Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.1800 Computer Systems Engineering: Spring 2023

Exam 1

There are **14 questions** and **11 pages** in this exam booklet. Answer each question according to the instructions given. You have two hours to answer the questions.

- The questions are organized loosely 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.
- You are not required to explain your answers unless we have explicitly asked for an explanation. You may include an explanation with any answer for possible partial credit.
- Some students will be taking a make-up exam at a later date. **Do not** discuss this exam with anyone who has not already taken it.
- Write your name and kerberos ID in the space below. Write your initials at the bottom of each page.

This is an open-book, open-notes, open-laptop exam, 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.

Name: SOLUTIONS

Kerberos ID: Solutions

1. [9 points]: Consider a DNS client C.

A. C is in the midst of resolving the hostname `www.a.b.c.com`. No machine caches responses and there is no recursive querying. As part of this process, C queries the `com.` nameserver, and receives the response `12.0.4.25`. What is `12.0.4.25` the address of? Circle the **best** answer.

(a) The machine that hosts `www.a.b.c.com`

(b) The machine that hosts `www.c.com`

[✓ (c)] The `c.com.` nameserver

(d) The root nameserver

If caching or recursion were involved, this answer could be different

root redirects to com. nameserver, which redirects to c.com. nameserver, etc.

Now, C sends all of its queries to the `com.` nameserver, which we'll call N. N performs recursive queries on behalf of C. No other nameserver in the network is currently performing recursive queries. There are no failures anywhere in the network, and every zone has distinct nameservers (i.e., there is no machine that serves as the nameserver for two separate zones).

B. Assume there is no caching anywhere in the network. On behalf of C, N resolves the hostname `web.eecs.mit.edu`. How many different nameservers will N have to contact as part of this process? Assume that N starts its querying with the root nameserver.

4

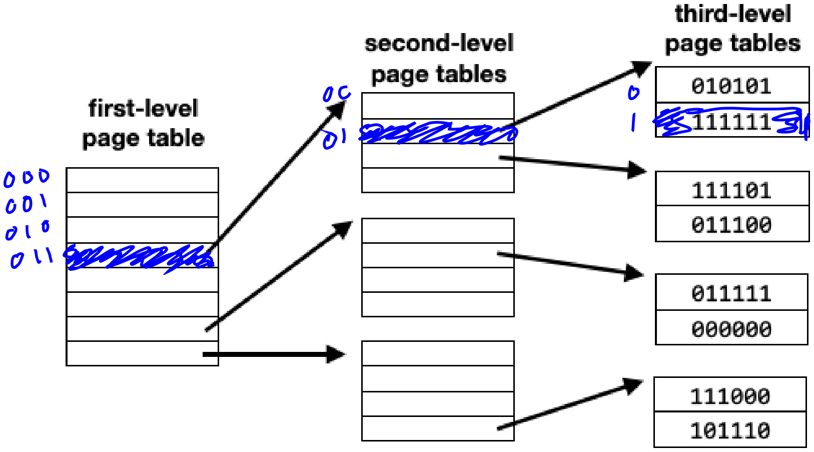
root, edu., mit.edu., eecs.mit.edu.

C. Now, assume that N enables caching. N again resolves C's query of `web.eecs.mit.edu`. Then C requests that N resolve the hostname `www.eecs.mit.edu`. How many different nameservers will N have to contact as part of this process? Assume that no cache entries have expired since N resolved `web.eecs.mit.edu`.

1

Just eecs.mit.edu. But not zero, because only the address for web.eecs.mit.edu is in the cache, not also www.eecs.mit.edu.

2. [10 points]: Deb's operating system uses **8-bit** addresses. It maps virtual addresses to physical addresses using multilevel page tables. The current layout of these page tables for a process p is shown below. The physical page numbers are given in **bits**, not hexadecimal. The most significant bit is first.



A. Based on the diagram above, how many bits are used to specify the offset of a memory address in Deb's operating system?

