6.033 Computer System Engineering

Spring 2009

Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033 Computer Systems Engineering: Spring 2007

Quiz I Solutions

There are 12 questions and 12 pages in this quiz booklet. Answer each question according to the instructions given. You have **50 minutes** to answer the questions.

Most questions are multiple-choice questions. Next to each choice, circle the word **True** or **False**, as appropriate. A correct choice will earn positive points, a wrong choice may earn negative points, and not picking a choice will score 0. The exact number of positive and negative points for each choice in a question depends on the question and choice. The maximum score for each question is given near each question; the minimum for each question is 0. Some questions are harder than others and some questions earn more points than others—you may want to skim all questions before starting.

If you find a question ambiguous, be sure to write down any assumptions you make. **Be neat and legible.** If we can't understand your answer, we can't give you credit!

Write your name in the space below AND at the bottom of each page of this booklet.

THIS IS AN OPEN BOOK, OPEN NOTES QUIZ.
NO PHONES, NO COMPUTERS, NO LAPTOPS, NO PDAS, ETC.

CIRCLE your recitation section number:

10:00	1. Madden/Komal		
11:00	2. Madden/Zhang	3. Katabi/Komal	10. Yip/Chachulski
12:00	4. Yip/Zhang	5. Katabi/Chachulski	
1:00	6. Ward/Shih	7. Girod/Schultz	
2:00	8. Ward/Schultz	9. Girod/Shih	

Do not write in the boxes below

1-4 (xx/30)	5-7 (xx/26)	8-10 (xx/22)	11-12 (xx/22)	Total (xx/100)

Name:

I Reading Questions

1. [6 points]: Which of the following statements are true of the X Windows System (as described in the X Windows paper, reading # 5)?

(Circle True or False for each choice.)

A. True / False When a program such as Xterm is run remotely and displays its window on your local workstation, that *remote* machine is considered an X server.

FALSE. The X server is the program that manages your display, keyboard, and mouse.

B. True / False In X Windows, interactive widgets such as pulldown menus and clickable buttons are implemented by the client.

TRUE. The X server provides only the most basic drawing mechanisms; higher level interactions and policy decisions are left to the client applications.

C. True / False Like most client/server protocols, the X Windows protocol exclusively uses big-endian byte ordering for all network communication.

FALSE. X Windows negotiates the byte ordering in order to eliminate redundant conversions. In the paper this uses two separate ports; modern servers use a negotiated protocol.

2. [8 points]: The UNIX designers chose to have the process created by a `fork()` share (inherit) the open file descriptors of its parent. Which of the following would become more complicated had they decided on an alternate `fork()` semantics, in which the child process is created with no open file descriptors?

(Circle True or False for each choice.)

A. True / False The management of standard I/O streams of child processes.

TRUE. Standard I/O streams are file descriptors, and by default child processes use the same standard I/O streams as parents.

B. True / False Structure of inodes as stored on the disk.

FALSE. The structure of inodes is not affected by the way in which file descriptors are passed from parents to children.

C. True / False The format of directories as stored on the disk.

FALSE. Directory structure is also unaffected by the mechanism for sharing file descriptors between children and parents.

D. True / False The creation of pipes between processes.

TRUE. For the pipe() system call to work as specified in Unix, it is necessary for children and parents to share a file descriptor.

Name:

3. [8 points]: Ben Bitdiddle is using a MapReduce cluster (as described in reading #8) to process some data. He finds that some of his programs are running slowly and needs your help to understand which performance enhancements to try. For each of the following MapReduce configurations and suggested enhancements, circle **True** if the statement about the enhancement is true, and **False** otherwise.

A. True / False If the performance of the system is not limited by network throughput or the rate at which map tasks can read from or write to the disk, then doubling the number of nodes in the system will probably improve performance if the number of map and reduce tasks is greater than the number of nodes.

TRUE. Doubling the number of nodes will make more resources available to process map and reduce tasks, and there are abundant map and reduce tasks waiting to be processed.

B. True / False If some tasks on some workers take much longer than others to complete, and these "stragglers" are the bottleneck in the system, then allocating additional nodes to these straggler tasks will definitely improve performance.

FALSE. Some straggler tasks may be slow because the amount of work they have to perform is large, not because they were assigned to a slow machine (which would be fixed by allocating additional nodes to the straggler tasks.)

C. True / False Assume that there are 1000 machines, 1000 map tasks and 100 reduce tasks, and that the CPU processing time for each map or reduce task to execute on a single machine is 1 second. If the entire 1000-node cluster can complete these MapReduce operations in 100 seconds then doubling the CPU performance (task processing rate) of each machine in the cluster may improve performance somewhat but not significantly.

TRUE. Performance will not increase much since the 1100 tasks can be completed in about 1.1 seconds by the current 1000 node cluster, so halving this (to .55 seconds) won't help the overall 100 second runtime very significantly.

D. True / False Suppose local disk reads and writes are 100 times as fast as network transfers and the coordinator is never able to achieve locality by assigning map workers to map input tasks stored on their local disks. In that case, if the system has an I/O bottleneck, then doubling network throughput will substantially improve performance.

TRUE. Since all data that is written to disk also has to be transferred over the network (due to the lack of locality), and disk is 100 faster than network, increasing network throughput will substantially decrease total runtime.

4. [8 points]: Which of the following statements about operating system kernels are true?
(Circle True or False for each choice.)

A. True / False Preemptive scheduling allows the kernel's thread manager to run applications in a way that helps avoid fate sharing.

TRUE. Preemptive scheduling prevents applications that are in an infinite loop from stalling other applications.

B. True / False The kernel serves as a trusted intermediary between programs running on the same computer.

TRUE. The kernel is responsible for mediating between programs by managing shared memory, pipes, etc.

C. True / False In an operating system that provides virtual memory, the kernel must be invoked to resolve every memory reference.

FALSE. For most memory references, no code inside the kernel runs. The kernel may be invoked to handle page fault exceptions.

D. True / False When a kernel switches a processor from one application to another application, the target application sets the PMAR register appropriately after it is running in user space.

FALSE. The kernel sets the PMAR register before switching to user mode and beginning execution of the target application.

II RPC SEMANTICS

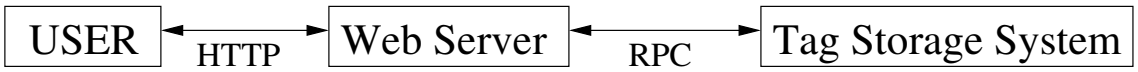
Ben is in charge of system design for Stickr, a new website for posting pictures of bumper stickers and tagging them. Luckily for him, Alyssa had recently completed implementing the Tag Storage System from design project 1. Alyssa's design supports the interface suggested by the design project, and has the following specification:

procedure FIND$(subj, rel, obj, start, count)$
// returns ok + array of matching tags

procedure INSERT$(subj, rel, obj)$
// returns ok (duplicate tags ignored)

procedure DELETE$(subj, rel, obj)$
// returns boolean, true if tag existed, false otherwise

Ben comes up with the following design:



To implement this, Ben uses an RPC interface to allow the web server to interact with the tag system. Ben chooses *at-least-once* RPC semantics. You may assume that the tag storage system does not crash (which would result in unspecified behavior), but the network between the web server and tag storage system is unreliable and may drop messages.

5. [8 points]: Suppose only a single thread on Ben's Web server is using the tag storage system and that this thread issues just one RPC at a time. What types of incorrect behavior can the Web server observe?

(Circle True or False for each choice.)

A. True / False The FIND RPC stub on the web server sometimes returns no results when matching tags exist in the storage system.

FALSE. At-least-once will keep trying until it succeeds. When it succeeds, it will always return tags if they exist in the system. Since the web server is single threaded, the requests will be serialized.

B. True / False The INSERT RPC stub on the web server sometimes returns ok without inserting the tag into the storage system.

TRUE and FALSE (credit given for either). If the RPC returns OK, it is certain that the given tags are now present in the system. However, if the tags were duplicates, OK could be returned without "inserting" the tags.

C. True / False The DELETE RPC stub on the web server sometimes returns false when it actually deleted a tag.

TRUE. Suppose the first delete succeeds, tries to return 'true', but the return value is dropped. When the delete is retried the tag will already be deleted and the delete will now return false.

D. True / False The FIND RPC stub on the web server sometimes returns tags that have been deleted.

FALSE. Deleted tags will not be in the system and therefore cannot be reported. Since the web server is multithreaded, requests to delete and find will be serialized.

6. [8 points]: Suppose Ben switches to *at-most-once* RPC; if no reply is received after some time, the RPC stub on the web server terminates with a timeout error code. Assume again that only a single thread on Ben's Web server is using the tag storage system and that this thread issues just one RPC at a time. What types of incorrect behavior can the Web server observe?

(Circle True or False for each choice.)

A. True / False Assuming it does not timeout, the FIND RPC stub on the web server can sometimes return no results when matching tags exist in the storage system.

FALSE. At most once tries exactly once. If the RPC does not time out, it must have succeeded on the first try. Thus, if matching tags are present, they must be reported.

B. True / False Assuming it does not timeout, the INSERT RPC stub on the web server can sometimes return ok without inserting the tag into the storage system.

