

logo.pdf

*Department of Electrical Engineering and Computer Science*

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

## 6.033 Computer Systems Engineering: Spring 2018

# Quiz I

There are **15 questions** and **11 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. Mark        3. Muriel    5. Karen

**11:00**    2. Mark        4. Muriel    6. Karen

**12:00**    7. Howard     9. Sam

**1:00**     8. Howard    10. Sam     11. Mark    13. Peter    15. Tim    17. Adam

**2:00**    12. Mark     14. Peter    16. Tim     18. Adam

**Name:**

## I Naming

1. [8 points]: Mark is designing an operating system, and is considering three different virtual-addressing schemes. In all schemes, his virtual addresses are 32 bits:

- **Vanilla Scheme:** “Standard” paging, where addresses are made up of 16-bit page numbers and 16-bit offsets.
- **Chocolate Scheme:** A one-level hierarchy, where addresses are made up of an 8-bit outer-page number, an 8-bit inner-page number, and a 16-bit offset.
- **Strawberry Scheme:** A two-level hierarchy, where addresses are made up of an 8-bit outer-page number, an 8-bit inner-page number, an 8-bit second-inner-page number, and an 8-bit offset.

For each of the following, circle all that apply.

- A. Which scheme(s) results in the largest virtual address space?
- (a) Vanilla Scheme
  - (b) Chocolate Scheme
  - (c) Strawberry Scheme
- B. Which scheme(s) has the largest pages?
- (a) Vanilla Scheme
  - (b) Chocolate Scheme
  - (c) Strawberry Scheme
- C. Assume that initially, **no** page tables are allocated. Then, a single virtual address translation takes place (no other translations take place). Which scheme(s) will result in the smallest amount of memory dedicated to page tables? Assume that individual page table entries are the same size in each scheme.
- (a) Vanilla Scheme
  - (b) Chocolate Scheme
  - (c) Strawberry Scheme

2. [6 points]: Circle True or False for each of the following questions about DNS.

- (a) **True / False** A DNS name (e.g., `mit.edu`) may be associated with multiple IP addresses, but each IP address has to be associated with a single DNS name.
- (b) **True / False** DNS caching reduces the time to resolve an IP address, but does not reduce DNS traffic on the Internet.
- (c) **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.

Initials:

## II Operating Systems and Performance

3. [10 points]: A guest OS kernel executes a single process, using the page table shown below on the left (which maps guest-virtual pages to guest-physical pages); this page table is stored in guest-physical page 4. In this table, the 'P' column indicates whether the page in question has been loaded into memory; the 'W' column indicates whether the page is writeable; and the 'U' column indicates whether the page is accessible to non-kernel processes.

The virtual machine monitor (VMM) uses the table shown below on the right to translate the guest-physical pages to host-physical pages.

guest virtual	guest physical	P	W	U
0	5	1	1	1
1	9	0	1	1
2	11	1	0	1
3	4	1	1	0
4	7	1	1	0

Guest OS's page table, stored in guest-physical page 4.

guest physical	host physical
4	2
5	6
6	4
7	3
11	8

VMM's table

To run the VM, the VMM needs to load a *shadow* page table, which maps guest-virtual pages to host-physical pages. The content of that table may differ depending on whether the guest OS has its U/K bit set to U (user) or K (kernel).

Fill in the shadow page table below, which the VMM should load when the guest OS has its U/K bit set to **K**. You can use a dash (—) to indicate values that don't matter or can't be discerned. Assume that the VMM supports virtualization using trap-and-emulate (rather than, e.g., binary translation), and that the VMM runs the guest kernel in user mode. The VMM updates the shadow page table state by tracing writes to the guest OS's page table.

guest virtual	host physical	P	W	U
0				
1				
2				
3				
4				

Initials:

4. [6 points]: Consider the following bounded buffer code (send and receive), assuming the variables `bb.in` and `bb.out` are 64 bits, never overflow, can be read and written atomically, and neither the compiler nor hardware will ever reorder instructions:

```

send(bb, m):
    // each invocation of send has
    // its own local variables:
    // my_send_index (64-bit int)
    while True:
        acquire(bb.lock)
        if bb.in - bb.out < N:
            my_send_index = bb.in
            bb.in = bb.in + 1
            release(bb.lock)
            bb.buf[my_send_index mod N] = m
            return
        release(bb.lock)

receive(bb):
    // each invocation of receive
    // has its own local variables:
    // my_rec_index (64-bit int)
    // m (message)
    while True:
        acquire(bb.lock)
        if bb.in > bb.out:
            my_rec_index = bb.out
            bb.out = bb.out + 1
            release(bb.lock)
            acquire(bb.rec_lock)
            m = bb.buf[my_rec_index mod N]
            release(bb.rec_lock)
            return m
        release(bb.lock)

```

Circle True or False for each of the following. Recall that a correct bounded-buffer implementation does not allow a sender to overwrite unread messages, and does not allow a receiver to read uninitialized messages.

- (a) **True / False** The code is correct if there is one sender and one receiver executing at the same time.
- (b) **True / False** The code is correct if there is one sender and many receivers executing at the same time.
- (c) **True / False** The code is correct if there are many senders and one receiver executing at the same time.
- (d) **True / False** The code is correct if there are many senders and many receivers executing at the same time.

**Initials:**

5. [6 points]: Consider the following snippet of code, as part of a larger system in which the variable `x` is modified. The locks `L1` and `L2` are not used anywhere else in the system.

```
Lock L1;
Lock L2;
int x;

function foo() {
    acquire(L1);
    print(x);
    release(L1);
}

function bar(int v) {
    acquire(L2);
    if (v == 0) {
        print(x);
    }
    release(L2);
}
```

The functions `foo()` and `bar()` are executed from separate threads, but Eraser never flags an error. Which of the following reasons might explain this? Circle **all** that apply.

- (a) `foo()` and `bar()` both execute, but never at the same time, so no race condition actually occurs.
- (b) `foo()` and `bar()` run concurrently, but `bar()` is always called with 1 as an argument.
- (c) Every time `foo()` or `bar()` is called, an additional lock `L3` is also held.
- (d) The value of `x` is changed for the last time before either `foo()` or `bar()` are called for the first time.

6. [6 points]: Circle True or False for each of the following questions about MapReduce.

- (a) **True / False** If there are  $m$  map tasks, using more than  $m$  workers in the map phase may still improve performance beyond that achieved with  $m$  workers.
- (b) **True / False** To achieve locality, map workers always execute on the same machine as the input data that they consume.
- (c) **True / False** Intermediate data passed between the map workers and reduce workers is stored in the Google File System (GFS).

Initials:

7. [8 points]: Consider the following pseudocode. The starting process (P1) forks, creating a child process (P2), which also calls fork, creating a child process of its own (P3).

```
int main()
{
    int f1 = open("file1.txt")
    int f2 = 0;
    int f3 = open("file3.txt")
    int f4 = open("file4.txt")

    // P1
    if (fork() != 0) {
        f2 = open("file2.txt");
    }

    else {
        // P2
        close(f3);
        if (fork() != 0) {
            close(f4);

        } else {
            // P3
        }
    }
}
```

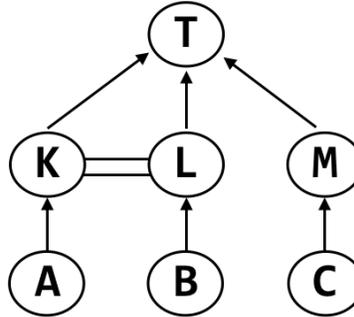
Circle True or False for each of the following to indicate what type of communication is available between P1 and P3.

- (a) **True / False** P1 and P3 can communicate via writing to/reading from the file descriptor f1.
- (b) **True / False** P1 and P3 can communicate via writing to/reading from the file descriptor f2.
- (c) **True / False** P1 and P3 can communicate via writing to/reading from the file descriptor f3.
- (d) **True / False** P1 and P3 can communicate via writing to/reading from the file descriptor f4.

**Initials:**

### III Routing

8. [6 points]: Consider the following AS graph. Arrows in the graph indicate the relationships between ASes: An arrow from X to Y ( $X \rightarrow Y$ ) means that X is a **customer** of Y. Two lines between X and Y ( $X = Y$ ) means that X and Y are **peers**.



There are two distinct paths between AS *A* and AS *C*:

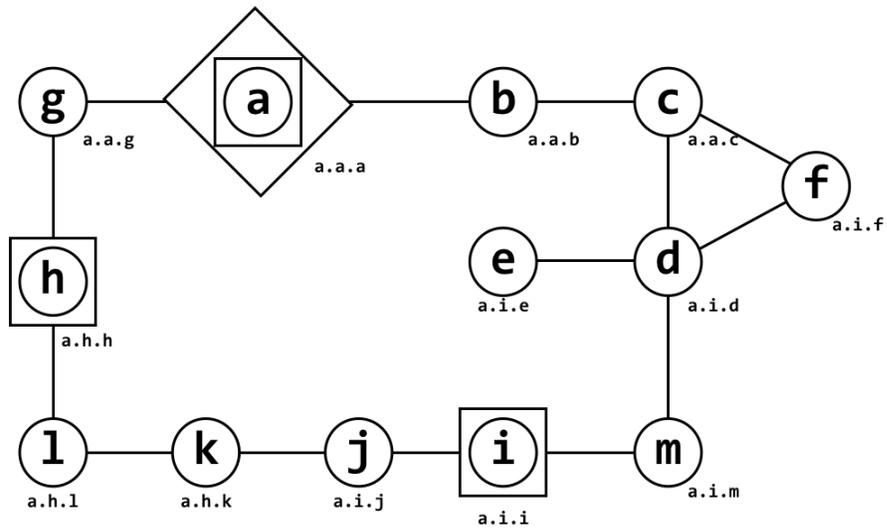
- **Path 1:**  $A \rightarrow K \rightarrow T \rightarrow M \rightarrow C$
- **Path 2:**  $A \rightarrow K \rightarrow L \rightarrow T \rightarrow M \rightarrow C$

Suppose that two RON nodes exist: one in AS *A*, and one in AS *C*. Is it possible that, via RON, a machine in AS *A* could route data to a machine in AS *C* via Path 2? Circle the **best** answer.

- (a) Yes, but RON is not needed; Path 2 is the path that BGP exposes between *A* and *C*.
- (b) Yes, and no additional RON nodes are needed.
- (c) Yes, but there must exist a RON node in AS *L*.
- (d) Yes, but there must exist a RON node in AS *L* and a RON node in AS *K*.
- (e) No.

Initials:

9. [9 points]: Consider the following network, which uses landmark routing.



Level 0, radius 2     
  Level 1, radius 3     
  Level 2, radius 8

A. Which of the following routers know about  $LM_1[i]$ ? Circle **all** that apply.

- (a) *a*
- (b) *e*
- (c) *h*
- (d) *m*

B. Which of the following routers know about  $LM_2[a]$ ? Circle **all** that apply.

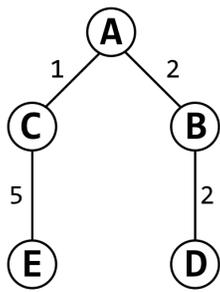
- (a) *e*
- (b) *h*
- (c) *i*
- (d) *m*

C. What path will packets from *a.i.d.* to *a.h.l.* take? Circle the **best** answer.

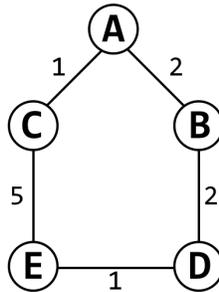
- (a)  $d \rightarrow m \rightarrow i \rightarrow j \rightarrow k \rightarrow l$
- (b)  $d \rightarrow c \rightarrow b \rightarrow a \rightarrow g \rightarrow h \rightarrow l$
- (c)  $d \rightarrow f \rightarrow c \rightarrow b \rightarrow a \rightarrow g \rightarrow h \rightarrow l$
- (d) None of the above

Initials:

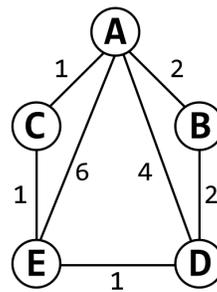
10. [8 points]: Consider the four topologies below. The number near each link is the link's cost.



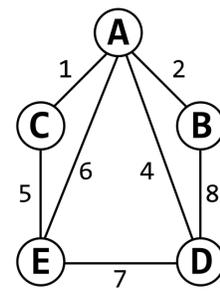
Topology I



Topology II



Topology III



Topology IV

For all parts of this problem, assume that the routing protocol has converged, and that there are no losses or failures in the network.

A. Assume that the network is running a **distance-vector** routing protocol. Node *A* sends the following advertisement.

$$[(A, 0), (B, 2), (C, 1), (D, 4), (E, 6)]$$

Which of the topologies are consistent with the above advertisement? Circle **all** that apply.

- (a) Topology I
- (b) Topology II
- (c) Topology III
- (d) Topology IV
- (e) None

B. Assume that the network is running a **link-state** routing protocol. Node *A* sends the following advertisement.

$$[542 : (A, 0), (B, 2), (C, 1), (D, 4), (E, 6)]$$

Which of the topologies are consistent with the above advertisement? Circle **all** that apply.

- (a) Topology I
- (b) Topology II
- (c) Topology III
- (d) Topology IV
- (e) None

Initials:

## IV Transport, Queues, and New Architectures

**11. [6 points]:** Consider a reliable sender and receiver. The sender sends a stream of new data packets—i.e., none of the packets are retransmissions—and in response receives the following (cumulative) ACKs from the receiver:

1 2 3 4 4 4 4 4 4

Now, suppose that the sender times out and retransmits the first unacknowledged data packet. When the receiver gets that retransmitted data packet, what can you say about the ACK,  $a$ , that it sends? Circle the **best** answer.

- (a)  $a = 5$
- (b)  $a \geq 5$
- (c)  $5 \leq a \leq 11$
- (d)  $a = 11$
- (e)  $a \leq 11$

**12. [4 points]:** Consider two senders,  $S_1$  and  $S_2$ , sending to two receivers,  $R_1$  and  $R_2$  respectively. The paths from  $S_1$  to  $R_1$  and from  $S_2$  to  $R_2$  share a bottleneck link. The round-trip-time between  $S_1$  and  $R_1$  is  $RTT_1$ ; between  $S_2$  and  $R_2$  it is  $RTT_2$ .

$S_1$  and  $S_2$  are both using TCP's AIMD congestion control; however, only  $S_2$  is using the slow-start optimization. For the purposes of this problem, you can assume that the queue at the bottleneck link is infinite; packets will not be dropped.

Assume that  $S_1$  and  $S_2$  start sending data at the exact same time, each with an initial window size of 1. Both  $S_1$  and  $S_2$  each need to send 30 packets, all of the same size. What must be true about  $RTT_1$  and  $RTT_2$  in order for  $S_1$ 's transfer to finish before  $S_2$ 's? Circle the **best** answer.

- (a)  $RTT_1 < RTT_2$
- (b)  $RTT_1 < \frac{5}{8}RTT_2$
- (c)  $RTT_1 < \frac{1}{5}RTT_2$
- (d)  $S_1$ 's transfer will complete before  $S_2$ 's regardless of the values of  $RTT_1$  and  $RTT_2$

**Initials:**

**13. [4 points]:** Which of the following best describe DCTCP? (Circle the **best** answer.)

- (a) A DCTCP sender will decrease its window-size by half as soon as it sees a single packet mark.
- (b) A DCTCP sender will decrease its window-size by at most half as soon as it sees a single packet mark.
- (c) A DCTCP sender will decrease its window-size by at most half as soon as it sees a single packet drop.
- (d) None of the above.

**14. [8 points]:** Consider a switch with two queues.  $Q_1$  contains only 10-byte packets, and  $Q_2$  contains only 20-byte packets. You can assume that each queue has infinitely many packets in it.

Muriel is trying to decide whether to use weighted-round-robin (WRR) or deficit-round-robin (DRR) to schedule packets from these queues. Regardless of the protocol she chooses, a single packet (of any size) takes one second to send (and so 10 packets will take 10 seconds to send). If a round of the protocol proceeds without sending any packets, that round will also take 1 second.

Muriel has two requirements:

- **Efficiency:** Every second, a packet should be transmitted (i.e., there should be no “wasted” seconds).
- **Short-term Fairness:** In each two-second window,  $Q_2$  should transmit twice as many bits as  $Q_1$ .

Which of the schemes below satisfy both of Muriel’s requirements? Circle **all** that apply.

- (a) DRR with the quantum for  $Q_1$  set to 1 and the quantum for  $Q_2$  set to 2.
- (b) DRR with the quantum for  $Q_1$  set to 10 and the quantum for  $Q_2$  set to 20.
- (c) DRR with the quantum for  $Q_1$  set to 100 and the quantum for  $Q_2$  set to 200.
- (d) WRR with the weight for  $Q_1$  set to  $\frac{1}{3}$  and the weight for  $Q_2$  set to  $\frac{2}{3}$ .

**15. [5 points]:** For each of the characteristics below, decide whether it better describes a P2P network, such as BitTorrent, or a CDN, such as Akamai. If a characteristic describes both a P2P network and a CDN, circle “Both”.

- (a) **P2P / CDN / Both** Uses an overlay network
- (b) **P2P / CDN / Both** Better for streaming content
- (c) **P2P / CDN / Both** Uses DNS rewrites
- (d) **P2P / CDN / Both** Has a hierarchical organization
- (e) **P2P / CDN / Both** Tolerates individual machine failures

**Initials:**