Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033 Computer Systems Engineering: Spring 2016

Quiz I

There are **15 questions** and **?? pages** in this quiz booklet. Answer each question according to the instructions given. You have two hours to answer the questions.

- The questions are organized (roughly) by topic. They are not ordered by difficulty nor by the number of points they are worth.
- **If you find a question ambiguous, write down any assumptions you make.** Be neat and legible.
- Some students will be taking a make-up exam at a later date. **Do not** discuss this quiz with anyone who has not already taken it.
- You may use the back of the pages for scratch work, but **do not** write anything on the back that you want graded. We will not look at the backs of the pages.
- Write your name in the space below. Write your initials at the bottom of each page.

This is an open-book, open-notes, open-laptop quiz, but you may **NOT** use your laptop, or any other device, for communication with any other entity (person or machine).

Turn all network devices, including your phone, off.

CIRCLE your recitation section:

10:00	1. Matei/Cathie	7. Mark/CK	8. Michael/Pratheek	15. Karen/Jacqui
11:00	9. Matei/Cathie	10. Mark/CK	11. Michael/Pratheek	16. Karen/Jacqui
12:00	12. Asaf/Anubhav			
1:00	2. Mike/Steven	5. Peter/Sumit	13. Mark/Jodie	14. Asaf/Anubhav
2:00	3. Mark/Jodie	4. Peter/Sumit	6. Mike/Steven	

Name: SOLUTIONS

I Virtual Memory

1. [9 points]: Operating system X uses a standard page-table scheme, where the first N bits ($1 \leq N < 32$) of a 32-bit address are interpreted as the virtual page number, and the last $32 - N$ bits are interpreted as the offset into the page.

Operating system Y uses a hierarchical page-table scheme. The first M bits ($M \geq 1$) are interpreted as the outer page number, the next P bits ($P \geq 1$) are interpreted as the inner page number, and the last $32 - M - P$ bits are interpreted as the offset into the inner page table ($M + P < 32$).

Answer the following questions for **each** operating system.

A. What is the maximum number of virtual addresses available to any process?

In OS X: 2^{32}

In OS Y: 2^{32}

Regardless of page-table technique, there are 2^{32} possible virtual addresses available; virtual addressing virtualizes the physical address space.

B. A process attempts to read a value in virtual address v. The process has read-access to this data (i.e., it has permission to read the data stored in v). What is the maximum number of page faults that could be thrown as a result of this access?

In OS X: 1

In OS Y: 2

In X, 1 page fault will be thrown if the physical page isn't present in memory (i.e., if the present bit is zero). In Y, page faults are thrown for the same reason, but there are two changes for that to happen: if the outer page isn't present *and* if the inner page isn't present.

C. Assume that each page-table entry (in any of the tables) is K bits. What is the maximum amount of memory that will be allocated **for page tables** after a single access?

In OS X: $K \cdot 2^N$

In OS Y: $K \cdot (2^M + 2^P)$

Page tables must be allocated all at once or not at all. In X, the page table contains 2^N K-bit entries. In Y, the outer page table contains 2^M K-bit entries, and a single inner page table contains 2^P such entries. At most one inner table will be allocated for a single access.

Initials:

2. [6 points]: Modern hardware has built-in support for virtualization (i.e., it has functionality that can support a VMM that manages multiple guest OSes). Which of the following is true about such hardware?

A. True / False There is a special VMM operating mode in addition to user and kernel operating modes.

B. True / False Such hardware supports both monolithic and microkernels.

C. True / False Guest OSes' page tables map directly from guest virtual addresses to host physical addresses.

(a) True. The VMM requires a special level of privilege.

(b) True. Virtualization allows multiple OSes to run on the same physical hardware; there is no restriction on whether the OS is monolithic or a microkernel.

(c) False. The Guest OS does not know the host physical addresses (as far as the guest is concerned, it's the only OS running). Guest OSes always have page tables that map from guest virtual to guest physical. In modern hardware, the machine itself can do the translation from guest virtual to guest physical to host physical.