2

First-level table has 8 rows → 3 bits to specify
Second-level has 4 rows → 2 bits
Third-level has 2 rows → 1 bit $8-(3+2+1)=2$

B. Deb's OS uses the above page tables to translate a virtual memory address v into the **physical** address 11111111 (given in binary). Specify the eight bits of the **virtual** address v below. If there is not enough information to specify a particular bit, write a question mark in that bit's spot.

$v = 0\ 1\ 1\ 0\ 1\ 1\ \underbrace{1\ 1}_{\text{offset stays the same}}$

Follow the blue boxes above

Deb decides that multilevel page tables are too complicated and updates her operating system to use a single page table per process. It continues to use eight-bit addresses. Assume that Deb's OS uses b bits to specify the offset.

C. How many rows will the page table for one process have? Give your answer as a formula in terms of b.

2^{8-b}

b bits for offset ⇒
8-b bits left to specify row in page table ⇒
2^{8-b} rows

Initials:

3. [8 points]: Jay is using UNIX. His home directory contains exactly three files: `X.txt`, `XY.txt`, and `Y.txt`. Jay runs the following three commands inside of his home directory:

```
ls > output.txt
grep "X" < output.txt
rm output.txt
```

Searching for files in the home directory that contain the letter X

A. What is the output of the **second** command above?

X txt XY.txt

B. It is possible for Jay to perform the same function of the above commands in a single line. What is that line?

pipes replace output/input redirection

ls | grep "X"

C. Jay decides to upgrade his UNIX machine. He installs a version of UNIX that works exactly as described in the paper, but uses 1024-byte blocks instead of 512-byte blocks.

(a) [✓ **True**] / **False** Assuming that he is storing the exact same set of files, Jay's upgraded version of UNIX would most likely use fewer indirect blocks than a version that uses 512-byte blocks.

(b) [✓ **True**] / **False** With his upgraded version of UNIX, Jay's machine can now store larger files than he would be able to with 512-byte blocks (assuming the machine also has the disk space for larger files).

We use indirect blocks when a file grows beyond 10 blocks. Larger blocks means more files are ≤ 10 blocks in size.

Similarly, the size of a file is limited by how many blocks the inode can point to. That hasn't changed w/ Jay's upgrade, but larger blocks ⇒ larger files.

4. [10 points]: Below is some **incorrect** code for bounded buffer send and receive:

```
send(bb, message):
  acquire(bb.lock)
  while True:
    if bb.in - bb.out < N:
       bb.buf[bb.in mod N] <- message
       bb.in <- bb.in + 1
       release(bb.lock)
       return
```

```
receive(bb):
  acquire(bb.lock)
  while True:
    if bb.out < bb.in:
      message <- bb.buf[bb.out mod N]
      bb.out <- bb.out + 1
      release(bb.lock)
      return message
```

Multiple senders and receivers are sharing a bounded buffer bb that uses the above code. bb has space for three messages (i.e., N=3 in the code above). It is currently empty, and bb.in = bb.out = 0.

For each of the below scenarios, decide whether the scenario will **always** lead to deadlock, **may** lead to deadlock, or **will not** lead to deadlock. Consider each scenario independently. In each case, there are no other calls to send and receive beyond what is described in the scenario, and no machine failures. Circle the **best** answer for each scenario.

A. Thread t_1 calls receive and acquires bb.lock. Other threads call send or receive afterwards.

[✓ Always deadlocks]   May deadlock   Will not deadlock

bb is empty & t_1 is preventing any other thread from sending

B. Thread t_1 calls receive and is interrupted before it acquires bb.lock. Thread t_2 calls send after that.

Always deadlocks   [✓ May deadlock]   Will not deadlock

No deadlock: t_2 completes, then back to t_1

Deadlock: t_2 interrupted, then back to t_1, which acquires lock.

C. Thread t_1 calls send and the call completes successfully. Thread t_2 calls receive and the call completes successfully. Thread t_3 calls receive and acquires bb.lock. Thread t_4 calls send after that.

[✓ Always deadlocks]   May deadlock   Will not deadlock

This is essentially Scenario A again.

D. Thread t_1 calls send and the call completes successfully. Thread t_2 calls receive but is interrupted (you don't know at what point the interruption occurred). Thread t_3 calls send after that.

