Distributed Algorithms, Final Exam

January 14, 2013

Name:
Sciper number:

Time Limit: 3 hours

Instructions:

- This exam is closed book: no notes or cheat sheets are allowed.
- Write your name on *each* page of the exam.
- If you need additional paper, please ask one of the TAs.
- Read through each problem before starting solving.
- Partial credit will be awarded, so explain your thinking carefully.
- State clearly any additional assumptions that you use which are not stated in the question.

Good Luck!

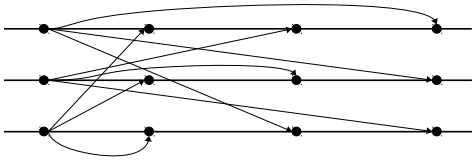
Grading (max points: 50)

Part 1	Part 2	Part 3	Part 4	Total

1 Multiple Choice Questions (15 points)

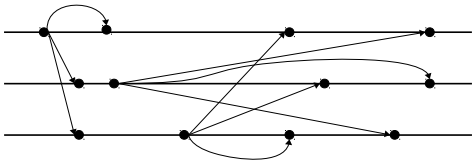
Question 1. (2 points) Consider the broadcast execution below. Which of the following statements are true about this execution?

1. The execution is a correct causal broadcast execution.
2. The execution is a correct total order broadcast execution.
3. The execution is a correct uniform reliable broadcast execution.
4. None of the above.



Question 2. (2 points) Consider the broadcast execution below. Which of the following statements are true about this execution?

1. The execution is a correct causal broadcast execution.
2. The execution is a correct total order broadcast execution.
3. The execution is a correct uniform reliable broadcast execution.
4. None of the above.



Question 3. (2 points) In the course, you saw a round-based concensus algorithm in which each process is leader of one round. A process goes from one round to the next when it receives a message from the leader of the current round or when it detects that the leader of the current round has crashed. Processes maintain a prefered value, which they update when receiving a message from the leader of the current round. A process that is the leader of the current round decides its preferred value and then broadcasts its preferred value. Which of the following are properties of this algorithm?

1. All correct processes decide the same value.
2. If a process decides, then it decides a value that was proposed.
3. If two processes decide, then they decide the same value.
4. All correct processes that decide decide a value that was proposed.

Question 4. (2 points) In the course you saw an algorithm implementing View Synchronous Communication using Uniform Terminating Reliable Broadcast, Best Effort Broadcast, and Group Membership. Which of the following statements are true?

1. The algorithm obtained by replacing Best Effort Broadcast by Uniform Reliable Broadcast solves Uniform View Synchronous Communication.
2. The algorithm obtained by replacing Best Effort Broadcast by Uniform Reliable Broadcast solves View Synchronous Communication.

Question 5. (2 points) Which of the following abstractions are equivalent to consensus in a system with reliable channels?

1. Atomic Commit.
2. Terminating Reliable Broadcast.
3. Total Order Broadcast.
4. Perfect Failure Detector.

Question 6. (5 points) Which of the following properties are safety properties and which are liveness? Mark an "S" or "L" next to each property.

1. For any message m_1 delivered by a process p_1, if there exists another process p_2 that delivered m_1 and then delivered a message m_2, then p_1 also delivers m_2.
2. For every pair of processes p_1 and p_2 that deliver a message m_1, if p_1 delivers a message m_2 before delivering m_1, then p_2 also delivers m_2 before delivering m_1.
3. Every process eventually terminates.
4. Every process terminates before time T.
5. No two processes deliver the same message.

2 Reliable Broadcast (13 points)

Question 1. (1 point) Give the total order property of total order broadcast.

Question 2. (6 points) If an algorithm implements total order broadcast, does it also satisfy the properties of the following?

1. Uniform reliable broadcast
2. Causal broadcast
3. Terminating reliable broadcast

For each of the three (separately), either explain why it does, or give an execution that is allowed by total order broadcast, but is not allowed by the corresponding broadcast abstraction.

Question 3. (6 points) Consider a broadcast algorithm that has the following properties:

- *Validity*: For any two processes p_i and p_j, if p_i and p_j are correct, then every message broadcast by p_i is eventually delivered by p_j.
- *No duplication*: No message is delivered more than once.
- *No creation*: If a message m is delivered by some process p_j, then m was previously broadcast by some process p_i.
- *Causal delivery*: No process p_i delivers a message m_2 unless p_i has already delivered every message m_1 such that $m_1 \rightarrow m_2$.

Does this broadcast algorithm satisfy the *agreement* property (if a message m is delivered by some correct process, them m is eventually delivered by every correct process)? Motivate your answer.

3 View Synchronous Communication (10 points)

The goal of this problem is to implement the View Synchronous Communication abstraction using Group Membership and a new abstraction called Stoppable Broadcast. Recall that View Synchronous Communication combines the properties of Reliable Broadcast and Group Membership as specified in Module 1.

Note that there is no specification in Module 1 for how the Block and BlockOk events should be used. As seen in the course, the Block event is used to request that the application stop broadcasting messages in the current view, while the BlockOk event indicates that the application agreed to stop broadcasting messages in the current view. We assume that the application layer is well behaved: whenever it is asked to stop broadcasting messages it eventually emits a BlockOk event and stops broadcasting message until the next view is installed.

In the course you saw an implementation of View Synchronous Communication that uses Uniform Terminating Reliable Broadcast, Best Effort Broadcast, and Group Membership.

