Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033 Computer Systems Engineering: Spring 2003

Handout 31 - Quiz III

All problems on this quiz 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. You have 50 minutes to answer this quiz.

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.
NO PHONES, NO LAPTOP, NO PDAS, ETC.

CIRCLE your recitation section number:

10:00	1. Lynch/Muthitacharoen	10. Horn/Bodhi	
11:00	2. Lynch/Li	6. Nagpal/Bodhi	11. Horn/Muthitacharoen
12:00	5. Nagpal/Li		
1:00	3. Jackson/Krevat	7. Saltzer/Dabek	8. Rinard/Laas
2:00	4. Jackson/Dabek	9. Saltzer/Laas	12. Rinard/Krevat

Do not write in the boxes below

1-4 (xx/30)	5 (xx/10)	6-8 (xx/30)	9-12 (xx/30)	Total (xx/100)

Name:

I Reading questions

1. [10 points]: In system R (Reading #19), the LOG file is

(Circle ALL that apply)

A. shadowed

B. mirrored

C. duplexed

D. archived

E. forced at every COMMIT

F. marked as a "winner" during a crash recovery

2. [10 points]: Consider each of the following benchmark workloads. For each workload, circle LFS if the LFS file system will complete the test first. Circle FFS if the Fast File System will complete the test first. Circle Tie if neither of the two file systems has a clear performance advantage.

Assume that FFS and LFS work as described in Reading #16, but with a segment size of 100 megabyte. The machine performing the benchmarks has a 100-megabyte buffer cache for data blocks and enough RAM to cache the LFS inode table. Each test starts with an empty file system and you can ignore cleaning of segments.

Some tests require a preparation phase. For your answers, however, you should consider only how the file systems perform on the test phase.

(Circle the right answers)

A. Preparation: create a 1-gigabyte file sequentially. Test: read the file sequentially.

LFS **FFS** **Tie**

B. Preparation: create a 1-gigabyte file; re-write all 1-kilobyte blocks of the file in a pseudo-random order. Test: Read the modified 1-gigabyte file sequentially.

LFS **FFS** **Tie**

C. Preparation: Seed the pseudo-random number generator with "6033"; create a 1-gigabyte file; re-write all 1-kilobyte blocks of the file in a pseudo-random order. Test: Seed the pseudo-random number generator with "6033"; read 1-kilobyte blocks from the modified 1-gigabyte file in the same pseudo-random order.

LFS **FFS** **Tie**

D. Preparation: Write a 1-gigabyte file sequentially. Test: Read 100,000 1-kilobyte blocks from the file at random locations.

LFS **FFS** **Tie**

3. [5 points]: The high performance implementation of a transaction system described in lecture uses undo records to?

(Circle BEST answer)

A. decouple commit from recovery

B. decouple cache replacement from commit

C. decouple redo of actions from recovery

D. decouple recovery from install

4. [5 points]: In lecture, Professor Jackson and Saltzer agree with Brooks's "mythical man-month" on

(Circle ALL that apply)

A. conceptual integrity is desirable in a design

B. it is difficult to come up with the right abstractions for a system

C. information flow diagrams are an essential design component

D. in general a designer's second system comes out better than the first

II Feature Interaction

Alice decided to try out MIT's new RFID Student Tracking System, so she created an access control list that allows a few close friends to track her. One of those friends, Bob, wants to ask Alice to join his 6.033 design project team, so this morning he requested that the tracking system give him a callback if Alice walks by the Great Sail. Alice, working in the Athena Cluster in 66-080, belatedly realizes that Bob is probably going to pop that question, so she logs in to the tracking system and removes Bob from her access control list. She then logs out and leaves for lunch. As she walks by the Great Sail, Bob comes running out of the library to greet her, saying that he just received a callback from the tracking system.

5. [10 points]: The designer of the tracking system made a security blunder. What is it?

(Circle the one most likely answer)

A. The tracking system didn't properly erase residues.

B. In her rush to leave for lunch, Alice removed Lucy, rather than Bob, from her ACL.

C. The tracking system has a time-of-check to time-of-use bug.

D. The system used a version of SSL that is subject to cipher substitution attacks.

E. The system did not require a face-to-face rendezvous between users and system administrator.

Name:

III Reconciling file system replicas

After reading the CODA paper and the end-to-end paper, Alice explores designing an application for reconciling UNIX file systems. Her program, *reconcile*, takes as input the names of two directory trees and attempts to reconcile the differences between the two trees. The typical scenario for *reconcile* is that one of the directory trees is on a server (e.g., an Athena locker). The other one is a replica of the server's directory tree, located on Alice's laptop.

When Alice is disconnected from MIT, she modifies files on her laptop while her friends may modify files on the Athena server. When Alice reconnects to MIT, she runs *reconcile* to reconcile the differences between the directory tree on her laptop and the server so that they are identical again. For example, if a file has changed on her laptop, but not on the server, *reconcile* copies the file from the laptop to the server. If the file has changed on both the laptop and server, then *reconcile* requires guidance to resolve conflicting changes.

The *reconcile* program maintains on each host a database (*fsinfo*), which is stored outside the directory tree being reconciled. This database is indexed by pathname, and it stores:

```
char  pathname[1024]; // pathname for this file
int160 hash; // hash = SHA1(content of file's data)
```

On disk a UNIX file consists of metadata (the inode) and content (the data blocks). The SHA1 is computed over the file's content. Pathnames are less than 1024 bytes. For this problem, you can ignore the details of reconciling directories, and assume that Alice has permission to read and write everything in both directory trees.

The *reconcile* program operates in 4 phases:

- Phase 1: Compute the set of changes on the laptop since the last reconciliation and the set of changes on the server since the last reconciliation.
- Phase 2: The laptop retrieves the set of changes from the server. Using the two change sets, the laptop computes a set of actions that must be taken to reconcile the two directory trees. In this phase, *reconcile* might decide that some files cannot be reconciled, because of conflicting changes.
- Phase 3: The laptop carries out the actions determined in phase 2. The laptop updates the files and directories in its local directory tree, and retrieves files, sends files, and sends instructions to the server to update its directory tree.
- Phase 4: Both the laptop and the server update the *fsinfo* they used to reflect the new content of files that were successfully reconciled on this run.

For the rest of the quiz you may assume that *reconcile* runs to completion before starting again. Furthermore, you may assume that when *reconcile* runs no other processes are allowed to modify the file system. You also may assume that initially the *fsinfo* databases are identical in content and computed correctly, and that after reconciliation they also end up in an identical state.

Name:

The first phase of *reconcile* runs the procedure *computeModifications* on both the laptop and the server. On each machine *computeModifications* produces two sets: a set of files that changed on that machine and a set of files that were deleted on that machine.

```
set changeset, deleteset;

procedure COMPUTEMODIFICATIONS(path, fsinfo)
 {
     changeset ← NULL;
     deleteset ← NULL;
     computeChangeSet(path, fsinfo);
     computeDeleteSet(fsinfo);
  }

procedure COMPUTECHANGESET(path, fsinfo)
 {
     info ← lookup(path, fsinfo)
     if info == NULL  then add(path, changeset)
     else if isDirectory(path)  then {
         for each file ∈ path  do computeChangeSet(path/file, fsinfo)
      }
     else if SHA1(content(path)) ≠ info.hash  then add(path, changeset)
  }

procedure COMPUTEDELETESET(fsinfo)
 {
     for each entry ∈ fsinfo  do {
         if not(exist(entry.pathname))  then add(pathname, deleteset)
      }
  }
```

The *computeModifications* procedure takes as arguments the pathname of the directory tree to be reconciled and the *fsinfo* database. The procedure *computeChangeSet* walks the directory tree and checks if a file was created or changed since the last time *reconcile* ran. SHA1 is an example of a cryptographic hash function, which has the property that it is computationally infeasible to find two different inputs i and j such that $SHA1(i) = SHA1(j)$

The *computeDeleteSet* procedure checks for each entry in the database if the corresponding file still exists; if not, it adds it to the set of files that have been deleted since the last run of *reconcile*.

6. [10 points]: What files will *reconcile* add to $changeset$ or $deleteset$?

(Circle ALL that apply)

A. Files whose content has decayed.

B. Files whose content has been modified.

C. Files that have been created.

D. Files whose inode have been modified.

E. Files that have been deleted.

F. Files that have been deleted but recreated with identical content.

G. Files that have been renamed.

The second phase of *reconcile* compares the two *changeset*s and the two *deleteset*s to compute a set of actions for reconciling the two directory trees. To avoid confusion we call the *changeset* on the laptop, *changeLeft*, and *changeset* on the server, *changeRight*. Similarly, the *deleteset* on the laptop is *deleteLeft* and *deleteset* on the server *deleteRight*. The second phase consists of running the procedure *computeActions* with the 4 sets as input; it produces 5 sets:

- *additionsLeft*: files that must be copied from server to the laptop
- *additionsRight*: files that must be copied from laptop to the server
- *removeLeft*: file that must be removed from laptop
- *removeRight*: file that must be removed from server
- *conflicts*: files that have conflicting changes

In the following code fragment, the notation $A - B$ indicates all elements of the set A after removing the elements that also occur in the set B. With this notation, the 5 sets are computed as follows:

conflicts $\leftarrow NULL$;
procedure COMPUTEACTIONS$(changeLeft, changeRight, deleteLeft, deleteRight)$
{
  for each $file \in (changeLeft)$ **do** {
    if $file \in (changeRight \cup deleteRight)$ **then** $add(file, \textit{conflicts})$
  }
  for each $file \in (deleteLeft)$ **do** {
    if $file \in (changeRight)$ **then** $add(file, \textit{conflicts})$
  }
  $additionsRight \leftarrow changeLeft - \textit{conflicts}$;
  $additionsLeft \leftarrow changeRight - \textit{conflicts}$;
  $removeRight \leftarrow deleteLeft - \textit{conflicts}$;
  $removeLeft \leftarrow deleteRight - \textit{conflicts}$;
}

7. [10 points]: What files end up in the set *additionsRight*?

(Circle ALL that apply)

A. Files created on the laptop that don't exist on the server

B. Files that have been removed on the server but not changed on the laptop

C. Files that have been removed on the laptop but not on the server

D. Files that have been modified on the laptop but not on the server

E. Files that have been modified on the laptop and on the server

8. [10 points]: What files end up in the set *conflicts*?

(Circle ALL that apply)

A. Files that have been modified on the laptop and on the server

B. Files that have been removed on the laptop but that exist unmodified on the server

C. Files that have been removed on the laptop and on the server

D. Files that have been modified on the server but not on the laptop

E. Files that have been created on the laptop and on the server but with different content

F. Files that have been created on the laptop and on the server with the same content.

Phase 3 of the *reconcile* executes the actions: deleting files, transfering files, and resolving conflicts. From now on, you can assume that conflicts are resolved by asking the user.

We focus on transfering files from laptop to server. Alice wants to ensure that transfers of files are atomic. You may assume that all file system calls execute atomically. The *reconcile* program transfers files from *additionsRight* by invoking the remote procedure *receive* on the server:

$boolean$**procedure** RECEIVE$(data, size, path)$
$\{$
  $name \leftarrow uniquename();$
  $fd \leftarrow creat(name);$
  if $fd >= 0$ **then** $\{$
    $n \leftarrow write(fd, data, size);$
    $close(fd);$
    if $n == size$ **then** $rename(name, path);$
    else $unlink(name);$
    $return(n == size);$
  $\}$
  else $return(false);$
$\}$

The *receive* procedure takes as argument the new content of the file (*data* and *size*) and the name of the file to be updated or created. First, *receive* creates a temporary file with some unique name (*name*) and writes the data into in it. After the write is successful, *receive* renames the temporary file to its real name (*path*), removing the old version of *path* first, if it exists; otherwise, it cleans up and deletes the temporary file. (You may assume that *rename* always executes successfully.)

9. [5 points]: Where is the commit point in the procedure *receive*?
(Circle the BEST answer)

A. right after *rename* completes

B. right after *close* completes

C. right after *create* completes

D. right after *unlink* completes

E. right after *write* completes

F. none of the above

Name:

After the server or laptop fails, it calls a recovery procedure to back out or roll forward a *receive* operation that was in progress when the host failed.

10. [5 points]: What must this recovery procedure do?

(Circle the BEST answer)

A. Remove any temporary files left by *receive*

B. Nothing

C. Send a message to the sender to restart the file transfer

D. Rename any temporary files left by *receive* to their corresponding pathname

11. [10 points]: Which advantages does the *reconcile* program have over the CODA reconciliation procedure (as described in reading #20)

(Circle ALL that apply)

A. *reconcile* repairs files that decay

B. *reconcile* doesn't require changes to the UNIX file system implementation

C. *reconcile* doesn't require a log on the laptop

D. *reconcile* propagates changes from the laptop to the server, and vice versa.

E. *reconcile* will run much faster on big files

Alice wonders if her code extends to reconciling more than two file systems. Consider 3 hosts (A, B, and C) that all have an identical copy of a file f, and the following sequence of events:

- at noon B modifies file f
- at 1pm B reconciles with A
- at 2pm C modifies f
- at 3pm B reconciles with C
- at 4pm A modifies f
- at 5pm B reconciles with A

You should assume that B has two *fsinfo* databases, one used for reconciling with A and one for reconciling with C.

12. [10 points]: Which of the following statements are correct, given this sequence of events and Alice's implementation of *reconcile*?

(Circle ALL answers that apply)

A. If the conflict at 3pm is reconciled in favor of B's copy, then *reconcile* will not report a conflict at 5pm.

B. If the conflict at 3pm is reconciled in favor of C's copy, then *reconcile* will report a conflict at 5pm.

C. If the conflict at 3pm is resolved by a modification to f that merges B's and C's versions, then *reconcile* will report a conflict at 5pm.

D. If the conflict at 3pm is resolved by removing f from B and C, then *reconcile* will not report a conflict at 5pm.

End of Quiz III — enjoy your summer!