Always deadlocks   May deadlock   [✓ Will not deadlock]

t_2 will not get stuck in the while loop. No matter when we return to t_2, there is at least one message in the buffer.

5. [6 points]: Below is some **incorrect** code for bounded buffer send and receive (but different code from the previous question). We've included line numbers on the left.

```
 send(bb, message):                  receive(bb):
1:  acquire(bb.lock)                   acquire(bb.lock)
2:   while bb.in - bb.out >= N:         while bb.out >= bb.in:
3:      release(bb.lock)                   release(bb.lock)
4:      wait(bb.has_space)                 wait(bb.has_message)
5:      acquire(bb.lock)                   acquire(bb.lock)
6:   bb.buf[bb.in mod N] <- message     message <- bb.buf[bb.out mod N]
7:   bb.in <- bb.in + 1                 bb.out <- bb.out + 1
8:   release(bb.lock)                   release(bb.lock)
9:   notify(bb.has_message)             notify(bb.has_space)
10:  return                             return message
```

Both interrupted before wait(), so they wait get the notification

This code suffers from the "lost notify" problem. Which of the following sequences of events lead to a lost notify? Circle **all** that apply.

[✓ (a)] Thread t_1 calls send, finds the buffer full, and is interrupted between lines 3 and 4. Thread t_2 calls receive, and the call completes sucessfully.

[✓ (b)] Thread t_1 calls receive, finds the buffer empty, and is interrupted between lines 3 and 4. Thread t_2 calls send, and the call completes successfully. no lost notify

(c) Thread t_1 calls send, finds the buffer full, and calls wait (line 4). Thread t_2 calls receive.

(d) Thread t_1 calls send and the call completes successfully. Thread t_2 calls receive.

(e) None of the above

No thread waits here; no notification to lose

6. [5 points]: Hybrid race detection works by using lockset-based detection first, and then using happens-before detection on the set of races that lockset-based detection flags. It reports a race only if *both* forms of race detection flag the race. We'll refer to this scheme as the "original" hybrid race detection scheme.

Jordan proposes a different type of hybrid race detection: one that runs happens-before first, and then lockset-based detection on the set of races that happens-before flags. His scheme reports a race only if both forms of race detection flag the race.

A. In most cases, will Jordan's scheme detect **more** races than the original, the **same** set of races as the original, or **fewer** races than the original?

Same

{races found by lockset} ∩ {races found by happens-before} is the same set of races no matter which comes first

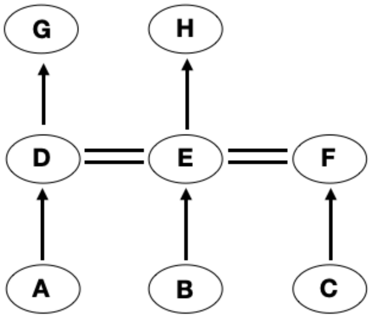
B. In most cases, will Jordan's scheme perform **worse**, the **same**, or **better** than the original (in terms of storage, computation, speed, etc.)?

Worse

happens-before is much more expensive than lockset; better to use it after lockset, not before

Initials:

7. [8 points]: Consider the following AS graph. Double-lines indicate peering relationships. Arrows indicate customer-provider relationships (in $K \rightarrow L$, K is the customer and L is the provider; traffic can flow both from K to L and from L to K).



A. According to the standard BGP export policies, which ASes will C learn about routes to?

B, E, F

B. Suppose that F became a customer of G (in addition to peering with E and being a provider for C). Which ASes would C learn about routes to? List **all** of the ASes that C can now reach, not just new ASes (if any).

B, E, F, G, D, A

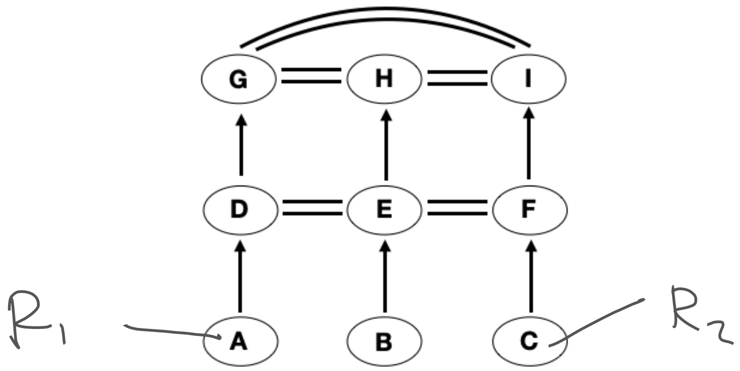
C. Suppose that F became a customer of H instead of G (in addition to peer with E and being a provider for C). Which ASes would C learn about routes to? List **all** of the ASes that C can now reach, not just new ASes (if any).

B, E, F, H

Initials:

8. [5 points]: If a RON node R_1 is placed in AS AS_1, and a RON node R_2 is placed in AS_2, then data routed between them will traverse the **default BGP path** between AS_1 and AS_2. If no such path exists, data cannot be routed from R_1 to R_2.

Consider the following AS graph.



Shuli is placing RON nodes throughout this network. So far they have placed one RON node R_1 in A and a second RON node R_2 in C.

A. With RON nodes in only those locations, what path will data travel between R_1 and R_2?

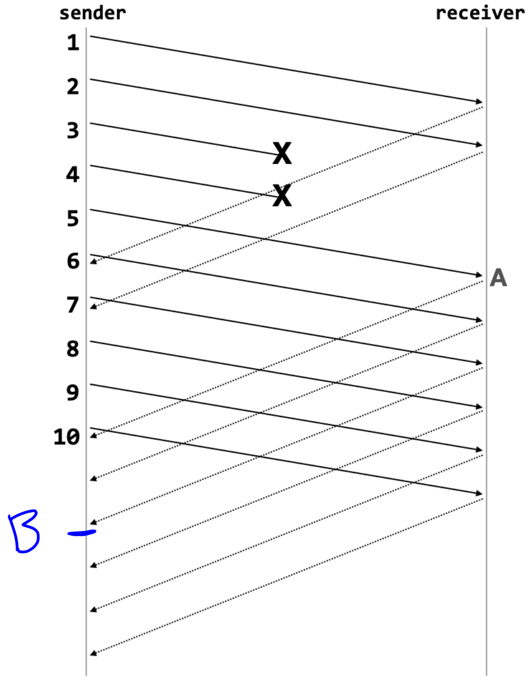
A - D - G - I - F - C

B. Assume that every link in the above graph—whether a customer/provider link or a peering link—has cost 1. Shuli wants data to travel between R_1 and R_2 on the shortest path possible. Shuli also wants to use as few additional RON nodes as possible. Where should they put additional RON node(s) in order to make this happen?

One node: R_3 in E

Then traffic travels A - D - E - F - C

9. [9 points]: Consider the waterfall diagram below, which depicts a TCP sender and receiver. The sender sends a window of ten packets (with sequence numbers 1–10); packets 3 and 4 are lost.



A. When the receiver sends an ACK in response to packet 5 (indicated on the diagram by the letter A), what will its sequence number be?

2

receiver has all packets up to and including 2.

B. At what point will the sender be able to retransmit packet 3, according to TCP's fast retransmit/fast recovery mechanism? **Clearly** label that point with the letter "B" on the graph. You can assume that the sender's retransmission timeout is large enough that packet 3 will not be retransmitted via timeout.

C. Assume that the receiver receives the retransmission of packet 3 successfully (i.e., the retransmitted packet does not also get lost). The receiver will send an ACK in response to that retransmission. What sequence number will it have?

3

Still don't have packet 4

10. [6 points]: Consider the following three queue management schemes: DropTail, RED (using packet drops, not packet marks), 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").

A. Can react to congestion before queues are full. → Part of the point of these schemes

DropTail [✓ RED] [✓ DCTCP] None

B. Requires **no** changes to a TCP sender's behavior. → Packet marks + diff. congestion rules in DCTCP

[✓ DropTail] [✓ RED] DCTCP None

C. Uses a measure of the average queue size to determine when to drop packets.

DropTail [✓ RED] DCTCP None

Uses instantaneous queue size (part of its appeal!)

Initials:

11. [10 points]: The drawing below shows the first six packets in two queues, P and Q. Each packet has a name (P1, P2, etc.) and an associated size (P1 is 150 bytes, for example). The head of the queue is the rightmost packet, i.e., P1 will be the first packet sent from queue P, and Q1 will be the first packet sent from Q.

Queue P:	P6: 200	P5: 200	P4: 100	P3: 150	P2: 100	P1: 150
Queue Q:	Q6: 100	Q5: 10	Q4: 20	Q3: 60	Q2: 30	Q1: 80

Suppose that a switch S containing these two queues (and only these two queues) is running deficit round robin with a quantum of 300 bytes for queue P and 100 bytes for queue Q. In each round, S process packets for queue P first and queue Q second.

A. In what order will these twelve packets be sent out of S? Specify your answer as a sequence of packet names.

P's credit: 300 50 350 100 400

P1 P2; Q1; P3 P4; Q2 Q3 Q4 Q5; P5 P6; Q6

Q's credit: 100 20 120 0 100

B. Assume the outgoing link from S sends 1000 bytes/second. Also assume that both queues have infinite offered load (i.e., they always have a packet ready to send; we are just showing you the first six in each queue). With these quantums, how much of that traffic will be allocated to queue P on average? Circle the **best** answer.

(a) 250 bytes/second

(b) 300 bytes/second

[✓ (c) 750 bytes/second]

(d) 1000 bytes/second

P is getting 300 bytes to Q's 100 bytes, and so 75% of the link.

C. Suppose that instead of 300 and 100, the quantums for P and Q were 3000 and 1000, respectively. What would change about the previous questions? Circle the **best** answer. Assume that everything else (e.g., packet sizes) remains the same; only the quantums have changed.

[✓ (a) The order in which the first twelve packets were sent out of S (i.e., the answer to Part A would change).]

(b) The amount of bandwidth allocated to queue P on average (i.e., the answer to Part B would change).

(c) Both the order in which the first twelve packets were sent out of S **and** the amount of bandwidth allocated to queue P on average would change.

(d) Nothing would change.

(e) It's impossible to tell.

Short-term fairness decreases, but the amount of bandwidth allocated on average does not ($3000/1000 = 300/100$)

Initials:

12. [6 points]: Answer the following two questions about Ethernet.

A. Assume the Ethernet is operating correctly. We have a single segment of Ether with two stations. Can we add another station? Circle the **best** answer.

[✓ (a)] Yes, we can just tap another station into the Ether.

(b) No, unless we add another segment of Ether and connect it to this one with a packet repeater.

(c) No, we can't add a station into this configuration.

B. Consider Ethernet's collision control algorithm (fig. 3 in the paper) and TCP's congestion control mechanism. How do these algorithms compare? Circle **all** that apply.

(a) In both cases, senders wait a random amount of time before sending a packet.

[✓ (b)] In both cases, senders send less frequently when there is more load on the network.

(c) In both cases, packets are guaranteed to reach their destination (though it may require multiple retransmissions to get them there).

13. [4 points]: Answer the following true/false questions about the end-to-end principle.

(a) **True** / [✓ **False**] There is no value in adding reliability at lower layers of a network, since an end-to-end reliability check must happen regardless.

(b) [✓ **True**] / **False** A network that adheres to the end-to-end principle can typically support a more diverse range of applications, compared to one that doesn't.

14. [4 points]: Many datacenters use multipath routing. In doing so, they must be careful how these routing protocols affect TCP senders in the datacenter. Why? Circle **all** that apply.

(a) Because different parts of the datacenter are controlled by different companies, switches on one path may use a different queueing scheme than switches on another path (e.g., a TCP flow might be split between two paths, one that uses DCTCP, and one that uses RED).

[✓ (b)] The correct window size for congestion control may be different for each path, but a TCP sender can only use a single window size.

(c) Because datacenter networks *must* mark packets when there is congestion, rather than drop them, a TCP sender in a datacenter has to be built to respond to marks, not drops.

(d) The TCP standard requires it to be coupled with a specific link-state routing protocol in order to work at all.