Question 1. (4 points) Give the specification of Uniform Terminating Reliable Broadcast and Group Membership.

Stoppable Broadcast is a new abstraction that you have not seen in the course. It is a broadcast abstraction augmented with the Stop and StopOk events.

As long as no process emits a Stop event, Stoppable Broadcast behaves like Reliable Broadcast. However, processes may request the broadcast to be stopped by emitting a Stop request. In the same spirit as with the Block event of View Synchronous Communication, we assume that when the application layer requests the broadcast to stop then the application layer also stops broadcasting new messages.

Stoppable Broadcast ensures that if all correct processes emit a Stop event, then all correct processes eventually receive a StopOk event. Crucially, when a process p receives a StopOk event, it is guaranteed that no more messages will ever be delivered by p. The specification of Stoppable Broadcast is given in Module 2.

Question 2. (5 points) Taking inspiration from the implementation of View Synchronous Communication seen in the course, give an implementation of Stoppable Broadcast using only Uniform Terminating Reliable Broadcast and Reliable Broadcast. If no process emits a Stop event, your algorithm should use Reliable Broadcast only. Remember that a Terminating Reliable Broadcast instance has a unique sender.

Question 3. (5 points) Implement View Synchronous Communication using only Stoppable Broadcast and Group Membership.

Module 1 View Synchronous Communication

Module:
    Name: ViewSynchronousCommunication, **instance** vsc.

Events:
    Request: $\langle vsc, Broadcast \mid m \rangle$: Broadcasts the message m to all processes.
    Indication: $\langle vsc, Deliver \mid p, m \rangle$: Delivers the message m, whose sender is p.

    Indication: $\langle vsc, View \mid V \rangle$: Installs a new view $V = (id, M)$ with view identifier id and membership M.
    Indication: $\langle vsc, Block \rangle$: Requests that no new messages are broadcast temporarily until the new view is installed.
    Request: $\langle vsc, BlockOk \rangle$: Confirms that no new messages will be broadcast until the next view is installed.

Properties:
    VS1: *View Inclusion*: If some process delivers a message m from process p in view V, then m was broadcast by p in view V.
    VS2 to VS5: The same as properties RB1 to RB4 of Reliable Broadcast.
    VS6 to VS9: The same as properties GM1 to GM4 of Group Membership.

Module 2 Stoppable Broadcast

Module:
    Name: StoppableBroadcast, **instance** sb.

Events:
    Request: $\langle sb, Broadcast \mid m \rangle$: Broadcasts the message m to all processes.
    Indication: $\langle sb, Deliver \mid p, m \rangle$: Delivers the message m, whose sender is p.

    Request: $\langle sb, Stop \rangle$: Requests that the broadcast be stopped. It is assumed that a process calling stop also stops broadcasting messages.
    Indication: $\langle sb, StopOk \rangle$: Confirms that the broadcast has been stopped.

Properties:
    SB1 to SB4: The same as properties RB1 to RB4 of ReliableBroadcast.
    SB5: If every correct process calls stop then every correct process eventually receives a $\langle sb, StopOk \rangle$ indication.
    SB6: When a process receives a $\langle sb, StopOk \rangle$ indication then it stops delivering messages forever.

4 Shared Memory (10 points)

Question 1. (2 points) For shared memory registers, give an execution that is

1. not safe
2. safe, but not regular
3. regular, but not atomic
4. atomic

Question 2. (8 points) Consider the code in Algorithm 3. The code is a variation of the 1-N atomic register implementation. Each process keeps two versions of the data value and its timestamp: val_{new} with timestamp ts_{new} and val_{old} with timestamp ts_{old}. The idea of the algorithm is to detect incomplete concurrent writes. If such a write is detected during a read operation, then the read will return the old value. If no such write is detected, the read updates the old value to the new value and returns it. In order to check whether there is a concurrent write that has not reached all processes, a process first reads the latest timestamp received by all other processes. If there is a process that has an old timestamp ($ts_j < ts_{new}$), then it means that a concurrent write has reached this process, but not process p_j.

Notice that in Algorithm 3 we use the notation from the slides. We consider that the functions are atomic, meaning that you do not need to worry about in-function concurrency problems, such as ts_{new} changing between the beginning and end of the Read handler.

1. Present, at a high level, the 1-N atomic register implementation described in the course. What is the strategy taken by the solution in the course? How does it compare to Algorithm 3? (You should write only up to 5 lines for this answer) (4 points)
2. Does Algorithm 3 solve the 1-N atomic register problem? Please explain. (4 points)

Algorithm 3 1-N atomic register implementation

Write(v) at p_1
  ts++
  send [W, ts, v] **to all**
  for every p_i, **wait until either**:
    receive [ack]
      or
    suspect [p_i]
  return OK

Read() at p_i
  send [ReadTS] **to all**
  for every p_j, **wait until either**:
    receive [ts_j]
      or
    suspect [p_j]

  $writeDone$:= TRUE
  for every ts_j **such that** p_j **is not suspected**
    if $ts_j < ts_{new}$
      $writeDone$:= FALSE

  if $writeDone$ = TRUE
    $val_{old} := val_{new}$
    $ts_{old} := ts_{new}$

  return val_{old}

At p_i
  When p_i **receives** [ReadTS]
    return ts_{new}

  When p_i **receives** [W, ts, v] **from** p_1
    $val_{new} := v$
    $ts_{new} := ts$