Distributed Algorithms 2014
Midterm

October 20, 2014

Name:

Sciper number:

Question 1

We consider a distributed system with processes that can crash. Mark each of the following properties with:

- **S**, if it is a safety property, or
- **L**, if it is a liveness property

1. If a process p delivers a message, then p broadcasts at least one other message. Neither **L** nor **S** (not graded)
2. If a process p delivers a message, then p has already broadcast at least one message. **S** (graded)
3. If a correct process broadcasts a message m, then every process eventually delivers m. Neither **L** nor **S** (not graded)
4. At least one process eventually crashes. **L** (graded)
5. At least one correct process eventually crashes. **S** (point for everyone due to conflicting information given during the exam)
6. If a process p broadcasts a message m, then every correct process delivers m within 10 seconds after m was broadcast by p. **L** (graded)
7. No process invokes operation A before time t. **S** (graded)

Question 2

a) Give the definition of Total-Order Broadcast.

Module

Name: TotalOrderBroadcast, instance *tob*.

Events

Request: $\langle$ *tob*, Broadcast | m $\rangle$: Broadcasts a message m to all processes.

Indication: $\langle$ *tob*, Deliver | p, m $\rangle$: Delivers a message m broadcast by process p

Properties:

TOB1: *Validity:* If a correct process p broadcasts a message m, then p eventually delivers m.

TOB2: *No duplication:* No message is delivered more than once.

TOB3: *No creation:* If a process delivers a message m with sender s, then m was previously broadcast by process s.

TOB4: *Agreement:* If a message m is delivered by some correct process, then m is eventually delivered by every correct process.

TOB5: *Total order:* Let m_1 and m_2 be any two messages. Let p be any correct process that delivers m_1 without having delivered m_2. Then no correct process delivers m_2 before m_1.

b) Give the definition of Consensus.

Module

Name: Consensus, instance *co*.

Events

Request: $\langle$ *co*, Propose | v $\rangle$: Proposes value v for consensus.

Indication: $\langle$ *co*, Decide | v $\rangle$: Outputs decided value v of consensus.

Properties:

CO1: *Termination:* Every correct process eventually decides some value.

CO2: *Validity:* If a process decides a value v, then v was proposed by some process.

CO3: *Integrity:* No process decides twice.

CO4: *Agreement:* No two correct processes decide differently.

c) Recall the Consensus-Based algorithm for Total-Order Broadcast from the lecture. It transforms a consensus abstraction (together with a reliable broadcast abstraction) into a total-order broadcast abstraction. Describe a transformation in the other direction, that is, implement a consensus abstraction from a total-order broadcast abstraction.

Implements:
Consensus, **instance** *co*.
Uses:
TotalOrderBroadcast, **instance** *tob*.

upon event <*co, Init*> **do**
decided := `false`;

```
upon event <co, Propose | v> do
  trigger <tob, Broadcast | v>;

upon event <tob, Deliver | p, v)> do
  if decided = false then
    decided := true;
    trigger <co, Decide | v>;
```

Question 3

Given the following interface and properties of FIFO-order (reliable) broadcast:

Module

Name: FIFOReliableBroadcast, instance *frb*.

Events

Request: $\langle$ *frb*, Broadcast | m $\rangle$: Broadcasts a message m to all processes.

Indication: $\langle$ *frb*, Deliver | p, m $\rangle$: Delivers a message m broadcast by process p

Properties:

FRB1: *Validity:* If a correct process p broadcasts a message m, then p eventually delivers m.

FRB2: *No duplication:* No message is delivered more than once.

FRB3: *No creation:* If a process delivers a message m with sender s, then m was previously broadcast by process s.

FRB4: *Agreement:* If a message m is delivered by some correct process, then m is eventually delivered by every correct process.

FRB5: *FIFO delivery:* If some process broadcasts message m_1 before it broadcasts message m_2 , then no process delivers m_2 unless it has already delivered m_1.

a) Implement FIFOReliableBroadcast using Reliable Broadcast.

For implementing FIFO reliable broadcast there are multiple solutions. Here is presented the solution with the sequence number.

Implements:
FIFOReliableBroadcast, **instance** *frb*
Uses:
ReliableBroadcast, **instance rb**

upon event <*frb, Init*> **do**
$lsn := 0$;
$pending := \emptyset$;
$next := [1]^N$;

upon event <*frb, Broadcast* | m> **do**
$lsn := lsn + 1$;
trigger <*rb, Broadcast* | $[DATA, self, m, lsn]$>;

upon event <*rb, Deliver* | *p, [DATA, s,m, sn]*> **do**
$pending := pending \cup \{(s, m, sn)\}$;
while exists $(s, m', sn') \in pending$ such that $sn' = next[s]$ do
$next[s] := next[s] + 1$;
$pending := pending \backslash \{(s, m', sn')\}$;
trigger <*frb, Deliver* | s, m'>;

b) Give the definition of Causal Broadcast.

Module

Name: CausalOrderReliableBroadcast, instance frb.

Events

Request: $\langle crb, Broadcast|m\rangle$: Broadcasts a message m to all processes.

Indication: $\langle crb, Deliver|p, m\rangle$: Delivers a message m broadcast by process p

Properties:

CRB1: *Validity:* If a correct process p broadcasts a message m, then p eventually delivers m.

CRB2: *No duplication:* No message is delivered more than once.

CRB3: *No creation:* If a process delivers a message m with sender s, then m was previously broadcast by process s.

CRB4: *Agreement:* If a message m is delivered by some correct process, then m is eventually delivered by every correct process.

CRB5: *Causal delivery:* For any message m_1 that potentially caused a message m_2, i.e., $m_1 \rightarrow m_2$, no process delivers m_2 unless it has already delivered m_2.

c) Give a non-blocking algorithm that implements causal broadcast, such that:

- Your algorithm only uses the FIFOReliableBroadcast abstraction as underlying module.
- Even if every correct process broadcasts an infinite number of messages, the message sizes do not grow indefinitely.

Implements:
ReliableCausalOrderBroadcast, **instance** *rco*
Uses:
FIFOReliableBroadcast, **instance** *frb*.

upon event <*Init*> **do**
delivered := $\emptyset$;

upon event <*rcoBroadcast* , *m*> **do**
trigger <*frbBroadcast*, *m*>;

upon event <*frbDeliver* | *m*> **do**
If $m \notin delivered$ **do**
trigger <*frbBroadcast*, *m*>;
trigger <*rcoDeliver*, *m*>;
delivered = *delivered* $\cup$ {m};