CSE 486/586 Distributed Systems
Final
Monday, 5/7/12

DIRECTIONS

- Time limit: 3 hours (3:30pm - 6:30pm)
- There are 50 points and 5 bonus points (whatever you score more than 50 will be your bonus points).
- This is a closed-book, no calculator, closed-notes exam.
- The only exception is your cheat sheet (two-sided, letter-sized).
- You should turn in your cheat sheet as well as your answer sheets at the end of the exam.
- Each problem starts on a new page.
- Please use a pen, not a pencil. If you use a pencil, it won't be considered for regrading.
- Each problem also explains its grading criteria. "Atomic" means that you get either all the points or no point, i.e., there are no partial points.

Name:	
UBITName:	

Problem #	Score
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

1. (a) Under what conditions can we design a failure detector that provides both completeness and accuracy?

(**Grading**: atomic 2 points)

Answer:

This is possible in a synchronous environment.

(b) Explain what CAP theorem means.

(**Grading**: atomic 2 points)

Answer:

CAP themrem states that among consistency, availability, and partition tolerance, you can only provide two of them at the same time.

(c) There are two issues with flash disks that Flash Translation Layer tries to overcome. State one of the issues.

(**Grading**: atomic 2 points)

Answer:

Two problems are slow write (due to erasure blocks) and uneven wear-out.

(d) The version of NFS discussed in class uses write-through caching (i.e., writes go all the way to the server's disk), but only at the call of close(). There are one advantage and one disadvantage of this approach. State either one advantage or one disadvantage.

(**Grading**: atomic 2 points)

Answer:

One advantage is that the write performance can be fast. One disadvantage is that the consistency is not guaranteed until close().

(e) Two-phase commit is a blocking protocol. Explain what the biggest disadvantage is.

(**Grading**: atomic 2 points)

Answer:

The biggest disadvantage is latency as every process has to wait.

2. (a) What is the purpose of using a digital signature?
(**Grading**: atomic 3 points)

Answer:

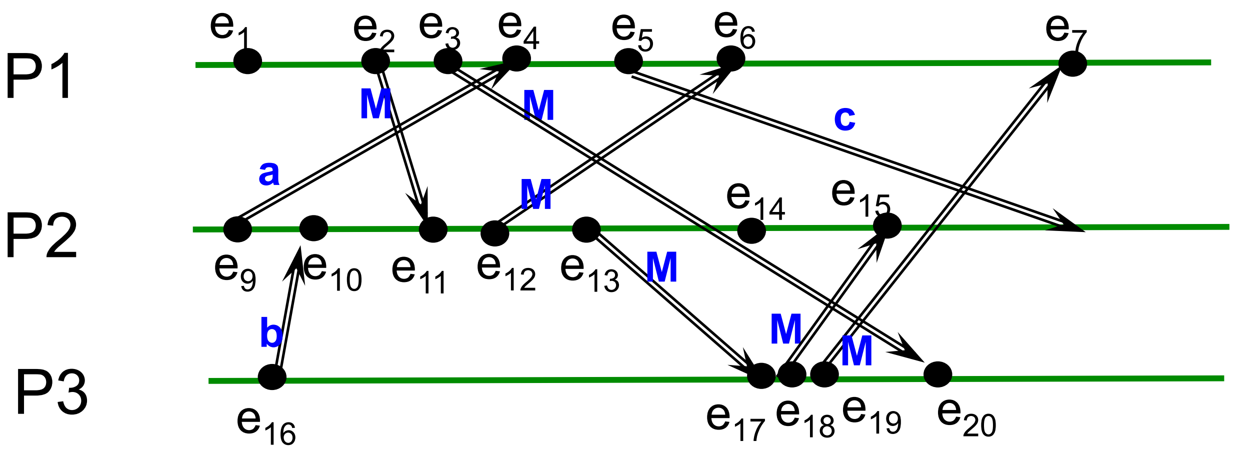
A digital signature provides integrity and authenticity.

(b) How does a digital certificate prove the binding between a public key and an entity that claims that the public key belongs to the entity?
(**Grading**: atomic 3 points)

Answer:

A digital certificate does this through establishing a chain of trust. There are pre-trusted, third-party CAs that sign digital certificates.

3. Using the Chandy and Lamport's snapshot algorithm, **give the snapshot taken by a run of the algorithm below.** Your answer should include the last event recorded for each process (excluding marker-related events) as well as each channel's state. "M" is a marker message in the scenario below.
   (**Grading**: 4 points)



Answer:
P1: $e_1, C_{21} = \{a\}, C_{31} = \{\}$
P2: $e_{10}, C_{12} = \{\}, C_{32} = \{\}$
P3: $e_{16}, C_{13} = \{\}, C_{23} = \{\}$

4. Assume a synchronous system with a fixed number of total processes that runs the bully algorithm discussed in class for leader election. This system has a special "oracle" process that never fails and knows which of the other processes have failed at any point of time. In order to communicate with this oracle process, a process can use an API *call_genie()*, which returns instantaneously without using any resources (e.g., computation or network).

   call_genie() returns one of the following three values:

   - -1 if the calling process is not the process with the second highest node id among all processes
   - 0 if the calling process is the process with the second highest node id among all processes and the leader is alive
   - 1 if the calling process is the process with the second highest node id among all processes and the leader is down

   (a) By adapting the bully algorithm with *call_genie()* and the oracle process, what is the **minimum** number of messages you would need to elect a new leader?
   (**Grading**: atomic 2 points)
   Answer:
   $n-2$. The second highest node can start the election when it gets 1 from *call_genie()*.

   (b) **Explain how you would adapt the bully algorithm** so that you can utilize this oracle process to achieve the minimum number of messages for electing a new leader. An English description should suffice if accurate.
   (**Grading**: 3 points)
   Answer:
   In general, the algorithm should be the same as the bully algorithm. The only exception is that each process has a thread that runs an infinite loop that gets a return value from *call_genie()*. If this value is 1, then the process should be the new leader, in which case it can send the "elected" message right away.

5. Consider two transactions, *T1* and *T2* that contain read and write operations. The following is the definitions of the operations:

   - *read(a)*: returns the value of variable *a*.
   - *write(a, v)*: writes value *v* to variable *a*.

   The transactions are the following:

   - *T1: x0 = read(b); write(a, x0); write(b,100); x1 = read(a); x2 = read(b);*
   - *T2: write(a,200); write(b,200); x4 = read(b);*

   Give two serially-equivalent interleavings of operations other than two trivial serializations of transactions, i.e., it is not acceptable if your answer simply executes $T1$ first and $T2$ next, or vice versa.
   (**Grading**: 4 points)

   Answer:
   Here are two possibilities.

Op	$T1$	$T2$
1	*x0 = read(b)*	
2	*write(a, x0)*	
3	*write(b,100)*	
4	*x1 = read(a)*	
5		*write(a,200)*
6	*x2 = read(b)*	
7		*write(b,200)*
8		*x4 = read(b)*

Op	$T1$	$T2$
1		*write(a,200)*
2		*write(b,200)*
3	*x0 = read(b)*	
4	*write(a, x0)*	
5		*x4 = read(b)*
6	*write(b,100)*	
7	*x1 = read(a)*	
8	*x2 = read(b)*	

6. Consider a transactional system that uses non-exclusive locks (read/write locks). There are four operations a transaction in this system can perform on each lock:

   - *acquire()* allows a transaction to acquire a read lock or a write lock.
   - *release()* allows a transaction to release a read lock or a write lock.
   - *promote()* allows a transaction to convert a *read* lock to a write lock.
   - *demote()* allows a transaction to convert a *write* lock to a read lock.

   In addition, a transaction in this system does not call *acquire()* on any new lock once it has called *release()* on any previously-held lock. However, a transaction can call *promote()* and *demote()* at any time.

   Does this system provide serial equivalence? If the answer is yes, explain why. If the answer is no, construct an execution scenario with *exactly* two transactions.
   (**Grading**: 4 points)

   Answer:
   No. One non-serially equivalenct example is the following:

   With two transactions, T and U,
   U: acquire(write, x)
   U: write(x, 10)
   U: demote(x)
   T: acquite(read, x)
   T: read(x) → 10
   T: release(x)
   U: promote(x)
   U: write(x, 20)
   U: release(x)

7. Using exactly two processes, $P1$ and $P2$, and up to two data objects, X and Y, first construct an execution scenario that is sequentially consistent, but *not* linearizable; then explain why your scenario does not satisfy linearizability but does satisfy sequential consistency. Your execution scenario should use read and write operations running in two processes accessing up to two data objects. Use the following notation of the operations:

   - *read(a)* → *v*: a read operation on variable *a* that returns value *v*
   - *write(a, v)*: a write operation of value *v* on variable *a*

   Again, your answer should have two parts—1) an execution scenario (using the table below) and 2) your explanation.
   (**Grading**: 4 points)

   Answer:
   One scenario that is sequentially consistent, but not linearizable is the following. If this were a linearizable system, operation #3 should return M.

Operation Sequence	$P1$	$P2$
1	write(X, N)	
2		write(X, M)
3	read(X) → N	
4		read(X) → M
5		

8. A startup from UB has recently designed a distributed storage that satisfies a new consistency model called *one-writer-wins consistency* and hired you to advertise the product. In order to understand what one-writer-wins consistency is, we define an ordering, denoted by $<$, as follows: (Every operation is either a read or a write; and it can be issued by any client.)

   (a) If a and b are two operations in a *single process* and a happens before b in the client's program order, then $a < b$.

   (b) If a is a write operation, and b is a read operation that returns the value written by a, then $a < b$.

   (c) For operations a, b, and c, if $a < b$ and $b < c$, then $a < c$.

   (d) $a \nless b$ denotes that two operations a and b do not satisfy the above rules.

   (e) A *conflict* between two write operations is defined as follows: for two operations a and b that write different values to a same data object, if $a \nless b$ and $b \nless a$, then they are in conflict.

   The following is the description of one-writer-wins consistency model:

   - If $a < b$, where a and b are write operations, a and b must be seen in that order (a first then b) by all clients of a storage that satisfies one-writer-wins consistency. For example, suppose that a writes value 0 on object X, b writes value 1 on the same object X, and $a < b$. There can be no client that reads value 1 from object X first then reads value 0 next.
   - If there are conflicting write operations, a storage that satisfies one-writer-wins consistency should detect this conflict, pick one write among all conflicting writes, and apply the result of only that particular write in all replicas eventually.

   Your job is to compare this model to sequential consistency and causal consistency in order to advertise the product properly. Out of these three consistency models, which one is the strongest and which one is the weakest in terms of consistency? **List these three consistency models in the order of strength and explain why.**
   (**Grading**: 5 points)

   Answer:
   Sequential consistency is stronger than one-writer-wins consistency and one-writer-wins consistency is stronger than causal consistency. The first part of the one-writer-wins consistency description actually describes causal consistency; the second part refines causal consistency and describes how to resolve conflicts, which causal consistency does not require. Thus, one-writer-wins consistency is stronger than causal consistency. One-writer-wins consistency is still weaker than sequential consistency because there is no single global order required for all operations.

9. Believe it or not, the CSE 486/586 course website relies on replicated web servers that run the PBFT protocol. There are 4 replicas and one of them is the primary. Unfortunately this system is under attack; some of the replicas might become malicious.

   (a) What is the **minimum** number of faulty replicas that an attacker needs in order to make the system inconsistent (i.e., different, non-faulty replicas execute requests in different orders)? (**Grading**: atomic 2 points)

   Answer:
   2 (from $3f+1$)

   (b) Using the minimum number of faulty replicas that you answered in the above question as well as the PBFT protocol, **construct a scenario that makes the system inconsistent**, i.e., different (non-faulty) replicas execute requests in different orders. Assume that there are two concurrent requests from two different clients in your scenario. Your answer should explain in English what has to happen in each phase of the PBFT protocol for each request, in order for the faulty replicas to make the system inconsistent. Again, there are 4 replicas and one of them is the primary. (**Grading**: 5 points)

   Answer:
   The solution must state that:

   i. One faulty replica has to be the primary and the other faulty is a backup.
   ii. The faulty primary comes up with two different orderings for the operations.
   iii. The faulty primary tells one ordering to one of the two non-faulty backups and a different ordering to the other non-faulty backup.
   iv. The faulty backup now follows the faulty primary in telling different orderings to different non-faulty backups.
   v. Because there are two faulty replicas, the $2f+1$ requirement for PBFT will always be met, and two non-faulty replicas will proceed with different orderings.

10. Some archaeologists have recently discovered that there was an island in East Asia called *Baxo*. What is surprising about this island is that the residents of Baxo, *Baxons*, have developed the exact same knowledge that we modern people know as Computer Science. Not very surprising then, Baxons also developed a consensus protocol. For convenience, we will refer to this consensus protocol developed by Baxons as Baxo.

    By analyzing a document describing Baxo, it has been proven that Baxo is almost identical to Paxos. Specifially, it maintains the same invariant that Paxos maintains:

    > For any *v* and *n*, if a proposal with value *v* and number *n* is issued, then there is a set *S* consisting of a majority of acceptors such that either,
    >
    > (a) no acceptor in *S* has accepted any proposal numbered less than *n* or,
    >
    > (b) *v* is the value of the highest-numbered proposal among all proposals numbered less than *n* accepted by the acceptors in *S*.

    However, there is a critical difference between Baxo and Paxos; it is because Baxons had an ability to design each acceptor in the following way:

    - Each acceptor keeps all proposals that it has accepted.
    - Each acceptor also knows all proposals numbered less than *n* that it *will* accept in the future.

    Given this ability, phase 1 and 2 of Baxo are slightly different from those of Paxos. For your reference, the following is the description of Paxos phase 1 and 2:

    (a) *Phase 1:* A proposer chooses its proposal number *n* and sends a prepare request to acceptors. Acceptors need to reply, 1) a promise to not accept any proposal numbered less than *n* any more, and 2) if there is, the accepted proposal with the highest number less than *n*.

    (b) *Phase 2:* If a proposer receives a reply from a majority, it sends an accept request with the proposal *(n, v)*, where *v* is either the value from the highest *n* among the replies from acceptors, or, if no accepted proposal was returned in phase 1, any value.

    Upon receiving *(n, v)*, acceptors need to either accept it, or, reject it if there was another prepare request with *n'* higher than *n*, and it replied to it.

    (Continue on the next page)

Explain how each phase of the two phases can be different between Baxo and Paxos and why those differences do not make the overall behavior of Baxo any different from that of Paxos. (**Grading**: 6 points)

Answer:
Since the invariants are the same, the overall behavior does not change. In phase 1, an acceptor does not need to return a promise for not accepting anything numbered less than *n*; this is because (from the second difference in the problem description) the acceptor knows all proposals numbered less than *n*. Also, in phase 2, an acceptor does not need to reject any proposal due to the same reason.