Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.1800 Computer Systems Engineering: Spring 2024

Exam 1

There are **14 questions** and **12 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@mit.edu

1. [7 points]: The diagram below shows part of the DNS tables for three different name servers: `a.root.net.`, `names.com.`, and `ns1.google.com.`. Each table includes a list of hostnames in the first column, the IP addresses for those hostnames in the second column, and the corresponding record type in the third column.

```
a.root.net.       | com.              | 2.2.2.2 | NS
                  | org.              | 3.3.3.3 | NS
                  | net.              | 4.4.4.4 | NS

names.com.        | google.com.       | 5.5.5.5 | NS
                  | apple.com.        | 6.6.6.6 | NS

ns1.google.com.   | mail.google.com.  | 7.7.7.7 | A
                  | cal.google.com.   | 8.8.8.8 | A
                  | google.com.       | 9.9.9.9 | A
```

Assume that each name server is the **only** name server for its respective domain (i.e., there are no replicas of any name server; this is not the case in practice). Assume that all name servers are operating correctly and that there are no failures of any type (e.g., no name servers fail, no packets are lost, etc.).

A. What is the IP address of `ns1.google.com`? If you cannot determine the IP address from the information above, write "Unknown".

5.5.5.5

we get this from the NS record for google.com

B. What IP address will `dig google.com` return? If you cannot determine the result from the information above, write "Unknown".

9.9.9.9

A record for google.com

C. A DNS client c wants to learn the IP address of `mail.google.com` by making only a **single** DNS query to the name server `names.com.`, and no queries to any other name servers. Is this possible? For each of the following scenarios, select whether c can **always** learn the IP address of `mail.google.com` in this manner, can **sometimes** (but not always) learn the IP address, or can **never** learn the IP address. Circle the correct answer in each case.

(a) **Sometimes / Always / Never** If `names.com.` has enabled caching but not recursion.

(b) **Sometimes / Always / Never** If `names.com.` has enabled recursion but not caching.

(c) **Sometimes / Always / Never** If `names.com.` has enabled neither caching nor recursion.

names.com is not authoritative for this name; without caching or recursion, we won't get the IP. With recursion, names.com will perform the lookup on c's behalf. With caching, if the mapping is in the cache, names.com will return it; if it's not in the cache, c would need to make additional queries.

Initials: SOLUTIONS

2. [12 points]: Consider a user-level process p. Hannah observes the MMU of her computer make the following address translations (from virtual address to physical address) for p:

`0x11111111` → `0x44444111`
`0x22222222` → `0x55555222`
`0x33333333` → `0x66666333`

A. From these translations alone, can you determine the number of **bits** (not hex digits) used for the page number and the number of bits used for the offset? If so, state those numbers; if not, write "No".

No.

Might be tempting to say 12, since we see the last 3 hex digits preserved in each case. But it could also be 8 (2 hex digits) or 4 (1 hex digit).

B. Regardless of your answer to the previous question, assume that the first five hex digits in an address represent the page number and the last three represent the offset. Assume that the MMU continues to use the same page table for p. For each of the following, give your answer in hexadecimal. If there is not enough information to determine any of the hex digits, write `?` in their place (i.e., if you could not determine any part of an eight-digit address, you'd write `0x????????`). You can assume that for this process, every virtual address maps to a unique physical address.

(a) What is the physical address corresponding to the virtual address `0x11111222` ?

The first translation tells us that vir. page 11111 maps to phys. page 44444

0x44444222

Offset remains the same.

(b) What is the physical address corresponding to the virtual address `0x22233444` ?

0x????4444

(c) What is the **virtual** address corresponding to the **physical** address `0x55555333` ?

0x22222333

We can determine this because every vir. addr. maps to a unique phys. addr.

C. Hannah wants to prevent p from both reading and writing the data stored at a particular physical address a. Which of the following techniques should she use? Select **all** that apply.

(a) Make sure the `P` bit for the relevant page table entry is set.
(b) Make sure the `U/S` bit for the relevant page table entry is set.
(c) Make sure the `R/W` bit for the relevant page table entry is set.
(d) None of the above

P bit doesn't deal with access.
R/W bit only limits write access.

Initials: SOLUTIONS

3. [9 points]: You are implementing a banking system for the bank YourBank. Every time a customer makes an online request, the bank server creates a new thread to handle the request. Currently there are three requests (and so three threads):

- Request 1 is a transfer of $5 from A's account to B's account.
- Request 2 is a transfer of $10 from B's account to A's account.
- Request 3 is a transfer of $20 from C's account to D's account.

The specific transfers given don't matter a ton, but serve to illustrate the problems below.

All accounts exist and contain enough money to complete each transfer.

Assume that this system provides no functionality beyond the ability to transfer money from one account to another (this is not a very good bank!). For the purposes of the following questions, that means that you only need to consider the given code; there are no other functions to reason about.

A. One version of the bank transfer code looks like this:

```
transfer(from_account, to_account, amount):
   acquire(YourBank.lock)
   from_account = from_account - amount
   to_account = to_account + amount
   release(YourBank.lock)
```

What kind of problem does this code present? Circle the **best** answer.

(a) Deadlock
(b) Exposes inconsistent state
(c) Performance
(d) None of the above

Coarse-grained locking. Slow! For instance, requests 1 & 3 can't run in parallel

B. A second version of the bank transfer code looks like this:

```
transfer(from_account, to_account, amount):
   acquire(from_account.lock)
   from_account = from_account - amount
   release(from_account.lock)
   acquire(to_account.lock)
   to_account = to_account + amount
   release(to_account.lock)
```

An interruption here would be bad. from_account has lost amount dollars but they've not yet been added to to_account.

What kind of problem does this code present? Circle the **best** answer.

(a) Deadlock
(b) Exposes inconsistent state
(c) Performance
(d) None of the above

C. A third version of the bank transfer code looks like this:

```
transfer(from_account, to_account, amount):
   acquire(from_account.lock)
   acquire(to_account.lock)
   from_account = from_account - amount
   to_account = to_account + amount
   release(from_account.lock)
   release(to_account.lock)
```

What kind of problem does this code present? Circle the **best** answer.

(a) Deadlock
(b) Exposes inconsistent state
(c) Performance
(d) None of the above

Example: interrupt Request 1 between lines 1 & 2; begin Request 2. Request 2 will get lock on B's account & need A's. Request 1 holds the lock on A's account, but needs B's.

4. [6 points]: Answer true or false for each of the following questions about UNIX.

A. True / False Checking the return value of `fork()` enables a child to execute different instructions from its parent.

B. True / False Running `command &` causes the shell to create a new process for the command, start it, and then move on to read the next command. Running `command`, on the other hand, causes the shell to run `command` to completion rather than creating a new process.

C. True / False If the shell uses `fork()` to create a new process to run a command, then the child process, and not the parent process, executes the command.

The shell is also an example of parts A and C. Shells fork a new process to run commands, and checking fork()'s return value allow the process to determine whether it's the child or parent.

A new process is always created.

Initials: SOLUTIONS

5. [8 points]: Jordan is implementing a bounded buffer. Currently he has written the following code for `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 share a bounded buffer `bb` that uses the above code. Assume that `N` is so large that every time a sender calls `send`, there will be room in the buffer for the message (i.e., `bb` will never be full).

A. Given that the buffer can never be full, will this code ever deadlock?

(a) This code will never deadlock.
(b) This code will sometimes deadlock.
(c) This code will always deadlock.

It will deadlock if we call receive on an empty buffer (when bb.out = bb.in)

Jordan updates his code to use condition variables (i.e., his code uses `wait()` and `notify()` instead of `acquire()` and `release()`). Assume that Jordan's code is now correct (i.e., if there were errors in his initial code, they are no longer present).

B. What is the benefit of using condition variables as opposed to calling `acquire()`/`release()` directly as in his initial code? Circle the **all** that apply.

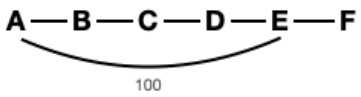
(a) Condition variables eliminate the need for locks.
(b) If used properly, condition variables help prevent threads from being woken up when they don't have work to do. this is the main point of condition variables.
(c) Condition variables allow threads to explicitly tell the processor when to suspend the current thread as opposed to waiting to be preempted. acquire + release + yield would also do this
(d) Condition variables prevent threads from being preempted by the operating system.

6. [3 points]: Which of the following is the best option to improve performance on a workload that consists of many random reads from a few large files? Assume that the data is currently stored on a hard-disk drive (HDD). Circle the **best** answer.

(a) Lay the files out contiguously on disk. → Not helpful for random reads
(b) Add a cache that stores the value of previous reads. Assume that the cache is large enough to hold 10% of the total data, and that it evicts data using a least-recently-used policy. → Also not particularly helpful for random reads (but could help a bit)
(c) Replace the HDD with a solid state drive (SSD). good for random reads!

Initials: SOLUTIONS

7. [8 points]: Consider the following network, which is running a distance-vector protocol. Every link in the network has cost 1, except the link A-E, which has cost 100.



At time 0, assume that every node in the network knows its neighbors, and its link costs to those neighbors, but nothing else. Nodes send advertisements synchronously every ten seconds at times 10, 20, 30, etc. Assume that the latency on each link is virtually zero and that advertisements are integrated within one second. This means that if a node sends an advertisement at time 10, every node that is meant to receive it will have the advertisement, and have integrated it, by time 11. Assume that no advertisements are lost.

A. By what time (in seconds) will A know **some** route to F?

11 It will hear about the path A-E-F in the first round

B. By what time (in seconds) will A know the **min-cost** route to F?

41 Knowledge of the path A-B-C-D-E-F has to propagate, and this takes four rounds.

C. By what time (in seconds) will C know **some** route to F?

21 C will here about both paths to E in the second round and will choose the min cost path.

D. By what time (in seconds) will C know the **min-cost** route to F?

21

Initials: SOLUTIONS

8. [6 points]: Answer the following two questions about Ethernet.

A. True / False Because Ethernet implements carrier detection and interference detection, the only way for two packets to collide is if their respective senders start sending at precisely the same time. If one sender starts before it hears the other, packets collide.

B. Ethernet's retransmission algorithm can be approximated by the following code, which is called every time there is a packet p to send.

```
1: load = 1
2: until p is sent successfully
3:    r = random number between 1 and load
4:    wait r time slots
5:    send p as soon as the ether is quiet
6:    load = 2*load
```

Grace is implementing this algorithm, and thinks that choosing a random number is too computationally expensive. Instead, she replaces line 3 with the line `r = load`.

Compared to the algorithm above, what will Grace's network experience? Circle the **best** answer.

(a) Grace's network will experience higher utilization.

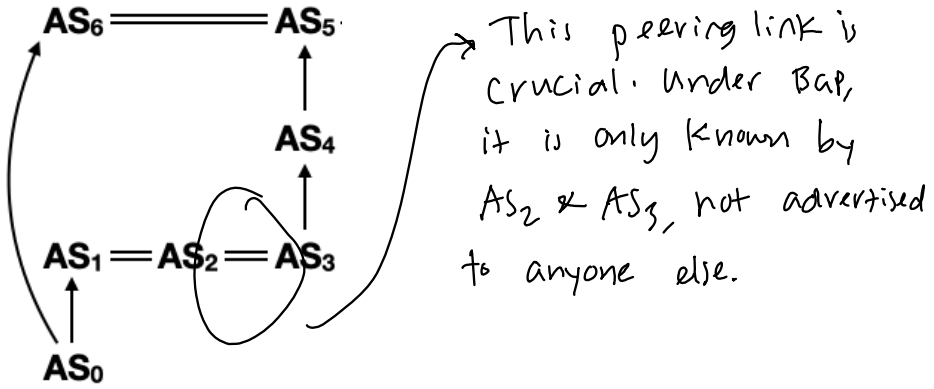
(b) Grace's network will experience the same utilization.

(c) Grace's network will experience lower utilization.

If multiple controllers have the same load estimate, once a collision takes place between them, they'll continue to pick the same timeslot at which to send, and so keep colliding.

Initials: SOLUTIONS

9. [12 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).



This network uses BGP with the standard import and export policies. Assume that there are no failures in this network (e.g., no advertisements are lost).

A. Under BGP, which path(s) to AS_2 will AS_0 learn about? Write down the path(s).

$AS_0 - AS_1 - AS_2$

B. Under BGP, which path(s) to AS_3 will AS_0 learn about? Write down the path(s).

$AS_0 - AS_6 - AS_5 - AS_4 - AS_3$

Now suppose that **the link between** AS_0 **and** AS_6 **fails** such that AS_6 can no longer provide transit for AS_0. Ophelia is concerned that this failure may prevent AS_0 from reaching AS_6 and starts deploying a Resilient Overlay Network (RON) to help. She starts by deploying one RON node in AS_0 and another in AS_6. For each scenario below, decide whether a machine in AS_0 would be able to send traffic to a machine in AS_6. You only need to answer "yes" or "no"; you don't need to give an explanation. Consider each scenario separately.

C. Ophelia deploys two additional RON nodes: one in AS_2 and one in AS_3.

Yes

Here (and in E) RON nodes in AS_2 & AS_3 bridge the gap between those two ASes.

D. Ophelia deploys two additional RON nodes: one in AS_2 and one in AS_4.

NO

The RON nodes in AS_2 & AS_4 can't reach each other

E. Ophelia deploys three additional RON nodes: one in AS_2, one in AS_3, and one in AS_4.

Yes

Initials: SOLUTIONS

10. [8 points]: Consider a reliable transport protocol KTP that uses sequence numbers and ACKs the same way that TCP does, but **only** retransmits packets after a timeout; there is no fast-retransmit/fast-recovery mechanism. KTP also does not have a congestion control mechanism, and instead uses a fixed window size.

A. Phoebe is testing out two different configurations of KTP. In the first—call it C_1—she sets the retransmission timeout value (the RTO) to .95 seconds. In the second—call it C_2—she sets the RTO to 1.05 seconds.

Consider a KTP connection between a sender S and a receiver R. Assume that the round-trip-time between S and R is one second and does not change over the course of the transmission.

Which scenario **best** describes the difference between the two configurations of KTP?

(a) S will send more packets in configuration C_1 than C_2, but only if there is loss on the network.

(b) S will send more packets in configuration C_1 than C_2 even if there is *not* loss, but it will only send a few more packets in C_1 than in C_2.

(c) S will send more packets in configuration C_1 than C_2 even if there is *not* loss, and it will send significantly more packets in C_1 than in C_2.

(d) There will be no difference between the transmissions in either configuration.

B. Sadhana is also experimenting with KTP. She decides to do away with the fixed window size and add TCP's AIMD congestion control mechanism, but she's skeptical about slow-start and decides to skip it (i.e., for an entire connection, her senders will adjust their window according to AIMD). Which of the below scenarios **best** describes Sadhana's KTP set-up?

(a) Short connections that only need to send a few packets will take longer to complete than if she used slow-start.

(b) In general, senders will see more loss than if she used slow-start.

(c) Connections will reach the optimal window size more quickly than if she were using slow-start.

S will spuriously retransmit every packet because its RTO estimate is too small => 2x times the traffic on the network.

The goal of slow-start is to get up to the correct window size quickly at the start of the connection.

Initials: SOLUTIONS

11. [6 points]: A sender S is in the midst of sending data to a receiver using a window-based transport protocol. Kevin observes how S's window changes over time. For each of the observations below, decide whether S could be using TCP or DCTCP (or neither). If the sender could be using *either* TCP or DCTCP, circle both of them.

(a) **TCP / DCTCP / Neither** S's window is currently 100 packets. It then sends a window's worth of data, and in response to the corresponding ACKs, adjusts its window to 70 packets.

(b) **TCP / DCTCP / Neither** S's window is currently 100 packets. It then sends a window's worth of data, and in response to the corresponding ACKs, adjusts its window to 50 packets.

(c) **TCP / DCTCP / Neither** S's window is currently 100 packets. It then sends a window's worth of data, and in response to the corresponding ACKs, adjusts its window to 10 packets.

12. [5 points]: For each of the characteristics below, decide whether it describes a P2P network such as BitTorrent, a CDN such as Akamai, or both.

(a) **P2P / CDN / Both** Uses an overlay network.

(b) **P2P / CDN / Both** Works well for streaming video.

(c) **P2P / CDN / Both** Uses DNS rewrites.

(d) **P2P / CDN / Both** Has a hierarchical organization.

(e) **P2P / CDN / Both** Tolerates individual machine failures.

13. [2 points]: TCP uses sequence numbers and acknowledgments (ACKs). As a packet traverses the network, where is its sequence number stored? Circle the **best** answer.

(a) As part of the packet's TCP header.

(b) As part of the packet's data.

(c) On switches along the path.

(d) Only at the endpoints, not also as part of the packet.

Note that this is a question about TCP but also about the E2E argument in a way.

Initials: SOLUTIONS

S_1 > R — D
S_2 >

14. [8 points]: Consider a network where two senders, S_1 and S_2, are sending packets into a switch R, which then sends them along to a destination D.

R has two queues: one for packets from S_1 and another for packets from S_2. Each of these queues can hold 1000 packets. The switch uses a scheduling algorithm to decide how to send packets from these queues to D, and attempts to give each sender an equal share of the bandwidth over time.

S_1 sends 1000 packets to R. Each of those packets is between 5 and 15 bytes long; the average packet size is 10 bytes. S_2 sends 100 packets to R. Each packet is between 50 and 150 bytes long; the average packet size is 100 bytes.

You are able to observe packets on the link between R and D. For each of the following observations, decide whether R could be using Deficit Round Robin (DRR). Write "yes" or "no" for your answers; you don't need to give an explanation. Treat each observation independently (i.e., the first observation has nothing to do with the second).

You can assume that in each case, the quantums for both queues are equal, but you do not know the precise value. The value may also be different from one observation to the next (i.e., the quantum value in part A may not be the same as in part B).

A. Ten packets from S_1, followed by one packet from S_2, followed by ten packets from S_1, followed by one packet from S_2.

Yes

Example: every packet from S_1 is 10B, every packet from S_2 is 100B, quantum = 100.

B. One packet from S_1, followed by one packet from S_2, followed by one packet from S_1, followed by one packet from S_2.

NO

S_2 must be accumulating at least 50B of credit per round, and thus so must S_1, meaning S_1 should be able to send > 1 packet in between.

C. 1000 packets from S_1, followed by 100 packets from S_2.

Yes

Example: same as in part A but with quantum = 10000

(a poorly-set quantum for this set up, but allowed!)

Initials: SOLUTIONS