II Threads and Processes

3. [9 points]: Ben Bitdiddle is attempting to implement bounded buffer `send` and `receive`. Below is the pseudocode of his current implementation:

```
// bb is a structure containing:
//    buf - the buffer itself
//    in - the number of messages that have been placed in buf
//    out - the number of messages that have been read from buf
//    lock - a lock which must be held to read from or write to buf

// N is a global variable specifying the capacity of the buffer

send(bb, message):
    acquire(bb.lock)

    while (bb.in - bb.out) == N:
        pass

    bb.buf[bb.in % N] = message
    bb.in += 1

    release(bb.lock)
    return
```

```
receive(bb):
    acquire(bb.lock)

    while (bb.in - bb.out) == 0:
        pass

    m = bb.buf[bb.out % N]
    bb.out += 1

    release(bb.lock)
    return m
```

A. Describe a scenario that could lead to deadlock in Ben's code, specifying the **precise sequence** of events that would lead to deadlock.

Write **no more than four sentences** (or bullet-points). If you write more than four sentences (or bullet-points), we reserve the right to only read the first four. We are interested in the clarity of your explanation as much as its content.

1. T_1 calls send and acquires the lock.
2. T_1 finds the buffer full, so spins
3. T_2 calls receive to remove a message (which would allow T_1 to complete its send), but cannot because T_1 holds the lock.

The opposite is also true: a call to receive with an empty buffer will result in deadlock.

Note that it is **not enough** to say that T_1 spins in the while loop. Deadlocks requires two threads to be waiting on each other.

Initials:

B. Ben claims that adding a single call to `yield()`—and no other code—in each of the two methods will fix his deadlock issue. Is he correct? Explain (again, in no more than four sentences).

He is not. yield() does not release locks, it only gives up the processor.

C. **True / False** All correct multi-threaded code—including Ben's, once it is correct—must call either `yield()` or `wait()`; otherwise, one thread may completely halt the process of other threads.

False. The OS will preempt programs even if they do not explicitly call yield.

4. [4 points]: Consider the following commands executed, in order, from a single UNIX shell, as described in "The UNIX Time-Sharing System" by Ritchie and Thompson. $ is the command prompt.

```
$ command1
$ command2 &
$ command3
```

The shell is ready to accept the next command—`command4`—from the user. Answer the following true/false questions.

(a) **True / False** Since the shell is ready for `command4`, all previous commands (1–3) have completed.

(b) **True / False** At the start of its execution, `command1` has zero open file descriptors.

(a) False. `command2` was run in the background, and so may still be running (the shell does not wait after it executes a background command).

(b) False. All commands executed from the shell start with three open file descriptors: 0, 1, and 2 (stdin, stdout, stderr).

5. [6 points]: You run Eraser—as described in "Eraser: A Dynamic Data Race Detector for Multi-threaded Programs" by Savage et al.—on a program that you wrote. It issues no warnings. Answer the following true/false questions about the program.

(a) **True / False** At the end of execution, the lockset of each variable was nonempty, but may have been empty at some point during execution.

(b) **True / False** Because Eraser issued no warnings, it must be the case that no threads competed for access to a single variable in your program.

(c) **True / False** Because Eraser issued no warnings, deadlock will not occur in your program.

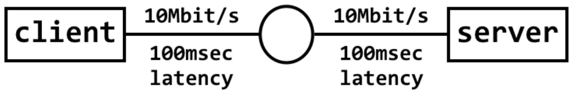
(d) **True / False** Because Eraser issued no warnings, data races will not occur in your program.

All false. For (a), locksets can only shrink; they cannot grow. For (b), you can have threads working on shared memory (specifically that referenced by variables in code) regardless of whether your code has data races or not. For (c) and (d), Eraser cannot guarantee that it discovers all instances of deadlock nor all instances of data races.

Initials:

III Performance

6. [8 points]: Consider a client and a server connected via a single switch. Each link in the network has a capacity of 10 Mbit/sec, and a one-way latency of 100 msec. The processing time at the switch is negligible. The **maximum** packet size in the network is 1000 bytes.



The server hosts a table that contains <`key, value`> pairs. The client sends requests to the server that contain a `key`; the server responds with the corresponding `value`. Each request packet has a 100-byte header and 100 bytes of data (the key). Responses have the same format (100-byte header with 100 bytes of data).

You have three possible techniques for improving the system:

(a) **Parallelism:** Add an additional network link, with 10 Mbit/s throughput and 100 msec latency, between the client and the switch. Have the client send requests out on both links (in parallel) instead of a single link.

(b) **Caching:** Cache server responses at the switch. If the switch has a cached <`key, value`> pair, it will return the value to the client immediately instead of forwarding the request to the server. Assume that the time it takes for the switch to look up a key in its cache is negligible.

(c) **Batching:** Have the client batch requests before sending them over the network: each packet to the server would include multiple keys, instead of one key per packet. Assume that the server can parse and process multi-key packets.

A. Of the three techniques, which (if any) will most significantly improve latency?

(a) Parallelism
(b) Caching   Solution: (b)
(c) Batching
(d) None of the techniques would significantly improve latency

B. Of the three techniques, which (if any) will most significantly improve throughput (number of requests per second)? Assume that the client is generating enough requests to take advantage of any improvements in throughput.

(a) Parallelism
(b) Caching
(c) Batching   Solution: (c)
(d) None of the techniques would significantly improve throughput

Caching results in some requests being served in 200 msecs rather than 400 msecs. None of the other techniques affect latency. With batching, with 1000 bytes we can now send nine requests (1 header + 9 requests) instead of five (5 headers + 5 requests). Hence the throughput improvement. Note that parallelism does not improve performance in this case; we added a single extra link between the client and switch, but *not* another between the switch and sender. Thus the bottleneck is not changed; we can't get anymore requests to or from the sender than before.

Initials:

7. [6 points]: Answer the following true/false questions about MapReduce, as described in "MapReduce: Simplified Data Processing on Large Clusters" by Dean and Ghemawat.

A. True / False Intermediate key-value pairs from `map` tasks must be written to GFS to protect against worker failures.

B. True / False MapReduce takes data locality into account when scheduling `map` tasks, but not when scheduling `reduce` tasks.

C. True / False MapReduce starts backup tasks for all tasks in order to quickly deal with stragglers.

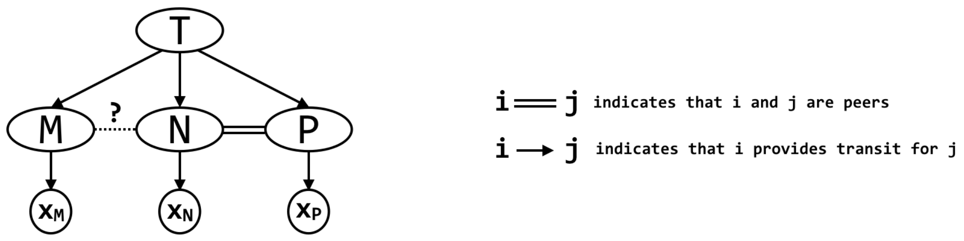
(a) False. Intermediate data is lost if workers fail.

(b) True. Section 3.4 of the paper describes using locality to improve map tasks. In general, it's impossible to do this for reduce tasks, because reduce tasks need data from multiple mappers.

(a) False. MapReduce starts backup tasks for all in-progress tasks at the end of its execution.

Initials:

IV Routing, Addressing, and Naming


T
M
N
P
?
x_M
x_N
x_P
i ═ j indicates that i and j are peers
i → j indicates that i provides transit for j

8. [9 points]: Consider the above AS topology. These ASes advertise routes according to the standard BGP rules discussed in class. Assume that there are no failures and no loss in the network and that all routes have converged.

All of the inter-AS relationships are known to you, except the relationship between M and N. In an attempt to infer that relationship, you observe some of the traffic flowing in the network.

For each of the following, indicate which relationships between M and N are consistent with the traffic that you observed. Consider each scenario independently.

A. Traffic between x_M and x_P traverses the following ASes: $x_M \to M \to T \to P \to x_P$.

(a) **Consistent / Not Consistent** M and N are in a peering relationship.

(b) **Consistent / Not Consistent** M provides transit for N.

(c) **Consistent / Not Consistent** N provides transit for M.

All consistent. In (a) and (b), N will not expose its route to x_P via P, so traffic from M must go via T. In (c) it will, but M may choose the given route instead.

B. Traffic between x_M and x_P traverses the following ASes: $x_M \to M \to N \to P \to x_P$.

(a) **Consistent / Not Consistent** M and N are in a peering relationship.

(b) **Consistent / Not Consistent** M provides transit for N.

(c) **Consistent / Not Consistent** N provides transit for M.

(c) is consistent, same explanation as above except that this time M chooses the alternate route.

C. Traffic between x_M and x_N traverses the following ASes: $x_M \to M \to N \to x_N$.

(a) **Consistent / Not Consistent** M and N are in a peering relationship.

(b) **Consistent / Not Consistent** M provides transit for N.

(c) **Consistent / Not Consistent** N provides transit for M.

All consistent.

Initials:

9. [4 points]: Answer the following true/false questions about RON, as described in "Resilient Overlay Networks" by Andersen et al.

A. True / False To improve scalability, each RON node only probes other RON nodes that are geographically close.

B. True / False Adding more nodes into a RON will cause more overhead but may expose more possible paths in the network.

(a) False. Each RON node probes every other RON node.

(b) True. The additional probing will add overhead, but we'll potentially see more paths.

10. [6 points]: Two DNS clients, A and B, make a DNS request to resolve `google.com` within milliseconds of each other. However, A and B receive different responses. For each of the following scenarios, circle "True" if that scenario could have caused the change.

(a) **True / False** The two requests were resolved by the same nameserver, but the mapping changed in between the two requests being processed.

(b) **True / False** `google.com` is using a CDN such as Akamai, which changes the value of the mapping depending on (among other things) the geographic location of the clients.

Both True. In (a), since mappings are allowed to change, A and B could have seen different mappings (some studnts claimed that the change wouldn't happen so quickly, but note that whenever you write a piece of data, there is some instant when the new value is visible rather than the old value; a read before that instant will see the old value, a read after that instant will see the new value). (b) is also possible; that is in fact how Akamai redirects clients to different edge servers.

V Transport

11. [9 points]: A TCP sender S sends a window of 9 packets—`101` through `109`—with .1 msec in between each packet (i.e., packet `101` goes out at time t, packet `102` at time $t + .1$, etc.). Packets `104`, `105`, and `106` are lost; all other packets arrive at the receiver successfully. The round-trip time between S and the receiver is 1 second, and S's TCP timeout is set to 2 seconds.

Assume that there is no additional loss on the network. I.e., no ACKs from the receiver are lost, and no retransmissions from the sender are lost.

A. Which packets will be retransmitted due to TCP's fast-retransmit mechanism?

(a) `104`
(b) `105`
(c) `106`
(d) None of the above.

B. Which packets will be retransmitted due to a TCP timeout?

(a) `104`
(b) `105`
(c) `106`
(d) None of the above.

C. Assume that S currently has a window size of 9. Let W_f be the size of S's window immediately after a retransmission due to fast-retransmit, and let W_s be the size of S's window immediately after a retransmission due to a timeout. Which of the following is true?

(a) $W_f > W_s$
(b) $W_f < W_s$
(c) $W_f = W_s$
(d) There is not enough information to decide.

Note: We actually meant .1 sec between consecutive packets, not .1 msec, but the solution below does not change even if you use .1 msec.

104 is transmitted via fast-retransmit, 105 and 106 via timeout. Packet 104 is transmitted at time $t + .4$–let's just say .4 for ease of use. The third dup-ack for 104 will be the ACK in response to packet 109; that packet was sent at time .9, and so the ACK returns at 1.9: 1.5 seconds after 104 was sent. Thus, a fast-retransmit is triggered before the timeout would fire (at time 2.4).

For 105 and 106, we don't need to do fancy calculations. The soonest we'll get any ACK back for 104—nevermind three dup-acks—is $1.9 + 1 = 2.9$, which is more than two seconds after 105 and 106 were sent. Thus, a timeout.

For part C, $W_s = 1$, $W_f = 4$ (or 5 if you round up). After slow-start, TCP drops its window down to 1.

Initials:

12. [6 points]: Consider the following four queue management schemes: DropTail, ECN, RED, and DCTCP. In each of the following questions, circle the name(s) of the queue management scheme(s) for which the statement is **true** (if it's not true for any of the schemes, circle "None").

(Note: we use "RED" to refer to the scheme that drops packets rather than marks packets. This is the convention that was used in lecture, but not the convention used in the DCTCP paper.)

A. Requires a measurement of the average queue size.

DropTail ECN RED DCTCP None

ECN, RED. DCTCP uses instantaneous queue size, and DropTail does not measure the queue size at all.

B. Can react to congestion before queues are full.

DropTail ECN RED DCTCP None

ECN, RED, DCTCP. Reacting to congestion before queues are full is, in large part, the point of queue management schemes other than DropTail.

C. Requires **no** changes to a TCP sender's behavior.

DropTail ECN RED DCTCP None

DropTail, RED. TCP senders react to drops; since RED and DropTail drop packets, there's no need to change sender behavior. ECN and DCTCP, on the other hand, mark packets; senders need to learn to react to marks in addition to drops.

13. [8 points]: Two types of flows are sending data through a single switch S to a single destination:

1. Small, latency-sensitive flows, which send at most 20 packets per second total (i.e., all latency-sensitive flows combined send no more than 20 packets per second in aggregate).
2. Large, throughput-sensitive flows. There is no limit to the amount of data that the large flows may send.

Currently, both flows send their data through a single queue in S. This queue is the bottleneck in the network; capacity is not an issue after the packets leave S. Packets will also not be lost or corrupted after they leave S. There are no failures in the network (e.g., switches dying or links going down).

The processing time for each packet in the queue is 1 msec, and the one-way latency to the destination is 100 msec. This means that, if there are x packets already in the queue, it will take 101+x msec for a packet to arrive at its final destination. All packets in the network are the same size.

For each of the following queueing schemes, circle "True" if it allows you to guarantee that every **latency-sensitive** packet arrives at its destination within 150 msec. Assume that all of the proposed queues can hold up to 1000 packets.

(a) **True / False** Use a single queue. When a latency-sensitive packet arrives, if there is room in the queue, place the packet at the **head** of the queue, instead of the tail.

(b) **True / False** Use two queues: Q_1 for latency-sensitive traffic, and Q_2 for throughput-sensitive traffic. Service the queues in a round-robin fashion, sending a single packet from each queue at a time. If there are no packets in a queue at its service time, simply move to the next queue.

(c) **True / False** Use two queues: Q_1 for latency-sensitive traffic, and Q_2 for throughput-sensitive traffic. Service the queues in a round-robin fashion, but send **all** packets from Q_1 when servicing it, and only a single packet from Q_2 when servicing it. If there are no packets in a queue at its service time, simply move to the next queue.

(a) False, because the queue could be full. Given finite queues, you need isolation between the two types of traffic.

(b) and (c) are both True. We're guaranteed no more than 20 latency-sensitive packets per second. In either (b) or (c), it'll take no more than 2 msec for us to process a latency-sensitive packet (its time in the queue plus the additional time we might need to process a throughput-sensitive packet in its queue), which means a latency of no more than 140 msecs.

Initials:

VI Newer Internet Technologies

14. [4 points]: Akamai is considering a new hybrid technology. When a client makes a request for a piece of content, they are directed to an edge server as usual. This edge server then points them to a BitTorrent swarm (e.g., by providing the client with a `.torrent` file). The client downloads half of the content from the edge server, and simultaneously downloads the other half of the content from the BitTorrent swarm. Akamai believes that the selling point of this technology is that it improves client performance.

Write **one sentence** describing a scenario where this technology **does not** improve client performance.

In essence: Fast server, slow swarm. If the P2P swarm is, say, 10x slower than the server, then even as the downloads proceed in parallel, this will be slower than just using the server. (There are other problems as well: if the client has a small uplink, its BitTorrent download will proceed slowly due to BitTorrent's incentive structure.)

Note, though, that this design is not all bad; it does decrease load on the server.

15. [6 points]: Consider two clients, A and B, attempting to send data to an access point, X. A and B can hear X perfectly, and X can hear A and B perfectly. However, A and B cannot hear each other.

Which of the following will prevent A and B's **data** packets from colliding? Circle all that apply. Ignore collisions from control-traffic—such as RTS/CTS messages—when applicable.

(a) Use CSMA/CA as the MAC protocol, but turn off carrier sense: the mechanism by which A and B listen to the channel before sending.

(b) Use CSMA/CA with the RTS/CTS mechanism. A "CTS" message for a client clears that client for sending in the next timeslot. Clients will **not** send in the next timeslot if they overhear a CTS message meant for someone else.

(c) Use the "round-robin" MAC protocol—A sends every other timeslot, B sends on the opposite timeslots—with X dictating the timeslot length, start of the timeslots, etc. (This "round-robin" MAC protocol is also known as TDMA.)

(d) None of the above.

(a) False. CSMA/CA has the hidden terminal problem; turning off carrier sense will make things work.

(b) and (c) are both true. To solve the hidden terminal problem, you need the AP to provide centralized control (i.e., to tell the senders when it's okay to send). This happens in RTS/CTS—which was designed specifically for this purpose—but also, effectively, in TDMA. Remember that TDMA is a perfect MAC protocol: zero collisions.