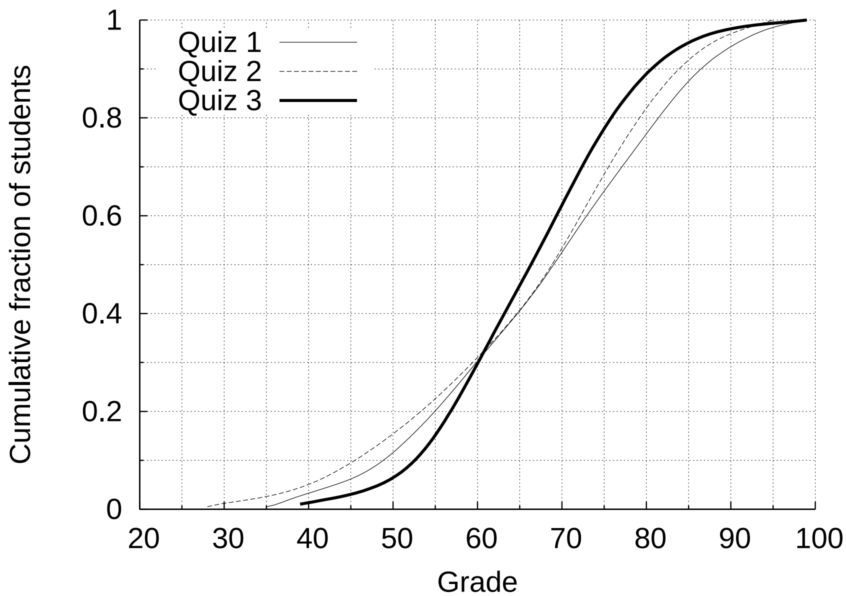
Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033 Computer Systems Engineering: Spring 2005

Quiz III Solutions



Statistics calculated over all students who took Quiz 3 on May 19, 2005.

Average: 66.9
Median: 67
Standard deviation: 10.8

Score range	# of students
39–49	6
50–59	43
60–69	62
70–79	58
80–89	19
90–99	3

Name: Alyssa P. Hacker

I Pot Pourri

1. [8 points]: The log-structured file system (LFS, reading #14)

(Circle True or False for each choice.)

A. True / False Selects the segment with the largest amount of free space to clean.

FALSE. *LFS uses a cleaning policy that takes both the amount of free space and the age of the segment into account.*

B. True / False Was designed to improve the performance of workloads dominated by small writes.

TRUE. *The authors argue that disk access latencies are not likely to show dramatic improvements, whereas reads are likely to be served from increasingly large RAM buffer caches.*

C. True / False Was designed to improve the performance of workloads dominated by small reads.

FALSE. *Same reason as in choice B above.*

D. True / False Forces data to disk before returning from an application's `write` of a file.

FALSE. *LFS gathers a segment's worth of data before data is flushed to disk.*

2. [8 points]: Ben Bitdiddle is implementing System R (reading #16). Unfortunately, he misreads the "write ahead log" protocol and instead implements the following "write behind log" protocol:

> *The state of the shadow database on disk is replaced by the current database state, before the transaction log is written to disk.*

In Ben's implementation, the above step may occur without taking a checkpoint. As described in the System R paper, a transaction commits at the instant its commit record appears on the log disk. The COMMIT() call returns to the application only after the commit record is on disk.

During failure recovery, the system starts at the end of the log and makes a list of winners and losers, as in the System R paper. It then undoes all losers.

Assume that failures do not corrupt the contents of the disk. Which of these assertions is true of Ben's resulting system?

(Circle True or False for each choice.)

A. True / False To provide correct transactional semantics in Ben's system, the failure recovery process must redo committed winners.

FALSE. *Committed winners will have their changes already reflected in the shadow database. The reason why the write-behind protocol does not provide recoverability is that some changes may have been made to the shadow database without a corresponding log entry being present, and the recovery process wouldn't know to undo those changes.*

B. True / False During failure recovery, Ben's system always undoes the actions of all transactions that were pending (neither committed nor aborted) at the time of the failure.

FALSE. *A pending transaction may have modified the shadow database, and the system may have failed before the corresponding log entry was written.*

C. True / False During failure recovery, Ben's system may sometimes undo the actions of the last committed transaction.

FALSE. *The recovery process only undoes losers. A committed transaction is a winner.*

D. True / False After the failure recovery process completes, the shadow database may sometimes contain the effects of actions that are not present in the log.

TRUE. *See choice B above.*

3. [8 points]: Which of these assertions about the Unison file synchronizer ("*How to Build a File Synchronizer*"), is true? (This paper is not in the handouts.)

(Circle True or False for each choice.)

A. True / False Unison is a trace-based file synchronizer.

FALSE. *Unison is a state-based file synchronizer (reconciler).*

B. True / False By default, Unison uses fingerprints rather than modification times to detect updates.

TRUE.

C. True / False Unison may overwrite files incorrectly without a user's knowledge if a replica's archive file is deleted.

FALSE *If an archive file is deleted, Unison takes a conservative approach to recover: any files that are unequal between the two replicas are marked as conflicts that the user must synchronize by hand.*

D. True / False When propagating files between replicas, Unison always transfers the entire file to ensure consistency.

FALSE. *Unison uses the rsync protocol internally, which does not transfer the entire file but only differences.*

4. [8 points]: In the paper "*Reflections on Trusting Trust*" (reading #18), Ken Thompson argues that

(Circle True or False for each choice.)

A. True / False The UNIX C compiler has no Trojan horses.

FALSE. *Thompson made no such claim, although it's quite likely that his compiler had no Trojan horses!*

B. True / False A binary program may contain a Trojan horse even if a careful analysis of its source code does not reveal one.

TRUE. *This point is one of the main ones in Thompson's paper, and the reason he went through the exercise of showing the self-reproducing code snippet in the beginning.*

C. True / False Programmers should read the source code of their compilers to be sure that they don't contain self-replicating code.

FALSE. *He didn't make this claim, and in any case, his main point is that even if you have the source code for the compiler, you can't be sure of its construction or provenance.*

D. True / False To be certain that a program will not exhibit malicious behavior when run, it suffices to write it entirely by yourself in assembly language, bypassing the untrusted compiler.

FALSE. *Thompson points out that any piece of system software can have the same problem that he raised with respect to the C compiler. An assembler is no exception.*

5. [8 points]: One theme in Professor Abelson's lecture was that the Internet has subjected people to laws from remote locations. He discussed three specific laws or legal standards:

A. The community standards clause of the "Miller Test" for obscenity.

B. The first amendment of the Constitution of the United States of America.

C. The "Good Samaritan" provision of the Communications Decency Act.

For each case and rulings given below, list which law or legal standard from the list above was applied by the Court in reaching their ruling.

(Fill in the blanks below with A, B, or C)

1. Ken Zeran is unsuccessful in his attempt to sue AOL for slander after Zeran is harrassed because of postings on AOL by an anonymous third party impersonating him.

   Law/standard cited by the Court: C. *The Court ruled that AOL, as a network provider, was not liable for the contents of messages that it carried.*

2. Robert Thomas, owner of the Amateur Action Bulletin Board System ("the nastiest place on earth") in Milpitas, CA is imprisoned after a suit brought by a resident of Memphis, TN.

   Law/standard cited by the Court: A. *That's right. Mr. Thomas should've known from the phone number of the downloading person that he was in Tennessee.*

3. Jane Doe, whose minor son is lured into a pornographic photo shoot by an online predator in an AOL chatroom, is unsuccessful in her attempt to sue AOL for participating in the sale and distribution of pornographic material.

   Law/standard cited by the Court: C. *Once again, as in (A) above, the court ruled that AOL wasn't liable for the contents of its chat rooms.*

4. In the case of *Reno v. ACLU*, the Court strikes down the "display provision" of the Communications Decency Act on the grounds that it subjects all communities to the standards of the most conservative community.

   Law/standard cited by the Court: B. *The "display provision" was found to be in violation of the right to free speech.*

II SOS: Stamp Out Spam

Spam, defined as unsolicited messages sent in large quantities, now forms the majority of all email and short message service (SMS) traffic worldwide. Various recent studies have estimated that about 100 billion (100×10^9) emails and SMS messages are sent per day, two-thirds of which is spam. Alyssa P. Hacker realizes that spam is a problem because it costs virtually nothing to send email, which makes it attractive for a spammer to send a large number of messages every day.

Alyssa starts designing a spam control system called *SOS*, which uses the following approach:

1. **Allocation.** Every sender is given some number of *stamps* in exchange for payment.
2. **Sending.** The sender (an outgoing mail server) attaches a fresh (unused) stamp to each email message.
3. **Receiving.** The receiver (an incoming mail server) tests the incoming stamp for freshness by contacting a *quota enforcer* that runs on a trusted server using a TEST_AND_CANCEL remote procedure call (RPC), which we describe just below. If the stamp is fresh, then the receiver delivers the message to the human user. If the stamp is found to be cancelled, then the receiver discards the message as spam.
4. **Quota enforcement.** The quota enforcer implements the TEST_AND_CANCEL RPC interface for receivers to use. If the stamp was not already cancelled, the quota enforcer cancels it in this procedure by storing cancellations in a database.

Alyssa's hope is that allocating reasonable quotas to everyone and then enforcing those quotas would cripple spammers (because it would cost them a lot), while leaving legitimate users largely unaffected (because it would cost them little).

Like postage stamps, SOS's stamps need to be unforgeable, for which cryptography can help. SOS relies on a central trusted stamp authority, SA, with a well-known public key, SA_{pub}, and a corresponding private key, SA_{priv}. Each sender S generates a public/private key pair, (S_{pub}, S_{priv}), and presents S_{pub} to SA along with some payment. In return, SA gives S a certificate (C_S) and allocates it a stamp quota.

$$C_S = \{S_{pub}, \texttt{expiration\_time}, \texttt{daily\_quota}\}_{SA_{priv}}$$

The notation $\{\texttt{msg}\}_k$ stands for the marshaling of `msg` and the signature (signed with key k) of `msg` into a buffer. In the certificate, `expiration_time` is set to a time one year from the time that SA issued the certificate, and `daily_quota` is a positive integer that specifies the maximum number of messages per day that S can send.

S is allowed to make up to `daily_quota` stamps, each with a unique integer `id` between 1 and `daily_quota`, and the current `date`. To send a message, S constructs and attaches a stamp with the following format:

$$stamp = \{C_S, \{\texttt{id}, \texttt{date}\}_{S_{priv}}\}$$

When a receiver gets a stamp, it first checks if the stamp is *valid* by running CHECK_STAMP_VALIDITY(stamp). This procedure verifies that the stamp has a properly signed, unexpired certificate, and that the contents of the stamp have not been altered. It also checks that the `id` is in the correct range, and that the `date` is either yesterday's date or today's date (i.e., each stamp has a two-day validity period).

If all these checks pass, then the stamp is considered *valid*. The receiver calls TEST_AND_CANCEL on valid stamps.

Unless otherwise mentioned, you should assume that:

1. No entity's private key is compromised.
2. It is impossible to forge a digital signature or break a cryptographic algorithm.
3. SA is trusted by all participants and no aspect of its operation is compromised.
4. Senders may be malicious. A malicious sender will attempt to exceed his quota; for example, he may attempt to send many messages with the same stamp, or steal another sender's unused stamps.
5. Receivers may be malicious; for example, a malicious receiver may attempt to cancel stamps belonging to other senders that it has not seen.
6. Most receivers cancel stamps that they have seen, especially those attached to spam messages.
7. Each message has exactly one recipient (don't worry about messages sent to mailing lists).
8. Spammers and other unsavory parties may mount denial-of-service and other resource exhaustion attacks on the quota enforcer, which SOS should protect against.

Alyssa implements TEST_AND_CANCEL, which runs at the quota enforcer, as follows. Because spammers have an incentive to reuse stamps, she wants to maintain the total number of TEST_AND_CANCEL requests done on each stamp. num_uses is a hash table keyed by $stamp$ that keeps track of this number.

The hash table supports two interfaces:

1. PUT$(table, key, value)$ inserts the $(key, value)$ pair into $table$.
2. GET$(table, key)$ returns the $value$ associated with key in $table$, if one was previously PUT, and 0 otherwise. Assume that a value of 0 is never PUT.

procedure TEST_AND_CANCEL$(stamp, client)$
{
  // assume that "client" is not a spoofed network address
  if CHECK_STAMP_VALIDITY$(stamp) \neq$ VALID **then return**;
  $u \leftarrow$ GET$(num_uses, stamp)$;
  if $u > 0$ **then** $status \leftarrow$ CANCELLED;
  else $status \leftarrow$ FRESH;
  $u \leftarrow u + 1$;
  PUT$(num_uses, stamp, u)$;
  SEND$(client, status)$; *// assume reliable data delivery*
}

6. [3 points]: Louis Reasoner looks at the TEST_AND_CANCEL routine, and declares, "*Alyssa, the client would already have checked if the stamp is valid, so you don't need to call* CHECK_STAMP_VALIDITY *again.*" Alyssa thinks about it, and decides to keep the check. Why?

(Answer legibly in the space below.)

This check prevents malicious receivers from mounting resource exhaustion attacks on the quota enforcer (see assumption #8 above). Without the check in place, a receiver R could call TEST_AND_CANCEL*(garbage, R), causing different garbage bits to be installed in num_uses and exhaust its space.*

7. [12 points]: Suppose that a recipient R gets an email message that includes a valid stamp belonging to S. Then, which of the following assertions is true?

(Circle True or False for each choice.)

A. True / False R can be certain that the email message came from S.

FALSE. *Nothing ties a stamp to a particular message. In fact, S could have sent a message with a stamp to T, and T could have used S's stamp without cancelling it on a message to R.*

B. True / False R can be certain of both the data integrity and the origin integrity of the certificate in the stamp.

TRUE. *The certificate is signed with SA_{priv}, and the assumptions state that no private key is compromised, that the cryptographic algorithms are not broken, and that digital signatures can't be forged. Any tampering of C_S will be detected.*

C. True / False R may be able to use the information in this stamp to cancel another stamp belonging to S with a different `id`.

FALSE. *Again, since private keys are uncompromised and the cryptographic algorithms are not broken, and a digital signature cannot be forged, R cannot manufacture a proper stamp that belongs to S unless it has already seen that stamp.*

D. True / False If an attacker breaks into a machine that has fresh stamps on it, he may be able to use those stamps for his own messages, even though the stamps were signed by another entity.

TRUE. *And, that might prevent users of the compromised machine from sending their own messages. One redeeming feature of this situation is that users will now notice, and have an incentive to keep their machines patched and more secure than in the status quo!*

E. True / False S may be able to tell that R received an email message if it finds (using TEST_AND_CANCEL) that the stamp attached to that message has been cancelled at the quota enforcer.

IT DEPENDS ON YOUR DEFINITION OF "MAY"! *As worded, this statement is ambiguous. S can send a message to R and wait until it thinks that R received it and called* TEST_AND_CANCEL. *Then, S can call* TEST_AND_CANCEL. *If the quota enforcer returns "CANCELLED", then S might be able to infer that R received the message, but it can't be sure of it because some other entity en route could've cancelled the stamp. The problem is that for any given message, S can't tell if R received it or not, so it isn't clear what "may" means in the assertion above.*

Note also that by doing a TEST_AND_CANCEL of a stamp that it used, S runs the risk of R discarding the message as spam. That would happen if R had not yet called TEST_AND_CANCEL on the stamp.

F. True / False If the email message were encrypted with R_{pub} by S, then R can be certain that no entity other than S or R could have read the contents of the message without S or R knowing.

TRUE. *Again, we're assuming that private keys aren't compromised, and that the cryptographic algorithms are good.*

The ACLU looks at Alyssa's proposal and throws a fit, arguing that SOS compromises the privacy of sender-receiver email communication because the stamp authority, which also runs the quota enforcer, may be able to guess that a given sender communicated with a given receiver. Alyssa decides that the SOS protocol should be amended to meet two goals:

G1. It should be computationally infeasible for the stamp authority (quota enforcer) to associate cancelled stamps with a sender-receiver pair.

G2. It should still be possible for a receiver to call TEST_AND_CANCEL and correctly determine a stamp's freshness.

Alyssa considers several alternatives to achieve this task, including ENCRYPT($\texttt{msg}, k$), which encrypts `msg` with key k, and HASH(`msg`), a cryptographically secure one-way hash function of `msg`.

8. [6 points]: Which of these methods achieves goals **G1** and **G2** mentioned above? You may assume that CHECK_STAMP_VALIDITY is no longer called inside TEST_AND_CANCEL and that TEST_AND_CANCEL has been modified to take in any bit-string as its first argument. Let S_{pub} be S's public key (from the certificate in the stamp) and R_{pub} be R's public key.

(Circle all goals that are met for each choice.)

A. G1 / G2 / Neither The receiving client R extracts $u = \{C_S, \texttt{id}, \texttt{date}\}$ from the stamp, and computes $e1 =$ ENCRYPT(u, S_{pub}). It then calls TEST_AND_CANCEL$(e1, R)$.

B. G1 / G2 / Neither The receiving client R extracts $u = \{C_S, \texttt{id}, \texttt{date}\}$ from the stamp, and computes $e1 =$ ENCRYPT(u, R_{pub}). It then calls TEST_AND_CANCEL$(e1, R)$.

C. G1 / G2 / Neither The receiving client R extracts $u = \{C_S, \texttt{id}, \texttt{date}\}$ from the stamp, and computes $h =$ HASH(u). It then calls TEST_AND_CANCEL(h, R).

This question turned out to have an unfortunate bug, because if the receiver did not use the bits of the signature of $\{\texttt{id}, \texttt{date}\}_{Spriv}$ *as part of the first argument to* TEST_AND_CANCEL*, G2 will never be met. That's because once it a receiver is in possession of any stamp owned by S, it can cancel all other stamps for the same* `date`*. Because this question comes across as too much of a "trick", we decided to eliminate it from grading and give everyone 6 points.*

Suppose the question had in fact asked about $\{C_S, \{\texttt{id}, \texttt{date}\}_{S_{priv}}\}$ *in each choice above. Then, G2 would have been true for A and C, but not for B. That's because each receiver would compute a different first argument to* TEST_AND_CANCEL *(because each receiver has a different public key), making it impossible to detect the reuse of a stamp.*

As for G1, as currently stated, none of the choices work because the stamp authority can enumerate all certificates it has issued. The number of stamps per participant per day is expected to be small for most participants (few hundreds, maybe a few thousand), so it should be feasible for the stamp authority to enumerate and store all possible ENCRYPT$(\{C_S, \texttt{id}, \texttt{date}\})$ *combinations, and compare each* TEST_AND_CANCEL *request's first argument against this list to determine which sender sent the message. That means that choice A would not achieve G1. A similar argument applies for the other choices as well, as presented.*

We leave as an exercise to the reader to decide which of the choices achieve G1, if each choice had been about $\{C_S, \{\texttt{id}, \texttt{date}\}_{S_{priv}}\}$*!*

Alyssa realizes that she needs several computers to run the quota enforcer to handle the daily TEST_AND_CANCEL load. Alyssa finds that storing the num_uses hash table used by TEST_AND_CANCEL on disk gives poor performance because the accesses to the hash table are random. When Alyssa stores this hash table in RAM, she finds that one computer can handle 50,000 TEST_AND_CANCEL RPCs per second on a realistic input workload, including the work required to find the machine storing the key (compared to ≈100 RPCs per second for a disk-based hash table implementation). The network connecting clients to the quota enforcer servers has adequate capacity and is not the bottleneck.

The space required to store stamps in Alyssa's current design is rather large. She decides to save space by storing HASH(stamp) rather than the much larger stamp. With this optimization, storing each cancellation in the num_uses hash table consumes 20 bytes of space. Assume that num_uses only stores stamps that are from today or yesterday. Alyssa purchases computers that have 10^9 bytes of RAM each for storing stamps.

9. [5 points]: Alyssa finds that the peak TEST_AND_CANCEL request rate is 10 times the average. Estimate the number of computers that Alyssa needs for SOS in order to handle 100 billion TEST_AND_CANCEL operations per day. Assume that there are 10^5 seconds in one day. Note that there are multiple potential bottlenecks that you should consider.

(Answer legibly in the space below.)

4,000 COMPUTERS. *Let's first consider the number of RPCs per second. The average request rate is 1 million per second, so to handle the peak rate requires* $\frac{10\times10^6}{50{,}000} = 200$ *computers.*

Now let's consider RAM storage. We need to store cancelled stamps for two days, which is $200 \times 10^9 \times 20$ *bytes. Each computer has* 10^9 *bytes to store stamps, so the number of computers needed =* $200 \times 20 = 4,000$.

The performance of the proposed system depends on the total RAM capacity.

Alyssa builds a prototype SOS system with multiple servers. She runs multiple TEST_AND_CANCEL threads on each server. Alyssa wants each thread to be recoverable and for all cancelled stamps to be durable for at least two days. She also wants the different concurrent threads to be isolated from one another.

Alyssa decides that a good way to implement the quota enforcer is to use transactions. She inserts a call to BEGIN_TRANSACTION at the beginning of TEST_AND_CANCEL and a call to COMMIT just before the call to SEND. She implements a disk-based undo/redo log of updates to the num_uses hash table using the write-ahead log protocol (each disk sector write is recoverable). She uses locks for isolation.

Because all stamp cancellations are stored in RAM, Alyssa finds that a failure purges the entire in-RAM hash table of previously cancelled stamps. A thread could also ABORT at any time before it commits (e.g., the operating system could decide to ABORT a thread).

10. [9 points]: Which of these statements about SOS's recoverability and durability is true?
(Circle True or False for each choice.)

A. True / False When a thread ABORTs, under some circumstances, it must undo some operations from the log.

TRUE. $num\_uses[s]$ *may have been updated, and would need to be backed out.*

B. True / False When a thread ABORTs, under some circumstances, it must redo some operations from the log.

FALSE. *An* ABORT *only needs to undo, not a redo.*

C. True / False The failure recovery process, under some circumstances, must undo some operations from the log.

FALSE. *Upon a failure, the in-RAM database is wiped out. Nothing needs to be undone during recovery.*

D. True / False The failure recovery process, under some circumstances, must redo some operations from the log.

TRUE. *Upon a failure, the in-RAM database is wiped out. To maintain the two-day durability requirement of cancelled stamps from committed transactions, the recovery process needs to redo committed winners from the log.*

E. True / False When the failure recovery process is recovering from the log after a failure, there is no need for it to ACQUIRE any locks as long as no new threads run until recovery completes.

TRUE. *The operations in the log correspond to some serial execution order, by construction, because they were written before the failure by transactions holding the appropriate locks (assuming, of course, that the locking protocol and implementation are bug-free. Because new threads aren't allowed to run until recovery completes, there's no need to* ACQUIRE *any locks during recovery.*

11. [3 points]: Recall that an important goal in SOS is to detect if any stamp is used more than once. Louis Reasoner asserts, "*Alyssa, any reuse of stamps will be caught even if you don't isolate* TEST_AND_CANCEL *threads from each other.*" Give an example to show why isolation is necessary.
(Answer legibly in the space below.)

Suppose the system didn't provide isolation. Suppose that S sends multiple messages to different receivers with the same stamp. Each receiver would call TEST_AND_CANCEL *on the same stamp, causing multiple threads to run. Each thread could execute lines 2 through 4 of the pseudocode on page 6 before any thread executes line 5. S's reuse of the stamp would go unnoticed.*

Satisfied that her prototype works and that it can handle global message volumes, Alyssa turns to the problem of pricing stamps. Her goal is "modest"—to reduce spam by a factor of 10. She realizes that her answer depends on a number of assumptions and is only a first-cut approximation.

12. [4 points]: Alyssa reads various surveys and concludes that spammers would be willing to spend at most US $11 million per day on spam sending costs. She also concludes that 66% (two-thirds) of the 100 billion daily messages sent today are spam.

Under these assumptions, what should the price of each stamp be in order to reduce spam by at least a factor of 10?

(Answer legibly in the space below.)

You don't really need all this space!

Alyssa wants to reduce global spam by a factor of 10, to 6.6×10^9 *messages per day. Because spammers have "only" $11 million a day to spend to send spam, that's the largest amount they can spend on stamps. So, if we price each stamp at* $\$\frac{11\times10^6}{6.6\times10^9} = 0.167$ *cents, and if the quota enforcer were able to do its job, we might be able to reduce worldwide spam by a factor of 10 (assuming, of course, that these numbers are in the right ballpark).*

III KeyDB

Keys-R-Us has contacted you to implement a key-value transactional store, $KeyDB$. $KeyDB$ provides a hash table interface to store key-value bindings and to retrieve the value previously associated with a key.

You decide to use locks for isolation. Lock ℓ_k is a lock for key k, which corresponds to the entry $KeyDB[k]$. A transaction may read or write multiple $KeyDB$ entries. Your goal is to achieve correct isolation for all transactions that use $KeyDB$. Transactions may abort. Assume that ACQUIRE(ℓ_k) is called before the first access to $KeyDB[k]$ and that RELEASE(ℓ_k) is called after the last access to $KeyDB[k]$.

13. [10 points]: For each of the following locking rules, is the rule is **necessary**, **sufficient**, or **neither necessary nor sufficient** to *always* guarantee correct isolation between any set of concurrent transactions?

(More than one may apply for each choice.)

A. Necessary / Sufficient / Neither
ACQUIRE(ℓ_k) anywhere before the first access to $KeyDB[k]$, and
RELEASE(ℓ_k) anywhere after the last access to $KeyDB[k]$.

NEITHER. *The* RELEASE *can't be anywhere. It has to be after all* ACQUIRE*s are done. Moreover, to ensure proper behavior on a transaction* ABORT*, the lock for an updated data item can't be* RELEASE*d before* COMMIT *is done.*

B. Necessary / Sufficient / Neither All ACQUIREs of locks before any other operation, and no RELEASE of a lock before COMMIT if the corresponding data item was modified by the thread.

SUFFICIENT. *This rule is the simple locking protocol.*

C. Necessary / Sufficient / Neither All ACQUIREs of locks before the first RELEASE, and no RELEASE of a lock before COMMIT if the corresponding data item was modified by the thread.

SUFFICIENT. *This rule is the two-phase locking protocol. It is sufficient, but not necessary, because other locking rules that violate two-phase locking may, in particular cases, guarantee isolation.*

D. Necessary / Sufficient / Neither All ACQUIREs of locks in the same order, and no RELEASEs of locks before COMMIT.

SUFFICIENT. *This rule happens to be "sufficient" because the way it is worded makes it identical to two-phase locking. That's because each data access is protected (by assumption above), and no* RELEASE *occurs before all* ACQUIRE*s are done.*

E. Necessary / Sufficient / Neither All ACQUIREs of locks before the first RELEASE, and RELEASE of a lock at any time after last access to corresponding data, and ACQUIREs of any necessary locks if the transaction has to undo during abort.

NEITHER. *This scheme does not provide isolation in general, because another transaction could have updated a variable and committed it, between a transaction releasing a lock and then subsequently aborting.*

14. [8 points]: Determine if each of the following approaches either avoids or eliminates permanent deadlock (with probability approaching 1 as time goes to ∞) between any set of concurrent transactions.

(Circle True or False for each choice.)

A. True / False All ACQUIREs of locks before any other operation, and no RELEASE of a lock before COMMIT.

FALSE. *Two or more transactions could attempt to* ACQUIRE *locks in different orders and deadlock.*

B. True / False All ACQUIREs of locks before the first RELEASE, and no RELEASE of a lock before COMMIT.

FALSE. *Same reason as choice A above.*

C. True / False All ACQUIREs of locks in the same order.

TRUE. *This scheme prevents deadlocks from occurring.*

D. True / False Using a timeout to ABORT any transaction that takes too long to complete, with a random exponential backoff before the transaction is retried.

TRUE. *This scheme ensures that eventually all transactions will make forward progress.*

End of Quiz III and 6.033
Enjoy your summer!