Carnegie Mellon

Computer Science Department.

15-712 Fall 2015
Midterm 2

Name: ____________________

Andrew ID: ____________________

INSTRUCTIONS:

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

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

Be clear and concise.

Closed book. 80 minutes.

We will grade **the first 7 questions answered**. There are 9 questions total: You get to skip two that you don't like. All questions have the same weight (10 pt). For the questions you answer, it is better to partially answer a question than to not attempt it at all.

Only answer 7 of the 9 questions.

Proof Carry Code

1. Necula and Lees paper introduces the concept of proof carrying code (PCC) as a mechanism for safe kernel extensions.

   (1) Describe *three* major advantages of PCC and briefly explain the reason.

   Solution:

   - Eliminates run-time checks: PCC needs to check the binary only once.

   - Tamper-proof: Even tampered binaries follows the safety policy.

   - Safety for code written in any language: The check is at assembly level.

   - Safety properties beyond memory protection: The safety policy is much more flexible than memory protection.

   (2) What is the major practical difficulty of PCC?

   Solution:

   Automatically generating the safety proofs.

   (3) The validation of PCC requires a *safety predicate* and a *safety proof.* What are they and how are they used to certify the safety of programs?

   Solution:

   A safety predicate is essentially a logic form generated from the semantic meaning of a program and constitutes a formal statement that the program, when executed, will not violate any safety checks.

   A safety proof is the proof of a safety predicate, written out in a checkable form, as the proof that the code obeys the safety policy defined by the code consumer (e.g. OS).

   A PCC binary consists of the assembled native code together with an ecoding of the proof of its safety predicate. To validate the binary, the code consumer (e.g. kernel) first extracts the native code and then computes its safety predicate using the VC rules. Then, it checks that the safety proof is a valid proof of the safety predicate.

Exokernel

2. The Exokernel paper evaluates the performance and flexibility on Exokernel systems.

   (1) Compare the extensability approach of Exokernel with either PCC (proof carrying code) or VM (virtual machines). Give one advantage and one disadvantage of Exokernel for your comparison.

Solution:

- Comparing with PCC:

  Advantage: Exokernel does not need to generate safety proof, a major practical difficulty of PCC.

  Disadvantage: PCC is more flexible than Exokernel because the partition of kernel and user level is pre-determined in Exokernel.

- Comparing with VM:

  Advantage: Exokernel maintains a single view of the machine and exposes more hardware resources to applications, which results in more efficient communication and hardware management.

  Disadvantage: VM provides better isolation between guest OSes as all manipulation hardware resources are isolated and protected by VMM (VM monitors).

(2) The storage system of Exokernel needs to be able to handle application-defined metadata layouts. Explain (a) what is the solution provided by the paper and (b) how do user-level and kernel-level components work together to provide this functionality.

Solution:

(a) The solution is UDFs (untrusted deterministic functions). UDFs are metadta translation functions specific to each file type. The kernel storage system uses UDFs to analyze metadata and translate it into a simple form the kernel understands.

(b) The user-level component invoke the kernel-level component with the metadata structure and the disk block. The kernel-level uses the corresponding UDF to translate the metadata structure and enforce necessary protection.

The Scalable Commutativity Rule

3. The scalable commutativity paper discusses how to use commutativity to reason about scalability.

(1) What is the *scalable commutativity rule* introduced in the paper?

Solution:

Whenever interface operations commute, they can be implemented in a way that scales.

(2) The paper introduces the concept of *SIM commutativity*. Explain what it is and contrast it with the conventional notion of commutativity (e.g. for algebraic operations).

Solution:

SIM commutativity is a form of commutativity that is state-dependent, interface-based, and monotonic. It is state-dependent as the commutativity depends on certain system states or arguments. It is interface-based as it requires the resulting states to be indistinguishable via the interface, not the implementation. It is monotonic as it requires any reordering of a prefix of an action does not change commutativity.

SIM commutativity allows conditional commutativity so it exposes many more opportunities to apply the rule to real interfaces than a more conventional notion of commutativity would.

(3) The paper describes four principles for changing POSIX interfaces to make them more scalable. List *three* of them and give an example POSIX interface change from the paper for each principle.

Solution:

1. Decompose compound operations: fork+exec versus posix_spawn
2. Embrace specification non-determinism: lowest FD versus any FD
3. Permit weak ordering: unordered sockets
4. Release resources asynchronously: delayed munmap

Optimistic Concurrency Control

4. Kung introduces optimistic concurrency control (OCC) to replace locks.

(1) Name three disadvantages of locks and how OCC overcomes these disadvantages.

Solution:

- Lock maintenance represents an overhead that is not present in the sequential case.
- There are no general-purpose deadlock-free locking protocols for databases that always provide high concurrency.
- Paging leads to long lock hold times
- Lock cannot be released until end of transaction.
- Locking may be necessary only in the worst case.
- Priority inversion.
- Lock-based programs do not compose: correct fragments may fail when combined.

In contrast, OCC provides high concurrency without the overhead of locking (also no deadlock issue). OCC assumes low contention in normal case, comparing to the worst case assumption of locking.

(2) In OCC, which of the following conditions are *not* sufficient when starting the write phase of transaction B? For all transactions A such that A completes before B, either

(a) A completes its writes before B starts its read phase

(b) A's read set equals B's write set AND A completes its reads before B starts its write phase

(c) A's write set does not intersect B's read set AND A completes its writes before B starts its write phase

(d) A's read set does not intersect B's write set AND A completes its reads before B starts its write phase

(e) A's write set does not intersect B's read or write set AND A completes its reads before B completes its reads

For each insufficient condition, give an example that causes trouble.

Solution:

(B) A's writes can overwrite B's writes. The intersect of their write sets may not be empty and their write phase can still overlap.

(D) A's writes can overwrite B's writes. The intersect of their write sets may not be empty and their write phase can still overlap.

Concurrency Control and Recovery

5. Franklins database survey paper introduces the mechanisms that can be used to support ACID (Atomicity, Consistency, Isolation, Durability) transactions in a database.

   (1) Serializability maintains the isolation property of transactions. Briefly define what makes a schedule of transactions serializable.

   Solution:

   Equivalent to some serial schedule (i.e., a one transaction at-a-time schedule)

   (2) Explain what are the STEAL policy, the NO-FORCE policy, and a Write Ahead Log.

   Solution:

   STEAL: uncommitted transaction can overwrite most recent committed value on non-volatile storage

   NO-FORCE: can commit before updates in non-volatile storage

   Write Ahead Logging: Write log record to non-volatile storage before update data. Transaction committed iff all its log records (incl. commit record) in non-volatile storage.

   (3) The database recovery mechanism (Aries) described in the paper comprises three phases: *Analysis, Redo, and Undo.* Explain the purpose of each phase, and how they are related to your answers in (2).

   Solution:

   Analysis pass: Processes log and find (1) the first update potentially lost during crash and (2) start of oldest in-progress transaction.

   Redo pass: Apply the Write Ahead Log entries that are associated with the transactions that lost data because of the NO-FORCE policy.

   Undo pass: Undo the updates caused by the uncommitted transactions because of the STEAL policy.

Fault-Tolerant State Machine

6. Schneider describes an approach to designing a fault-tolerant service by replicating servers, where each server is a state machine. To make sure all replicating servers process requests in an order consistent with potential causality:

(O1) Each server processes requests by a single client in the order issued.

(O2) If request r by client c could have caused a request r' to be made by client c', then each server processes r before r'.

All requests are to be assigned with unique identifiers (UIDs), where the order of UIDs must conform to O1 and O2. A request is *stable* at a server once no lower-ID request from a correct client can be delivered to the server.

(1) Consider implementing the UIDs with Lamport *logical clocks* or with approximately synchronized *real-time clocks*. For each, describe (a) how O1 and O2 can be fulfilled (are additional assumptions required?) and (b) how to test if a request r is stable.

Solution:

Logical clocks: Assuming messages between a pair of processors are delivered in the order sent. Also, a processor detects a fail-stop processor failed only after it has received the last message from the failed processor. Because the logical clocks are used to generate UIDs, the servers conform O1 and O2 by processing requests based on the order of UIDs. The process order can be guaranteed by processing stable requests only. A request is stable at replica sm_i if a request with larger timestamp has been received by sm_i from every client running on a nonfaulty processor.

Real-time clocks: O1 follows provided no client makes two or more requests between successive clock ticks. O2 follows provided the degree of clock synchronization is better than the minimum message deliver time. Assuming Δ to be constant such that a request r with UID $uid(r)$ will be received by every correct processor no later than time $uid(r) + \Delta$ according to the local clock at the receiving processor. Once the clock on a processor p reaches time τ, p cannot subsequently receive a request r such as $uid(r) < \tau - \Delta$.

(2) What is a major advantage of using logical clocks to implement O1 and O2? What is a major advantage of using real-time clocks?

Solution:

Logical clocks: The process of message is not bounded by the worst case delay Δ in real-time clocks implementation.

Real-time clocks: Its stability test can tolerate Byzantine failures because it can process request after waiting for Δ. In contrast, logical clocks implementation can not make progress if faulty processors refuses to make requests.

(3) A server could be faulty and need to replaced. For a fail-stop failure, describe how the hand over to a replacement server can be done on-the-fly while the system is running when using real-time clocks? How does your answer change if the system could exhibit Byzantine failure?

Solution:

A state machine sm_i integrates a state machine sm_new by sending the values of its state variables and copies of any pending requests to sm_new, and then sends to sm_new every request received during the next interval of duration Δ.

When processors can exhibit Byzantine failures, a single state machine sm_i is not sufficient for integrating a new element into the system. To tolerate t failures in a system with $2t + 1$ state machine replicas, $t + 1$ identical copies of the state information and $t + 1$ identical copies of relayed messages must be obtained.

Paxos

7. A fault-tolerant e-commerce service with three servers A, B, and C uses Paxos for every operation. It has only 4 TVs left in stock on a Black Friday. Now, server A got a request from a client to buy 3 TVs, and server B got a request from another client to buy 2 TVs.

   The initial communications between servers are ("`A->B:M`" indicates A sends a message M to B, and the possible Paxos message types are PREPARE-REQ, PREPARE-RESP, ACCEPT-REQ, and ACCEPT-RESP):

```
A -> A: PREPARE-REQ(1)
A -> B: PREPARE-REQ(1)
A -> C: PREPARE-REQ(1)
A -> A: PREPARE-RESP(1, nil)
C -> A: PREPARE-RESP(1, nil)
B -> A: PREPARE-RESP(1, nil)
A -> A: ACCEPT-REQ(1,"Buy 3 TVs")
```

   (1) Complete the message sequence for a scenario where the servers agree on (i.e., choose) the client's request to A (buying 3 TVs).

Solution:

```
A -> B: ACCEPT-REQ(1,"Buy 3 TVs")
A -> C: ACCEPT-REQ(1,"Buy 3 TVs")
A -> A: ACCEPT-RESP(1,"Buy 3 TVs", OK)
B -> A: ACCEPT-RESP(1,"Buy 3 TVs", OK)
C -> A: ACCEPT-RESP(1,"Buy 3 TVs", OK)
```

   (2) Is it possible that the servers instead agree on the client's request to B (buying 2 TVs)? Justify your answer with the reason (if not) or by completing a message sequence (if it's possible).

Solution:

Yes. (the message sequence can be different)

```
A -> A: ACCEPT-RESP(1,"Buy 3 TVs", OK)
B -> A: PREPARE-REQ(2)
B -> B: PREPARE-REQ(2)
B -> C: PREPARE-REQ(2)
A -> B: PREPARE-RESP(2, nil)
C -> B: PREPARE-RESP(2, nil)
B -> B: PREPARE-RESP(2, nil)
B -> A: ACCEPT-REQ(2,"Buy 2 TVs")
B -> B: ACCEPT-REQ(2,"Buy 2 TVs")
B -> C: ACCEPT-REQ(2,"Buy 2 TVs")
A -> B: ACCEPT-RESP(2,"Buy 2 TVs", OK)
B -> B: ACCEPT-RESP(2,"Buy 2 TVs", OK)
C -> B: ACCEPT-RESP(2,"Buy 2 TVs", OK)
```

   (3) Is it possible that the servers do not agree on either of the requests? If it is possible, how can this be prevented from happening repeatedly?

Solution:

Yes, A and B can keep trying to propose larger value and no one gets the quorum to proceed. The guarantee progress, a distinguished proposer must be selected as the only one to try issuing proposals. The distinguished proposer can be elected with some algorithm such as randomness or timeouts.

Byzantine Generals

8. Recall that in the Byzantine Generals Problem, a commanding general must send an order to $n - 1$ lieutenant generals such that:

IC1: All loyal lieutenants obey the same order.

IC2: If the commanding general is loyal, then every loyal lieutenant obeys the order she sends.

(1) Use an example to explain why 3 generals cannot tolerate 1 traitor.

Solution:

When the commanding general is a traitor, she can send 'attack' to one lieutenant (L1) and send 'retreat' to another lieutenant (L2). L1 and L2 will get conflicting message from each other. L1 can not tell if the commanding general or L2 is the traitor. According to IC2, L1 will attack and L2 will retreat. However, it violates IC1 because L1 and L2 are both loyal but they do not obey the same order. Therefore, 3 generals can not tolerate 1 traitor.

(2) If messages are signed using public key cryptography, can 3 generals tolerate 1 traitor? Justify your answer, referring to IC1 and IC2.

Solution:

When messages are signed using public key cryptography, all generals can verify the authenticity of the message. Therefore, when a lieutenant receives conflicting messages, she can determine who is the traitor. In the 3 generals example, if the commanding general sends 'attack' to L1 and 'retreat' to L2, both L1 and L2 will know the conflicting message is sent by the commanding general. They can follow a common "choice" function to determine their action in this scenario (e.g. choice ('attack', 'retreat') = 'retreat'). Therefore, both IC1 and IC2 are conformed.

Security

9. Anderson's paper discusses how cryptosystems fail, focusing on ATMs as a case study.

(1) One conclusion from the paper is that the bulk of the security R&D budget (at least back in 1994) is for activities that are of marginal relevance to real needs. Use *three* examples from the paper to justify this conclusion.

Solution:

Most R&D budget at that time focused on the technical sophisticated attacks that involves crypt-analysis or the manipulation of transactions. However, the main causes of ATM failures are

- Program Bugs
- Postal Intercept of ATM Cards
- Thefts by Bank Staff
- Observe PINs being typed in & get account number from discarded receipts
- False ATM Terminals
- Programmer arranged for only 3 different PINs
- etc.

(2) To make a system more secure, the paper advocates following four design principles used by safety critical systems. List *three* of them and briefly explain each one.

Solution:

- Specification lists all possible failure modes, including every substantially new accident/incident ever reported
- Explain what strategy has been adopted to prevent (or make acceptably unlikely) each of these failure modes.
- Spell out how each strategy was implemented, including the consequences when each single component fails, including technical factors, training, management issues
- Certification program reviewed by independent experts; test whether the equipment can in fact be operated by people with the stated level of skill & experience, include monitoring program for reporting all incidents.