Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033 Computer Systems Engineering: Spring 2013

Quiz I Solutions

There are 11 questions and 8 pages in this quiz booklet. Answer each question according to the instructions given. You have **50 minutes** to answer the questions.

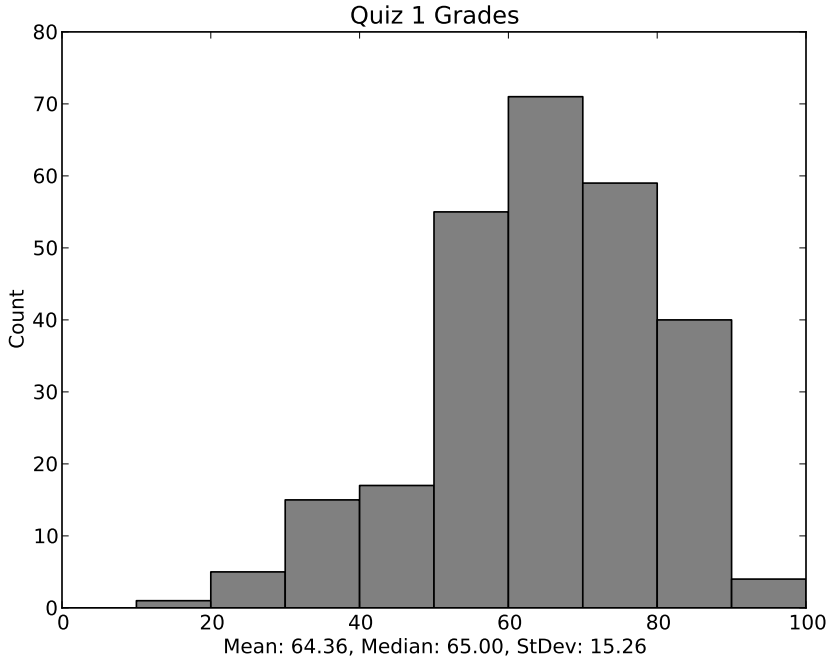
Some questions are harder than others and some questions earn more points than others—you may want to skim all questions before starting.

For true/false and yes/no questions, you will receive 0 points for no answer, and negative points for an incorrect answer. We will round up the score for every *numbered* question to 0 if it's otherwise negative (i.e., you cannot get less than 0 on a numbered question).

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!

THIS IS AN OPEN BOOK, OPEN NOTES, OPEN LAPTOP QUIZ, BUT DON'T USE YOUR LAPTOP FOR COMMUNICATION WITH OTHERS. TURN YOUR NETWORK DEVICES OFF.

Grade distribution histogram:



I DNS

1. [20 points]: Which of the following statements are true and which ones are false?
(Circle True or False for each choice.)

A. True / False A DNS name (e.g., cnn.com) may be associated with multiple IP addresses, but each IP address has to be associated with a single DNS name.

Answer: False

B. True / False DNS caching reduces the time to resolve an IP address, but does not reduce DNS traffic on the Internet

Answer: False

C. True / False If all root DNS servers fail, no DNS names can be resolved to IP addresses

Answer: False

D. True / False A DNS request for the IP address of a host foo.com will be resolved to the same IP address regardless of which machine issues the request.

Answer: False

E. True / False DNS servers remember which clients have cached DNS replies so that the servers can send invalidation messages when name bindings change.

Answer: False

II Unix

Consider UNIX as described in the paper "The UNIX time-sharing system" by Ritchie and Thompson. One UNIX process (we'll call it P, for "producer") is producing chunks of data, to be read in and handled by another UNIX process (which we'll call C, for "consumer"). We might worry that, if P fails during its execution, C will wait around forever for new input, wasting machine resources.

Consider each of the following implementation suggestions, where we start with a single process, and `fork()` it into P and C.

2. [12 points]: Indicate whether each approach **N**ever works with the UNIX API, **S**ometimes works, or **A**lways works. To simplify reasoning, you may assume infinite memory and disk space. By "works" we mean C will not wait around forever and will receive some of the data that P produced before it failed.

A. N S A After the `fork()`, P opens file `/tmp/data` for writing and writes chunks to the file with `write()` as they are ready. C opens `/tmp/data` for reading and receives chunks by reading them from the file with `read()`.

Initials:

Answer: A is incorrect. There is no way for C to know if P has crashed, so C will wait forever in that case. Depending on the implementation of the reading/writing chunk scheme, this approach either sometimes works or never works.

B. **N S A** Before the `fork()`, initialize a bounded buffer in memory. After the `fork()`, P sends data chunks to the bounded buffer, and C receives them from the buffer.

Answer: N. There is no shared memory in Unix V6. `fork` clones the memory rather than sharing it, so each process ends up with a distinct buffer.

C. **N S A** Before the `fork()`, create a pipe. After the `fork()`, P sends data chunks to the pipe with `write()`, and C receives them from the pipe with `read()`.

Answer: A. Since UNIX closes all open file descriptors when a process exits, if P fails then C is guaranteed to exit successfully.