TRUE and FALSE (credit given for either). If it does not time out, the RPC must succeed. The insert will therefore guarantee that they keys are present after the call completes. However, if the tags were duplicates, OK could be returned without "inserting" the tags.

C. True / False Assuming it does not timeout, the DELETE RPC stub on the web server can sometimes return false when it actually deleted a tag.

FALSE. If it does not time out, the RPC must succeed. The response from the delete will therefore not be lost, and will return false iff the tag was not present.

D. True / False Assuming it does not timeout, the FIND RPC stub on the web server can sometimes return tags that have been deleted.

FALSE. If it does not time out, the RPC must succeed. The find will therefore succeed and cannot return tags that had been deleted.

III NAMING

Bob and Alice are using a file UNIX system as described in Appendix 2A. The file system has two disks, mounted as `/disk1` and `/disk2`. A system administrator creates a "`home`" directory containing symbolic links to the home directories of Bob and Alice via the commands:

```
mkdir /home
ln -s /disk1/alice /home/alice
ln -s /disk2/bob /home/bob
```

Subsequently, Bob types the following to his shell:

```
cd /home/alice
cd ../bob
```

and gets an error.

7. [10 points]: Which of the following best explains the problem?
(Circle the BEST answer)

A. UNIX forbids the use of "`..`" in a cd command when the current working directory contains a symbolic link.

INCORRECT. UNIX never forbids the use of '..' in a cd command.

B. Since Alice's home directory now has two parents, UNIX complains that "`..`" is ambiguous in that directory.

INCORRECT. UNIX does not notice that '..' could have two interpretations.

C. In Alice's home directory, "`..`" is a link to `/disk1`, while the directory "`bob`" is in `/disk2`.

CORRECT. Since '..' points to /disk1, ../bob means /disk1/bob. However, this directory does not exist. Note, that if Bob had been using bash, this would *have changed into his directory. But given that he got an error, this is the only possible reason.*

D. Symbolic links to directories on other disks are not supported in UNIX; their call-by-name semantics allows their creation, but causes an error when they are used.

INCORRECT. Symbolic links to directories on other disks are supported on UNIX. Hard links are not.

E. None of the above.

INCORRECT. (C) is correct.

Name:

IV GOOMBLE

Observing that US legal restrictions have curtailed the booming online gambling industry, a group of former Therac programmers has launched a new venture called Goomble. Goomble's web server allows customers to establish an account, deposit funds using a credit card, and then play the Goomble game by clicking a button labeled **I FEEL LUCKY**. Every such button click debits their account by $1, until it reaches zero.

Goomble lawyers have successfully defended their game against legal challenges by arguing that there's no gambling involved: the Goomble "service" is entirely deterministic.

The initial implementation of the Goomble server uses a single thread, which causes all customer requests to be executed in some serial order. Each click on the **I FEEL LUCKY** button results in a procedure call `Lucky(account)`, where account is a pointer to a data structure representing the user's Goomble account. Among other data, the account structure includes an unsigned 32-bit integer Balance, representing the customer's current balance in dollars.

The Lucky procedure is coded as follows:

```
procedure LUCKY(Account account)
  if account.Balance > 0  then {              // line 1: test
   account.Balance ← account.Balance − 1;     // line 2: update
  }
```

The Goomble software quality control expert, Nellie Nervous, runs Eraser (as described in the Eraser paper, reading #7) on the single-threaded Goomble server code to check for race conditions; Eraser reports no problems.

8. [6 points]: Which conclusions might Nellie draw?

(Circle True or False for each choice.)

A. True / False Eraser reports no problems because it found no shared variables; each variable is accessed by a single thread.

TRUE. Nellie only ran the server with one thread when she tested with Eraser and Eraser only flags variables that are actually accessed by multiple threads during the program's exection. This means that Eraser will not detect the race condition on account.Balance.

B. True / False Eraser reports no problems because the code contains no locks.

FALSE. It is true that Eraser reports no problems, but the reason is correctly stated in 8.A; it is not because there are no locks.

C. True / False The single-threaded server has potential race conditions that Eraser missed because the timing details of Nellie's test run didn't expose them.

FALSE. Again, it is true that Eraser reports no problems, but the reason is correctly stated in 8.A; it is not because the execution happened to use a particular scheduling of instructions.

The success of the Goomble site quickly swamps their single threaded server, limiting Goomble's profits. Goomble hires an MIT-educated server performance expert, Threads Galore, to improve server throughput.

Threads modifies the server as follows: Each **I FEEL LUCKY** click request spawns a new thread, which calls `Lucky(account)` and then exits. All other requests (e.g. setting up an account, depositing, etc) are served by a single thread. Threads argues that the bulk of the server traffic consists of player's clicking **I FEEL LUCKY**, so that his solution addresses the main performance problem.

Unfortunately, Nellie doesn't run Eraser on the multi-threaded version of the server. She is busy with development of a follow-on product: the Goomba, which simultaneously cleans out your bank account and washes your kitchen floor.

Suppose Nellie had run a brief test using Eraser with Goomble's multi-threaded server (and their million-customer audience), sufficient to run each piece of code (e.g., Lucky) many times, and to identify all shared variables.

9. [6 points]: If that test had been run, which of the following would be true?
(Circle the BEST answer)

A. Eraser would definitely find a problem.

CORRECT. Nellie executed a test that was "sufficient to run each piece of code (e.g. Lucky) many times, and to identify all shared variables". Eraser flags variables that are shared by multiple threads, but not protected by a lock. Since the test exercised each piece of code, and was sufficient to identify all shared variables, Eraser would have flaged the race condition on accout.Balance.

B. Eraser would definitely find no problem.

INCORRECT. Contradicts 9.A

C. Eraser might report a problem, depending on the details of the run-time scheduling of the test run.

INCORRECT. Timing does not affect Eraser's results in this case. The only case when timing can affect the results is for "initialization errors" (described in the Eraser paper) and this race condition is not a member of that category.

Willie Windfall, a compulsive Goomble player, has two computers and plays Goomble simultaneously on both (using the same Goomble account). He has mortgaged his house, depleted his retirement fund and the money saved for his kid's education, and his Goomble account is nearly at zero. One morning, clicking furiously on **I FEEL LUCKY** buttons on both screens, he notices that his Goomble balance has jumped to something over four billion dollars.

10. [10 points]: Explain the source of Willie's good fortune. Give a simple scenario involving two threads (T1 and T2) with interleaved execution of lines 1 and 2 in calls to `Lucky(account)`, detailing the timing that would result in a huge *account.Balance*. We have given the first step of the scenario; fill in the subsequent steps (you may not need to use all steps.)

- Step 1. T1 evaluates "if account.Balance > 0", find statement is true
- Step 2. T2 evaluates "if account.Balance > 0", finds statement is true
- Step 3. T2 decrements account.Balance by 1, and the balance is now 0
- Step 4. T1 decrements account.Balance by 1, and the balance is now a very large positive number because we are using unsigned integers.

```
// VERSION 1
procedure LUCKY(Account account)
  ACQUIRE(global_lock);
  if account.Balance > 0 then {
   account.Balance ← account.Balance − 1;
  }
  RELEASE(global_lock);


// VERSION 2
procedure LUCKY(Account account)
  integer temp;
  ACQUIRE(account.lock);
  temp ← account.Balance;
  RELEASE(account.lock);
  if temp > 0 then {
   ACQUIRE(account.lock);
   account.Balance ← account.Balance − 1;
   RELEASE(account.lock);
  }

// VERSION 3
procedure LUCKY(Account account)
  ACQUIRE(account.lock);
  if account.Balance > 0 then {
   account.Balance ← account.Balance − 1;
  }
  RELEASE(account.lock);
```

Figure 1: Possible rewrites

Word of Willie's big win spreads rapidly, and Goomble billionaires proliferate. In a state of panic, the Goomble board calls you in as a consultant to review three possible fixes to the server code to prevent further "gifts" to Goomble customers. Each of the proposals involves use of a lock—either global or specific to an account—to rule out the unfortunate timing bug. Each of the possible rewrites of Lucky is shown in figure 1.

Name:

11. [12 points]: Which of the proposed versions eliminates the timing bug that caused Willie's big win?

(Circle True or False for each choice.)

A. True / False VERSION 1

TRUE. Versions 1 and 3 eliminate the bug, by holding a lock for a period including the test and subsequent decrement of account.Balance. This enforces a mutual exclusion constraint that eliminates the possibility of another thread (running the same code) changes the value of account.Balance between the test and the subsequent write operation.

B. True / False VERSION 2

FALSE. While this version surrounds both the read of account.Balance and its subsequent write by an ACQUIRE/RELEASE pair, the lock is released for a period between the read and the subsequent write. Another thread may decrement account.Balance during this period, invalidating the requirement that it be greater than zero when decremented.

C. True / False VERSION 3

TRUE. See answer for version 1.

12. [10 points]: Which version would you recommend, considering both reliability and performance goals?

(Circle the BEST answer)

A. VERSION 1

B. VERSION 2

C. VERSION 3

VERSION 3. As version 2 remains buggy, A and C are the only plausible choices. But the use of a single global lock in version 1 serializes all executions of the body of Lucky, lowering performance to something approaching that of the single-threaded server. Version 3 avoids this serialization by using a dedicated lock for each account.

End of Quiz I

Name: