Carnegie Mellon
Computer Science Department.

# 15-712 Fall 2007
# Midterm 1

## Name: _____

## Andrew ID: _____

**INSTRUCTIONS**:

There are 12 pages (numbered at the bottom). Make sure you have all of them.

Please write your name on this cover and put your initials at the top of each page.

If you find a question ambiguous, be sure to write down any assumptions you make.

Be clear and concise. We will grade the first answer provided. If you provide more than one answer to a question, please cross out all but one or otherwise make clear which answer we should grade.

Closed book. 80 minutes.
All questions have the same weight. **NOTE:** We will grade the first 8 questions answered.

List the 8 questions we should grade: _____

| Question | Score (out of 10) |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |
| 10 | |
| Total (out of 80): | |

## O Christmas B-Tree

1. Lehman and Yao proposed a small modification to the B-tree structure to permit more concurrent operations to take place. How did their modified B-tree ensure that operations were still correct under concurrency?

> **Solution:** The solution maintained a linked-list at each level that could be updated atomically. All seek operations could eventually find the correct value by traversing the linked list even if the rest of the tree had not been updated.

How did their modified B-tree provide efficient operations despite the modification? How effective was their change at overcoming the limitations of atomic B-tree transformations?

> **Solution:** Most operations will still use the index to get to the correct item or at least quite near it. In general, the number of extra operations should be less than or equal to the number of concurrent inserts (and in practice, should be smaller than that). The change is therefore quite effective.

What are two drawbacks to their proposed B-tree structure?

> **Solution:** a) It is more complex. It requires careful coding to order updates properly.
>
> b) It does not re-balance the tree upon deletion, but requires a periodic global cleanup pass to do so. As a result, the tree is not guaranteed to remain balanced.
>
> c) It does not guarantee worst-case log(n) tree depth.

## Failure

2. Identify the three major sources of failures that can affect a transactional database, the timescale on which the system must recover from those failures, and one mechanism that the system can use to do so (for each failure mode).

**Solution:** Let's assume that our system should target a reasonable availability of 99.9%. Such uptime allows us $365 * 0.001 * 24 = 8.7$ hours of downtime per year. With that assumption in mind, we can calculate the timescale on which the system must recover (keeping in mind that the system might have to take longer!).

**Disk failure**: These failures destroy not only the running system (which presumably crashes when it discovers it has no disks), but the primary copy of the database. Using proper care (e.g., RAID), these failures can be expected to be fairly rare—perhaps O(yearly) or every few years. As a result, the system can probably afford to spend multiple hours in such a recovery. The only mechanism that sufficies for this is backups to redundant storage, though there are many ways to create such backups.

**System failure/crash**: These failures lose all data stored in memory on the database system. They might be expected to occur on the order of every few months, though these numbers will vary widely. If the OS fails, recovery will be bounded by the amount of time it takes to reboot (call it 10 minutes); it seems reasonable, then, that database recovery should occur within a reasonable factor of that, perhaps up to an hour. The primary technique used to recover from system failures is write-ahead logging.

**Transaction failure**: These failures must roll back an in-progress transaction. The transaction may have changed some data values that need to be restored. Such "failures" are common and must be recovered from nearly instantly. The recovery is handled by standard transactional mechanisms—either by updating only shadow copies of data until the transaction is done, or by recording the old values so that the transaction's effects can be reverted.

Some of what you wrote above probably involves logging in one form or another. Explain briefly one advantage and one disadvantage of physical logging (writing blocks exactly as they are) and logical logging (recording the stream of operations that would change the blocks instead of the actual content of the block).

**Solution:** Physical logging is relatively straightforward and makes recovery simpler, because the database can simply revert the physical pages. It is, however, frequently inefficient—a small update operation that only touched 20 bytes in a page would still result in the entire (4k or larger) page being written to the log.

Logical logging's advantages and disadvantages are the opposite—it is efficient, but can be difficult to reason about what parts of a transaction had been committed to disk (e.g., if the log contains "add 200 to everybody's salary," the DB doesn't know if some of those updates had been applied and others not).

## Mercury in the water

3. "Recovery Management in Quicksilver" [Haskin88] describes a distributed database system with a broader agenda. Specifically, [Haskin88] sets out to offer transactions as a simple-to-use, nearly transparent failure recovery service for all processes in a distributed system. (a) Explain how this might benefit non-database applications. But non-database applications differ from database applications, so a variety of specializations are offered. (b, c, d) Describe THREE such specializations and how they are beneficial.

**Solution:** (a) Haskin88 offers non-database distributed applications an automated recovery method for distributed systems which are exposed to partial failures. By basing it on IPC, the recovery tools can be responsible for identifying what modules are involved in a transaction. All the application has to do is have a local commit/abort structure.

There were several specific commit specializations mentioned in the paper and in lecture. Any three of these answers the question:

(b) 1 phase commit services: simple volatile services which get only one message and then end.

(c) Truncated 2nd phases based on vote: commit read-only if nothing to recover and no interest in second phase.

(c) Transaction graph cycles: first invocation votes for real, the rest give commit read-only.

(d) Transactions continuing after commit: messaging which aren't stateful or part of the last transaction are allowed after the commit starts and cannot cause an abort.

(e) Coordinator migration: rotate the graph such that the most appropriate nodes become coordinator.

(f) Coordinator replication: allows interposed mirrored processing for a replicated coordinator.

## I hear Texas has a part-time parliament, too...

4. As demonstrated by Mike Fischer, Nancy Lynch, and Mike Paterson, a distributed system in an asynchronous environment that reaches consensus cannot both be correct and always terminate (see "Impossibility of Distributed Consensus with One Faulty Process," JACM 1985, which won the 2001 Dijkstra prize—this is sometimes called the FLP impossibility result). Paxos reaches consensus and is correct. Sketch out a scenario where it does not terminate, briefly describing what happens in the prepare and accept phases (phases 1 and 2) to prevent termination.

**Solution:** Two concurrent proposers fight (see Section 2.4 of "Paxos made simple"). Proposer A proposes number 1 in prepare, proposer B proposes number 2. Proposer A attempts an accept, but fails seeing that number 2 has been proposed. Proposer A then proposes number 3. Proposer B attempts an accept, but fails seeing that number 3 has been proposed. Proposer B then proposes number 4. This dual may repeat ad infinitum, preventing termination.

## Those wild, wild writes

5. Hive partitions an OS into cells for fault isolation. In particular, Hive attempts to prevent against wild writes, which can occur when faulty software or hardware attempts to write to the wrong memory page. Give one example of a wild write that Hive prevents:

**Solution:** Hive prevents wild writes using the FLASH memory firewall, which uses a 64-bit vector for each page to track which processors (up to 64 processors or groups of processors) can write to each page. The local OS requests a firewall status change to the remote OS over RPC when a local process maps a remote shared memory page into its address space. The remote OS only maps user pages.

So, e.g., any hardware or software fault that tried to write into remote kernel pages would be prevented. (Of course, this is just one example. We accepted other correct answers.)

Give one example of a wild write that Hive does not prevent:

**Solution:** A corrupted page table for a local process that erroneously referenced a remote shared page may lead to an invalid wild write.

## Istanbul (not Constantinople)

6. Byzantine fault-tolerance in asynchronous environments requires a lot of servers—$3f + 1$ to tolerate $f$ faults. Suppose we want to provide a Byzantine fault tolerant database server to handle payroll in each local office of a large multinational corporation. Rather than deploy $3f + 1$ servers in each office, we could deploy $2f + 1$ servers in each office and use $f$ servers in a remote office to total $3f + 1$ total servers. Using Castro and Liskov's protocol, how would this affect client latencies when all servers are correct? (Hint: what would happen if $f$ nodes had *crashed* rather than just being slow.)

**Solution:** When all servers are correct, Castro and Liskov's protocol executes in four message delays even if $f$ servers have crashed. So you should be able to build a system using this technique without affecting client latencies.

A common misconception was that a Byzantine fault tolerant replicated state machine could continue with only $f+1$ correct and responsive servers in an asynchronous environment—at least $2f+1$ servers are required (though there must be $3f + 1$ servers to guarantee liveness).

How would this affect the reliability of the system compared to deploying $3f + 1$ servers at each local office? What do you think the effect on security would be?

**Solution:** Our initial answer was the following: With $2f+1$ servers at each local office, an adversary can cause problems by compromising any one server at a local office and $f$ servers at a remote office. With $3f + 1$ servers, an adversary would need to compromise $f + 1$ or more servers at each office. In a less-adversarial scenario, if a blizzard cuts off the remote office and one local server crashes, the local office with $2f + 1$ servers cannot run its database but the office with $3f + 1$ (for $f \geq 1$) still can.

The question was a little bit vague (reliability and security in what context?), so we accepted several different answers. For example, some noted that keeping payroll remotely may be a privacy issue, some noted that the remote site could serve as a backup of last resort, and someone even noted that this technique allows tolerance of more faults for a fixed number of servers between two sites. Full credit was given to anything that was well argued.

## Finnegans Wake, featuring Muster Mark

7. In the parliamentary election on October 21st, 2007, the transmission from polling station to tabulation center of votes cast in Geneva for Switzerland's general election were secured using quantum cryptography (provided by a Swiss firm, ID Quantique, of course). Assume, for purposes of this question, that the salient features of quantum crypto are that it is unbreakable and on the bleeding edge of technology. Argue why this choice was likely or not likely to improve security in the election. (Provide concrete reasons for your answer, ideally with reference to material we've examined in the course, hint hint.)

**Solution:** I'll let the blogosphere answer this one – here's what Bruce Schneier has to say (http://www.schneier.com/blog/archives/2007/10/switzerland_pro.html):

"This is so silly I wasn't going to even bother blogging about it. But the sheer number of news stories has made me change my mind.

"Basically, the Swiss company ID Quantique convinced the Swiss government to use quantum cryptography to protect vote transmissions during their October 21 election. It was a great publicity stunt, and the news articles were filled with hyperbole: how the 'unbreakable' encryption will ensure the integrity of the election, how this will protect the election against hacking, and so on.

"Complete idiocy. There are many serious security threats to voting systems, especially paperless touch-screen voting systems, but they're not centered around the transmission of votes from the voting site to the central tabulating office. The software in the voting machines themselves is a much bigger threat, one that quantum cryptography doesn't solve in the least."

A quick read of Ross Anderson's paper ("Why Cryptosystems Fail") describing ATM fraud should have provided the same impression: "The three main causes of phantom withdrawals did not involve cryptography at all: they were program bugs, postal interception of cards, and thefts by bank staff."

A common mistake was to ignore our statement that quantum crypto is unbreakable. We were looking for some evidence that you absorbed the high-level points of the lectures on security.

Consider two pieces of legislation, the first mandating the use of smart cards for all ATM machines, the second making banks liable for phantom withdrawals. Argue which law would make more of an impact on ATM fraud.

**Solution:** See Ross Anderson's paper, "Why Cryptosystems Fail."

## Ticket to ride

8. Consider these two approaches to delegating your access privileges to a server. (1) Give your account name and password, so that it can authenticate to a third party as you, or (2) give it a capability (e.g., your Kerberos ticket) to act as you. Explain why the second approach is superior. Also, propose a situation in which it would not work.

**Solution:** Both approaches allow the server to act as you, but the second approach allows you to limit the time period and rights of what can be done as you. With your password, the server can become you whenever it wants. With a capability, the server can do only what you specified and only for as long as the capability is valid.

The second approach will not work for services that do not include support for capatilities delivered by you. For example, the services you wish to use may not have Kerberos support or may reside in a realm that does not know about you.

## The way we was

9. The nature of the Internet and UNIX around the time of the Internet worm were both helpful in dissemination the worm and helpful in defeating the worm. List two characteristics and how each helped and hurt.

**Solution:** (a) Wide-scale network communication: the propagation of the Worm depended upon network communication. Likewise, much propagation of information and fixes relied upon the same network.

(b) Homogenous systems: many of the attacked systems all had the same security weaknesses, allowing the Worm's simple mechanisms to spread widely. Since the set of weaknesses was small, the set of necessary repairs was also small.

(c) Well-documented holes: the weaknesses exploited by the Worm were all well-known before its release, which made it relatively easy to construst. Likewise, identifying and repairing the weaknesses exploited by the Worm was easier because those weaknesses were already known to exist.

In "The Protection of Information in Computer Systems" [Saltzer75], the authors describe the following attack: "...a masquerader could exactly record the enciphered bits in one communication, and then intercept a later communication and play them back verbatim." (This technique is sometimes called spoofing or a replay attack.)

Although the spoofer may learn nothing by this technique, he might succeed in thoroughly confusing the user or the computer system. The general countermeasure for spoofing is to include in each enciphered message something that is unique, yet predictable, such as _____." There are three common implementation techniques for the blank in this quote. (a) Describe TWO such techniques. (b) Explain how each counters "spoofing." (c) Contrast the costs or limitations of each.

**Solution:** (a1) "just issued challenge nonce" – a message should contain a value that the receiver recently created to be used uniquely in this message.

(b1) This counters spoofing in that the receiver constructs the challenge nonces so that it never reuses the values, so no old message could be replayed and confuse it.

(a2) "synchronized-clock timestamp" – a message should contain the current clock value at a resolution small enough to never be used twice, provided that the receiver can be comfortable determining that the stamp is the current time.

(b2) If the receiver's current time is exactly the message timestamp, and clock values don't repeat, then a timestamped message cannot be replayed. Since perfect synchronization is unlikely, given message skew and clock drift, a receiver has to accept stamps within a window around its clock's value, and maintain a copy of all other messages it has accepted in this window so it can detect replays in the duration of the window.

(a3) "session-specific sequence number" – inside a session with a sequence number, a challenge nonce can be computed by following a rule such as "the last challenge's value + 1", so that only the first sequence number in a series needs to be negotiated.

(b3) Provided sequence numbers never wrap in a session, each message will have a new, unique, yet predictable value for the sequence number, and no old message can be replayed successfully.

(c) Challenge nonces have only a small amount of state and none per message, or per session. But they often require an extra roundtrip message for the receiver to give the sender the challenge nonce. Sequence numbers derive the next challenge nonce from a prior challenge nonce using a pre-agreed

algorithm, but must track state for each session using this algorithm, where there is at least one session per machine pair, and often one session per logical service pair. Timestamps also avoid the extra roundtrip to issue a challenge nonce, but instead have to remember every message within the clock skew + message skew window around the current timestamp, and compare incoming messages to each. This increase in state may be more or less than the state needed for sequence numbers, depending on the skew sizes and the number of sessions likely to be active.

## Trust

10. Attestation is one of the principal tools provided by proposed trusted computing platforms.

    What is attestation?

> **Solution:** Attestation is creating an unforgeable statement by a trusted party that the computer is running a particular piece of code.

What properties does attestation seek to provide / what guarantees does it make? Answer in the context of a distributed system such as a game.

> **Solution:** It seeks to allow the programs to trust that the others are behaving properly. It achieves a proxy of this by asserting that they are running certain code which is assumed to be trusted.

What is the design alternative to attestation, and why do these platforms favor attestation over it?

> **Solution:** The major design alternative is a secure-boot-like solution that restricts the code that is allowed to run on the platform. This solution is used, e.g., in many game consoles so that only "licensed" developers can write games (these developers typically pay heavily for the privilege). Attestation raises fewer concerns of vendor lock-in.
>
> **Alternate:** Many people answered "use a centralized system" and discussed the pros and cons of such an approach. Though this wasn't an intended answer, it seems within the scope of the way we asked this question, so this answer gets credit.

What is a major challenge in using the attestation primitive to build practical trusted systems used by a huge number of people? List two mechanisms you might use to deal with this challenge.

> **Solution:** One of the biggest challenges is knowing what code to trust. There are many possible operating systems, BIOSes, sets of OS patches, and application versions that a user could be running.
>
> - Property attestation, as provded by Nexus OS: Trust a small set of authorities to vouch that programs provide certain properties (e.g., "this video player will not allow copying"). Then set up trust models so that any program that provides the right property is allowed.
> - Vendor attestation: Trust a small set of vendors to sign a certificate for their code, and then assign trust by vendor.