III Eraser

Consider the paper "Eraser: a dynamic data race detector for multithreaded programs" by Savage et al.

Assume that variables v1 and v2 have been initialized to zero by thread T3 prior to invoking threads T1 and T2 whose code is shown below:

```
T1:                          T2:
  acquire(lockA)               acquire(lockA)
  acquire(lockB)               acquire(lockB)
  v1 = v1 + 1                  v1 = v1 + 1
  release(lockA)               release(lockB)
  v2 = v1 + 5                  v2 = v1 + 6
  release(lockB)               release(lockA)
```

3. [5 points]: When Eraser is run on the code above:

(Circle the BEST answer)

A. Eraser will report a race for v1 and v2.

Answer: A (See next question for an explanation)

B. Eraser will report a race for v1 but not for v2.

C. Eraser will report a race for v2 but not for v1.

D. Eraser will not report any races.

Initials:

Suppose Eraser is modified such that $C(v)$ is unaffected by reads, and uses the following rules for generating warnings:

- On each read of v by thread t, if $C(v) == \{\}$, then issue a warning.
- On each write of v by thread t, set $C(v) = C(v) \cap locks\_held(t)$, and if $C(v) == \{\}$, then issue a warning.

4. [4 points]: When Eraser is run on the same above:
(Circle the BEST answer)

A. Eraser will report a race for v1 and v2.

B. Eraser will report a race for v1 but not for v2.

C. Eraser will report a race for v2 but not for v1.

Answer: C. Both versions of Eraser will report v2 as a (correct) race since v2 is being written with different locks being held in the two threads. v1 will be reported as an arguably false race by the original Eraser. In the modified Eraser, v1 is read holding a lock each time, though a different one. With the modified version of Eraser v1 won't be reported as a race.

D. Eraser will not report any races.

IV MapReduce

5. [15 points]: For each *claim* below indicate if it is true or false
(Circle True or False for each choice.)

A. True / False Alice has a large number of images stored in GFS. Each image is in a separate file and Alice wants to know to what extent images are correlated according to some correlation function f. That is, Alice wants to compute $f(F_i, F_j)$ for each pair of images (F_i, F_j). *Claim*: MapReduce is designed to solve this type of problem.

Answer: False. Since each file needs to be compared to each other file, it is not possible to efficiently split this problem into sub problems that can be solved by map and reduce tasks.

B. True / False Alice realizes that stragglers slow down a MapReduce computation and that stragglers often result from slow machines. *Claim*: By scheduling all tasks on two machines, MapReduce definitely improves its performance.

Answer: False. MapReduce should only schedule stragglers to a second machine. In general, performance decreases if all tasks are scheduled to two machines.

Initials:

C. True / False GFS offers fault tolerance by replicating data: If one copy of a file is lost, another copy of the file is likely to be available. To avoid network latency and reduce network usage, MapReduce attempts to assign each map task to a machine that has in its local storage most of the data needed for completing the map task. *Claim*: It is a good idea to store each file and its copies in local storage of a single machine.

Answer: False. This creates a single point of failure: If the machine is inaccessible, the file cannot be retrieved from any other location.

V RON

6. [16 points]: Which of the following statements about RON (Resilient Overlay Networks) are correct, according to the paper by Andersen *et al*?

(Circle True or False for each choice.)

A. True / False The authors' experiments determined that outages occur only in the inner core of the Internet, and never on last-hop links such as cable-modem access links.

Answer: False. The experiments weren't designed to evaluate last-hop failures, and the second dataset did encounter failures RON couldn't route around.

B. True / False When an application requests a path optimized for high throughput, RON uses a path scoring metric based on combinations of measured loss rate and latency.

Answer: True

C. True / False In order to route around transient failures related to BGP reaction time in the core Internet, a RON network should include nodes at locations served by many different ISPs.

Answer: True

D. True / False RON demonstrated that choosing between paths based on application-centric metrics such as latency and loss rate often improves application performance, and concluded that BGP should enact these same link measurements in order to scale up RON's performance gains to the entire Internet.

Answer: False. The first part of the statement is true, second part is not.

VI TCP

Consider the topology in Figure 1, where the delay along each link in the forward direction is 0.1s and the capacity is 100 packets/s. Assume the delay in the reverse direction is negligible (i.e., zero) and the capacity of the links in the reverse direction is infinite; said differently, assume that acks are delivered immediately and never dropped. Let the buffer size at the router be $B = 100$ packets (i.e., the router queues a maximum of B packets). The source and destinations can process packets faster than 100 pkt/s.

Initials:

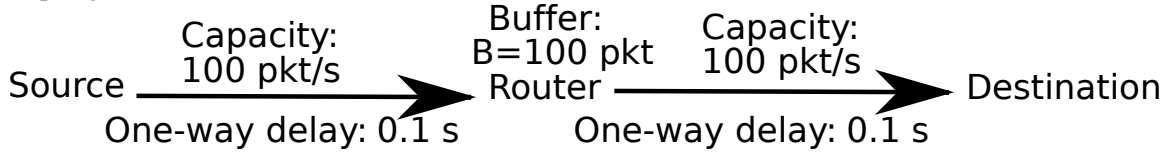


Figure 1: Router with two identical links. The source and destination can process packets faster than 100 pkt/s.

7. [5 points]: Consider a modified TCP, called TCP* which uses a fixed congestion window $cwnd = 18$ packets. Assume a single TCP* connection running between the source and the destination. What is the throughput of this TCP* connection?

Answer: Throughput = 18/0.2 packets/s = 90 packets/s.

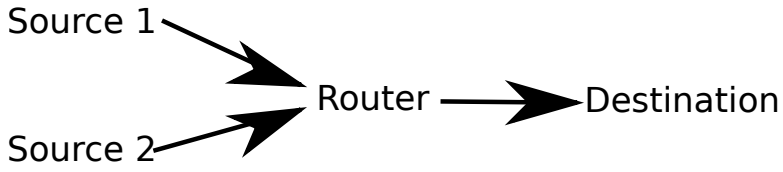


Figure 2: Two sources.

8. [5 points]: Consider running two TCP* connections from Source 1 and Source 2 to the Destination node (as shown in Figure 2). Again, all links are 100 packet/s and 0.1s latency, and all acks are delivered immediately to the sources. What is the average throughput of each of these connections?

Answer: Both connections achieve a throughput of 50 packet/s.

9. [5 points]: For the same setting as the previous question, what is the average RTT of each connection?

Answer: The RTT of both connections is 18/50 second = 0.36s

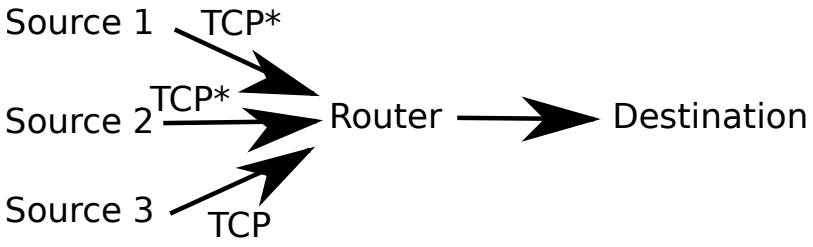


Figure 3: 2 TCP* connections and one TCP connection.

10. [5 points]: Consider the same topology after adding a third link with a source that runs a standard TCP (as shown in Figure 3). All connections have the same destination node. The capacity of all

forward links is 100 packets/s and the delay of each forward link is 0.1s. Let the router's buffer size be $B = 100$ packets. What is the throughput of the standard TCP connection in comparison to the TCP* connections?

(Circle the BEST answer)

A. The standard TCP connection's throughput is **Larger** than each of the TCP* connections.

Answer: Larger. The standard TCP obtains a higher throughput because the buffer is B=100, which allows the TCP to open up its cwnd to a much bigger size than the window of TCP*.

B. The standard TCP connection's throughput is **Smaller** than each of the TCP* connections.

C. The standard TCP connection's throughput is **Equal** to each of the TCP* connections.

VII Therac 25

In the Therac-25 (as described in "Medical Devices: The Therac-25" by Leveson), a race condition between the Keyboard thread, the Round-Table thread, and the Beam thread caused a sequence of terrible accidents. In the design for the Threac-25b, it was suggested that a race prevention mechanism be added to the software so as to provide mutually exclusive operation of the various threads. The locking code for the proposed mechanism is as follows:

```
int turn;
boolean busy = false;

void lock() {
  int me = ThreadID.get();  // 1 = keyboard, 2 = Round-Table, and 3 = Beam
  do {
    do {
      turn = me;
    } while (busy);
    busy = true;
  } while (turn != me && busy);
}

void unlock() {
  busy = false;
}
```

In a `do-while` loop, the body of the loop is executed first, then the condition is evaluated. If the condition is true, then the loop is repeated.

11. [8 points]: What properties does this code have, assuming that the rest of the system uses `lock()` and `unlock()` correctly:

Initials:

(Circle True or False for each choice.)

A. True / False This algorithm provides mutual exclusion when all three threads access the lock.

Answer: False. Consider the following counterexample: Threads 1, 2 and 3 all get to the `busy = true` line, where 3 was last to write `turn = me`. Next, thread 3 runs past the outer `while` test line into critical section because `turn = 3`. Next, threads 1 and 2 set `busy = true` and get to the test line. Now thread 3 leaves critical section setting `busy` to false. Threads 1 and 2 now enter the critical section together.

B. True / False This algorithm may result in a deadlock if only two threads access the lock (*e.g.,* the Round-Table thread and the Beam thread).

Answer: True. Thread 2 writes `turn = me` after thread 1 and both are at the `while(busy)` test. Thread 1 passes `while(busy)` and sets `busy = true`. Now thread 2 cannot get past the `while(busy)` test, and thread 1 cannot pass the `while (turn != me && busy)` test.

End of Quiz I

Please double check that you wrote your name on the front of the quiz,
and circled your recitation section number.

Initials: