Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.1800 Computer Systems Engineering: Spring 2023

Exam 2

There are **15 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:

Kerberos ID:

1. [8 points]: Three of our TAs are purchasing disks to use with different RAID set-ups: Allen is using RAID-1, Marlena is using RAID-4, and Hannah is using RAID-5. The disks all cost the same amount, and each disk is able to store $m >> 1$ blocks of data. Each TA has $k \cdot m$ blocks of data to store, where $k >> 1$ (i.e., if they used no redundancy whatsoever, they could each store their data on k disks).

Allen, Marlena, and Hannah have purchased **exactly** as many disks as needed to store their data in their respective setups (this quantity may be larger than k because of the need to store redundant data).

A. Which TA spent the most money on disks? If there is a tie, select multiple TAs.

(a) Allen
(b) Marlena
(c) Hannah
(d) There is not enough information to decide

B. What is the maximum number of **data** blocks (not parity blocks) that each TA's system can read concurrently, assuming that each read is processed at the same time on a different disk, and each disk can read exactly one block at a time? Give your answers as formulas in terms of k and m.

(a) Allen's system (RAID-1):

(b) Marlena's system (RAID-4):

(c) Hannah's system (RAID-5):

C. On each TA's system, a single data block gets corrupted. Assuming that the failure has already been detected, how many disks must the system read from in order to correct the error? Give your answers as formulas in terms of k and m.

(a) Allen's system (RAID-1):

(b) Marlena's system (RAID-4):

(c) Hannah's system (RAID-5):

2. [6 points]: Ophelia has decided to combine the ideas of GFS and RAID into a single system, RAIDO. In RAIDO, files are broken into three chunks—F_1, F_2, F_3—and a parity block $P = F_1 \oplus F_2 \oplus F_3$ is calculated. F_1, F_2, F_3, and P are stored on four separate machines.

Answer the following questions about GFS and RAIDO. In all parts assume that GFS is using its default configuration with a replication factor of three. When we consider failures in GFS, assume that the controller will **never** fail (we're only worried about failures related to storing actual file data).

A. Consider a single file F that is B gigabytes large. Which system will require more storage space for this file? Circle the **best** answer.

(a) GFS
(b) RAIDO
(c) GFS and RAIDO will require the same amount of storage space for F
(d) It is impossible to tell

B. Which system(s) can recover from a single-disk failure? Circle the **best** answer.

(a) GFS
(b) RAIDO
(c) Both GFS and RAIDO
(d) Neither GFS nor RAIDO

C. Which system(s) can recover from a two-disk failure, where both disks fail at exactly the same time? Circle the **best** answer.

(a) GFS
(b) RAIDO
(c) Both GFS and RAIDO
(d) Neither GFS nor RAIDO

3. [7 points]: Consider a system that uses a log and cell storage but no cache. The code for writing and recovery is given below; reads come directly from cell storage. Note that the code for writing here is **different** than the code you saw in lecture (and thus is not guaranteed to be correct).

```
write(var, value):
  old_value = read(var)  % read from cell storage
  cell_write(var, value)
  log.append(current_transaction_id, "UPDATE", var, old_value, value)

recover(log):
  commits = []
  for record r in log[len(log)-1] .. log[0]:
    if r.type == COMMIT:
      commits.add(r.transaction_id)
    if r.type == UPDATE and r.transaction_id not in commits:
      cell_write(r.var, r.old_value)
```

The system begins to execute transactions T_1 through T_{100}, which read and write the variables `A` and `B`. `A` and `B` are initially set to zero. The transactions execute one after the other in order (i.e., the system runs T_1, then T_2, then T_3, etc.; transaction T_n commits before T_{n+1} begins). Transactions T_1 through T_{79} commit successfully, but the system crashes in the middle of running transaction T_{80}.

After the system comes back up, **but before `recover()` is run**, you observe the following values in cell storage: `A = 10`, `B = 20`. After `recover()` is run, the values in cell storage change to `A = 10`, `B = 30`.

A. What can you say about the last **committed** value for A? Circle the **best** answer.

(a) The last committed value for `A` was `A=10`.

(b) The last committed value for `A` cannot be determined.

B. What can you say about the last **committed** value for B? Circle the **best** answer.

(a) The last committed value for `B` was `B=30`.

(b) The last committed value for `B` was `B=20`.

(c) The last committed value for `B` cannot be determined.

4. [6 points]: Consider the log below. Assume that this log is complete; there were no other updates to any of the relevant variables before or after what you see pictured here.

```
    +--------+--------+--------+--------+--------+--------+--------+
TID |   T0   |   T0   |   T1   |   T1   |   T1   |   T2   |   T2   |
    | UPDATE | UPDATE | UPDATE | UPDATE | COMMIT | UPDATE | UPDATE |
OLD | A=0    | B=0    | C=0    | D=0    |        | C=15   | D=25   |
NEW | A=10   | B=20   | C=15   | D=25   |        | C=30   | D=40   |
    +--------+--------+--------+--------+--------+--------+--------+
```

Notice that transaction `T1` makes two updates. Suppose that `T1` reads the value of `A` between these two updates and finds the value to be `A=10`. Which of the following is correct? Circle the **best** answer.

(a) This is the expected behavior for a system implementing isolation via strict two-phase locking with reader/writer locks.

(b) This is *not* the expected behavior for a system implementing isolation via strict two-phase locking with reader/writer locks, but could occur in an incorrect implementation where transactions release the writer locks prior to their commit point.

(c) This is *not* the expected behavior for a system implementing isolation via strict two-phase locking with reader/writer locks, but could occur in an incorrect implementation where a writer lock for a variable can be held at the same time as a reader lock for that variable.

(d) Both (b) and (c) are correct.

(e) None of the above.

5. [4 points]: Veronica is implementing a two-phase locking protocol on a machine with 512 cores. As part of her implementation, she is making a decision about how to handle deadlock. She has two choices:

- **Method 1:** Use a centralized deadlock detector. When this detector detects deadlock, it aborts the transaction that's using the fewest number of resources.
- **Method 2:** Whenever a transaction attempts to acquire a lock but is denied, abort that transaction.

Veronica implements both of these methods and runs them over the same write-intensive workload. During each experiment, she measures the number of transactions per second that the system can process. What do you expect the result to be? Circle the **best** answer.

(a) The two methods will perform the same because neither will abort any transactions.

(b) Method 1 will process more transactions per second than Method 2 because a correctly-implemented two-phase locking protocol should never deadlock (and thus Method 1 will never abort a transaction).

(c) Method 1 will process more transactions per second than Method 2, although it may detect some deadlocks.

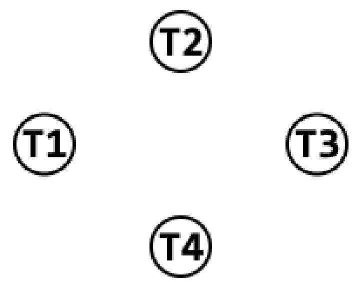
(d) Method 2 will process more transactions per second than Method 1.

Initials:

6. [8 points]: Consider the schedule of transactions below. Each step is labeled `Ti.j` where `i` refers to the transaction ID and `j` refers to the step within transaction `Ti`.

A. Next to the below schedule, draw its conflict graph. We've already drawn the nodes for you, you just need to fill in any edges.

```
T1.1 write(x)
T2.1 read(x)
T4.1 read(y)
T4.2 read(x)
T2.2 write(y)
T3.1 read(y)
T4.3 read(z)
T2.3 write(z)
T3.2 write(z)
```



B. It turns out that transaction `T3` is incomplete: it also needs to perform a read of the variable x (i.e., `read(x)`). In terms of the semantics of `T3`, however, it does not matter where this read occurs. To create a conflict-serializable schedule, where should this read happen?

(a) The resulting schedule will be conflict-serializable no matter where `read(x)` occurs in `T3`.

(b) The resulting schedule will be conflict-serializable only if `read(x)` occurs in `T3` prior to `T1`'s `write(x)` (i.e., prior to the very first step of the given schedule).

(c) The resulting schedule will be conflict-serializable only if `read(x)` occurs in `T3` after `T1`'s `write(x)` (i.e., after the first step of the given schedule).

(d) It is impossible to create a conflict-serializable schedule that includes this read.

7. [8 points]: Peyton's MapReduce system consists of five machines: a controller machine and four worker machines $M_1, \ldots, M_4$. Each worker machine has four cores, and can run one map or reduce job per core in parallel. Each worker machine M_i also has access to its own local disk D_i.

Peyton's MapReduce job starts with sixteen map tasks; the controller assigns one task per core.

A. After running for some time, all map tasks have completed except `Task 1` running on M_1. What will the controller do in response? Circle the **best** answer.

(a) Nothing, unless M_1 fails.

(b) Assign `Task 1` to an available machine.

(c) Subdivide `Task 1` into multiple smaller map tasks, and assign those smaller tasks to available machines.

(d) Kill the process running `Task 1` on M_1 and begin to assign reduce tasks, ignoring the result of `Task 1`.

B. Eventually, M_1 fails. This is the only failure in the system. Circle True or False for each of the following.

(a) **True / False** M_1 will notify the controller of its failure.

(b) **True / False** The controller will re-assign any jobs currently running on M_1 to another available machine.

(c) **True / False** The controller will copy data on D_1 to another disk.

Initials:

8. [8 points]: A coordinator C is running a two-phase commit protocol with two servers, X and Y. During this process, you observe a snippet of X and Y's logs for a particular transaction, `T1`. These snippets are from the exact same time, and they represent the **end** of each server's log (i.e., there are not yet any entries after the ones shown here).

For each of the following log snippets, specify whether it is consistent with a correct two-phase commit protocol (circle the correct answer). Consider each snippet independently.

A. Consistent / Not Consistent

```
    X's log                                  Y's log
    +--------+---------+                     +--------+---------+
TID |   T1   |   T1    |                 TID |   T1   |   T1    |
    | UPDATE | PREPARE |                     | UPDATE | PREPARE |
OLD | A=0    |         |                 OLD | Z=0    |         |
NEW | A=10   |         |                 NEW | Z=20   |         |
    +--------+---------+                     +--------+---------+
```

B. Consistent / Not Consistent

```
    X's log                                  Y's log
    +--------+---------+--------+            +--------+
TID |   T1   |   T1    |   T1   |        TID |   T1   |
    | UPDATE | PREPARE | COMMIT |            | UPDATE |
OLD | A=0    |         |        |        OLD | Z=0    |
NEW | A=10   |         |        |        NEW | Z=20   |
    +--------+---------+--------+            +--------+
```

C. Consistent / Not Consistent

```
    X's log                                  Y's log
    +--------+---------+--------+            +--------+---------+
TID |   T1   |   T1    |   T1   |        TID |   T1   |   T1    |
    | UPDATE | PREPARE | COMMIT |            | UPDATE |  COMMIT |
OLD | A=0    |         |        |        OLD | Z=0    |         |
NEW | A=10   |         |        |        NEW | Z=20   |         |
    +--------+---------+--------+            +--------+---------+
```

D. Consistent / Not Consistent

```
    X's log                                  Y's log
    +---------+--------+--------+            +---------+--------+--------+
TID |   T1    |   T1   |   T1   |        TID |   T1    |   T1   |        |
    | PREPARE | UPDATE | COMMIT |            | PREPARE | UPDATE | COMMIT |
OLD |         | A=0    |        |        OLD |         | Z=0    |        |
NEW |         | A=10   |        |        NEW |         | Z=20   |        |
    +---------+---------+-------+            +---------+--------+--------+
```

Initials:

9. [7 points]: Consider a Raft set-up with N nodes. In Term 1, Node X is elected the leader, and client C_1 sends two updates to X; call them $\texttt{C}_1\texttt{.1}$ and $\texttt{C}_1\texttt{.2}$. At the beginning of Term 2, Node Y is elected the leader. You know that at the beginning of Term 2, both $\texttt{C}_1\texttt{.1}$ and $\texttt{C}_1\texttt{.2}$ have been committed.

A. True / False Assuming that there were no failures (i.e., no machines failed and no links went down), $\texttt{C}_1\texttt{.1}$ and $\texttt{C}_1\texttt{.2}$ must be reflected in the logs of all N nodes in the system.

No updates occur during Term 2, but a network partition occurs. For the purposes of this question, you can assume that N is an odd number, which means that we can't have an equal number of nodes on both sides of the partition. We'll refer to the side with more nodes as the "larger" side (the other side is the "smaller" side).

An election timeout occurs, and Node Z is elected leader in Term 3. Y also remains a leader. Assume that there are no additional failures (including packet losses) in the network, and that each link has low latency (i.e., we aren't considering excessive network delays for any part of this problem).

B. Which of the following is true? Circle the **best** answer.

(a) Node Y is on the smaller side of the partition; Node Z is on the larger side.

(b) Node Y is on the larger side of the partition; Node Z is on the smaller side.

(c) Nodes Y and Z are on opposite sides of the partition, but we can't tell which node is on the larger side.

(d) Nodes Y and Z are not necessarily on opposite sides of the partition.

Assume—regardless of the answer to Part B—that Nodes Y and Z were indeed on opposite sides of the partition. During Term 3, client C_1 sent Y two updates, $\texttt{C}_1\texttt{.3}$ and $\texttt{C}_1\texttt{.4}$; Client C_2 sent Z one update, $\texttt{C}_2\texttt{.1}$.

C. Assume that the network partition has healed and that the system has gone through multiple additional terms. There may have been additional failures. Circle **True** or **False** for each of the following.

(a) **True / False** It's possible that none of the updates $\texttt{C}_1\texttt{.3}$, $\texttt{C}_1\texttt{.4}$, and $\texttt{C}_2\texttt{.1}$ appear in any log.

(b) **True / False** It's possible that all of the updates $\texttt{C}_1\texttt{.3}$, $\texttt{C}_1\texttt{.4}$, and $\texttt{C}_2\texttt{.1}$ appear in every log.

(c) **True / False** It's possible that just one client's updates appear in every log (i.e., either $\texttt{C}_1\texttt{.3}$ and $\texttt{C}_1\texttt{.4}$ are in every log, or $\texttt{C}_2\texttt{.1}$ is in every log).

Initials:

10. [8 points]: Amelia is building a system that authenticates users via usernames and passwords. She knows that the correct way to do this is to store the following in a table on the server, where *salt* is a randomly-generated salt (each user gets their own salt) and H is a "slow" hash function:

username	salt	$H(password \mid salt)$

In this system, Amelia would authenticate users by taking their inputted password, concatenating it with the stored salt, and checking whether the hash of that string matched what is stored in the table.

Amelia wants to mix things up, and tries implementing her system by storing different information in the table. Below, we propose four schemes. Each scheme stores **only** the three given pieces of information about each user.

For each of the proposed schemes below, decide whether Amelia could use this information to properly authenticate users without opening her system up to additional attacks from adversaries with read-access to her system. Circle the schemes that allow Amelia to do so.

You can assume for all parts of this question that any salts are generated properly (e.g., they are random in the way that we need them to be) and that H is a proper slow hash function (cryptographically secure, etc.).

(a)

username	salt	$H(password) \mid salt$

(b)

username	salt	$H(password \mid H(salt))$

(c)

username	$H(salt)$	$H(password \mid salt)$

(d)

username	$H(salt)$	$H(password \mid H(salt))$

(e) None of the schemes allow Amelia to properly authenticate users without opening her system up to additional attacks from adversaries with read-access to her system.

Initials:

11. [6 points]: Consider the code below. This code is very similar to one of the examples you saw in Lecture 22, but not exactly the same.

```
void win() { printf("code flow successfully changed\n"); }

int main(int argc, char **argv) {
   char x;
   char buffer[32];
   char y;
   gets(buffer);
   int i;
   int j;
}
```

Sam runs this code on a machine that has no protection against stack-smashing. On his machine, integers are four bytes long and characters are one byte long. Any pointers—e.g., the base and instruction pointers, function pointers—are also four bytes long. When a function is called, the following happens (in this order):

1. Any arguments to the function are pushed onto the stack
2. The base pointer (`BP`) is pushed onto the stack
3. The instruction pointer (`IP`) is pushed onto the stack
4. Local variables to the function are pushed onto the stack

The list above describes **everything** that happens on the stack. There aren't, e.g., any other pointers pushed onto the stack between BP and IP. Sam's goal is to overwrite the saved instruction pointer and force the function `win()` to be called, by inputting a string into this program.

A. How long should Sam's string be, in **bytes**?

B. Where should the address of `win()` appear in the string? Circle the **best** answer.

(a) As the first four bytes of the string
(b) As the final four bytes of the string
(c) Anywhere; it doesn't matter, so long as the address is there
(d) The address of `win()` doesn't need to appear in the string to get the code to jump to this function; Sam can use a random string, and it will work

Initials:

12. [4 points]: Jay is experimenting with the toy example in Listing 1 of the Meltdown paper, which is given below. Each element in `probe_array` is one byte.

```
raise_exception();
// the line below is never reached
access(probe_array[data * 4096])
```

Unlike the paper, Jay is using a system that has 2048-byte pages. Jay runs the example on his system, and observes that the access time for Page 64 is around 200 milliseconds; for every other page, the access time is around 400 milliseconds. Assume that pages are numbered starting at 0.

Given Jay's observations, what is the value of `data`? If there is more than one possible value of `data`, list them all.

13. [8 points]: Alice and Bob are communicating via what they hope is a secure channel. They've exchanged a symmetric key, k, which they use to encrypt, decrypt, *and* MAC.[1] You can assume that only Alice and Bob know k. They've also agreed ahead of time on two random numbers: r_A (Alice's random number) and r_B (Bob's random number). $r_A \neq r_B$, and you can assume that, like k, only Alice and Bob know these two numbers.

To send a message to Bob, Alice does the following:

1. Computes $c = encrypt(k, m \mid r_A)$
2. Computes $h = MAC(k, c)$
3. Sends $c \mid h$ to Bob

When Bob receives this message, he recomputes $MAC(k, c)$ and makes sure that the result equals h. He also decrypts c and checks that resulting message ends in r_A.

For Bob to send a message to Alice, he does the same, but uses r_B where Alice uses r_A. When Alice decrypts, she checks that the resulting message ends in r_B (and also checks that the MAC works out).

Which of the following is true of Alice and Bob's scheme? Circle **all** that apply.

(a) It provides confidentiality.

(b) It provides integrity.

(c) It is secure against replay attacks.

(d) It is secure against reflection attacks.

(e) None of the above

[1] In reality, we would use different keys for encryption and MAC'ing, but for simplicity we're going to use the same key for both, just like we did in Lecture 23.

Initials:

14. [6 points]: Answer True or False for each of the following.

(a) **True / False** Disabling caching from DNS would solve many of the problems that DNSSEC attempts to solve.

(b) **True / False** DNSSEC provides confidentiality.

(c) **True / False** DNSSEC helps prevent adversaries from launching DNS amplification attacks.

15. [6 points]: Consider a Tor circuit between a client A and a server S. A sends traffic to S via proxies P_1, P_2, and P_3 in that order. This is the exact same set up you saw in Lecture 24, and you can assume that the proxies have correctly installed all of the necessary state that lets them forward traffic along this circuit.

A. In order for Tor to work correctly, A must know which three proxies its traffic is traveling through. Why? Circle the **best** answer.

(a) So that A can reject any proxies that it doesn't trust.

(b) So that A can add the appropriate layers of encryption for onion routing.

(c) So that A can make sure its traffic is traveling on a path that doesn't introduce particularly high latency.

B. A sends data through this circuit by first adding multiple layers of encryption, one of which is stripped off at every step. What is the purpose of these layers of encryption in Tor? Circle the **best** answer.

(a) Because the data has the most layers of encryption when it leaves A (before any of the layers are stripped off), this gives A's traffic more protection on A's local network.

(b) So that an adversary cannot mount "timing" attacks, which correlate the timing of packets from A to P_1 with the timing of packets from P_3 to S.

(c) So that an adversary cannot observe the same data traveling across multiple hops in this circuit.

Initials: