commit	**all**	A	B	C	D		Oops	8 /9
9	Write	**all**	A	B	C	D		Oops	9 /9
10	Abort	**all**	A	B	C	D		Oops	10 /9
11	Buffer cache performance	**all**	A	B	C	D		Oops	11 /6
12	Reasons for recovery failure	best	A	B	C	D	E	Oops	12 /9
13	Concurrent transactions	**all**	A	B	C	D	E	Oops	13 /6
14	Swapped lines in commit	**all**	A	B	C	D	E	Oops	14 /9
15	Dertouzos lecture	best	1	2	3	4		Oops	15 /1
									TOTAL: /100

Name: ____

6.033 Computer System Engineering: Spring, 1998

Quiz 3

All problems on this exam are multiple-choice questions. In order to receive credit you must circle the correct answer or answers for each question **ON THE ANSWER SHEET**.

Some problems 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 problems ambiguous, be sure to circle the *Oops* field and write down any assumptions you make **ON THE BACK OF THE ANSWER SHEET**.

THIS IS AN OPEN BOOK, OPEN NOTES QUIZ.

I. Any of this sound familiar?

1. [6 points]: 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. [6 points]: 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. [6 points]: 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. [6 points]: Which of the following are changes that would occur if Coda were changed to use pessimistic rather than optimistic replica control? **(circle letters for all that apply on answer sheet)**

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. [6 points]: 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).

6. [6 points]: What operation does System R do during crash recovery that cannot be done in BCPL without scanning the entire BCPL log?

A. Redo the database disk updates of uncommitted transactions.

B. Redo the database disk updates of committed transactions.

C. Undo the database disk updates of uncommitted transactions.

D. Undo the database disk updates of committed transactions.

E. None of the above

(circle letter for best answer on answer sheet)

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
```

7. [6 points]: 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)

8. [9 points]: For the `recovery` and `read` procedures to be correct, which of the following could be correct implementations of the `commit` function? **(circle letters for all that apply on answer sheet):**

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)

9. [9 points]: For the `recovery` and `read` procedures to be correct, which of the following could be correct implementations of the `write` function? **(circle letters for all that apply on answer sheet):**

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.

10. [9 points]: Which of the following could be correct implementations of the `abort` function? You may assume the `recovery` function is changed correspondingly. **(circle letters for all that apply on answer sheet):**

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).

11. [6 points]: Why will adding a buffer cache for reads to the database disk make the system more efficient? **(circle letters for all that apply on answer sheet):**

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.

12. [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.

13. [6 points]: What possible values can account 2 have after completing these two transactions (assuming no crashes)? **(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.

14. [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. **(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)

III. Give-away department

15. [1 point]: Michael Dertouzos' lecture in 6.033 was

1. 50% vision
2. 30% techno-trash
3. 20% ballet
4. all of the above.

(circle letter for best answer on answer sheet)

END OF QUIZ 3

(some day you'll look back on 6.033 fondly...)