Chalmers Techniska Högskola — CTH TDA297
Data- och Informationsteknik — GU DIT290

Distribuerade system fk

Tentamen 2020-08-21

Dag, *Tid, Sal*: March 21s 2020, 14:00-18:00, Home sweet home

***Kursansvarig*:** Philippas Tsigas (Tel: 772 5409)

Totalt *Poängtal*: 60

***Betygsgränser*:**

CTH: 3:a 30 p, 4:a 38 p, 5:a 48 p

GU: Godkänd 30p, Väl godkänd 48 p

Instructions

- Please answer in English, if possible.
  If you have very big difficulty with that, though, you may answer in Swedish.
- **Do not forget to write your personal number and if you are a GU or CTH student and at which "linje".**
- Please start answering each assignment on a new page; number the pages and use only one side of each sheet of paper.
- Please write in a tidy manner and explain (Clearly) your answers.

LYCKA TILL !!!!

1. (10 points) A connected bidirectional asynchronous network of n processes with identities has diameter D and may contain zero or more "special" processes. Suppose that all processes wake up at time 0 and start whatever protocol we have given them. Suppose that each process initially knows whether it is special, and knows the identities of all of its neighbors. However, the processes do not know the number of processes n or the diameter of the network D. Give a protocol that allows every process to correctly return the number of"special" processes. Your protocol should only return a value once for each process (no converging to the correct answer after an initial wrong guess). Discuss and prove correctness and the me time complexity of your algorithm.

   Clarification: You get 10 points for a correct answer and 5 extra points if your algorithm terminates in $O(D)$ time.

2. (10 points) Suppose that we modify the problem of synchronous agreement with crash failures so that instead of crashing a process forever, the adversary may jam some or all of its outgoing messages for a single round. The adversary has limited batteries on its jamming equipment, and can only cause f such one-round faults. There is no restriction on when these one-round jamming faults occur: the adversary might jam f processes for one round, one process for f rounds, or anything in between, so long as the sum over all rounds of the number of processes jammed in each round is at most f. For the purposes of agreement and validity, assume that a process is non-faulty if it is never jammed. Can we modify the round based algorithm that we discussed in the course to work in this model? If yes, as a function of f and n, how many rounds does it take to reach agreement in the worst case in this model, under the usual assumptions that processes are deterministic and the algorithm must satisfy agreement, termination, and validity? If no, provide a proof.

3. (10 points) Moa is implementing a shared counter object that support two operations `inc` and `read`, where `read` returns the number of previous `inc` operations. Her system is an asynchronous message-passing system subject to $f < n/2$ crash failures. Moa designed the algorithm described in Figure 1. In the algorithm, each process i maintains a vector c_i of contributions to the counter from all the processes, as well as a nonce r_i used to distinguish responses to different read operations from each other. All of these values are initially zero. Show that the implemented counter is linearizable, or give an example of an execution where it isn't.

4. (10 points) Conflict-free Replicated Data Types, eventual consistency and strong eventual consistency:

   1. Define eventual consistency and strong eventual consistency in the context of Replicated Abstract Data Types.
   2. What was the motivation that lead to the introduction of Conflict-free Replicated Data Types?
   3. What is the consistency that they guarantee.
   4. Describe, by providing pseudocode and an informal description, an example of a CRDT implementation. Provide also a proof of its correctness.

5. (10 points) Consider the dinning philosophers problem, n philosophers $P_1, \ldots, P_n$ in a bidirectional ring. Moa wants to help them eat and not starve. She assigns n priorities to them in the following way: P_1 has the lowest priority, P_2, has the second lowest, and so on up to P_n who has the highest priority. Moa instructs each philosopher to seek forks in parallel and to use their priorities to resolve

```
1  procedure inc
2      c_i[i] ← c_i[i] + 1
3      Send c_i[i] to all processes.
4      Wait to receive ack(c_i[i]) from a majority of processes.
5  upon receiving c from j do
6      c_i[j] ← max(c_i[j], c)
7      Send ack(c) to j.
8  procedure read
9      r_i ← r_i + 1
10     Send read(r_i) to all processes.
11     Wait to receive respond(r_i, c_j) from a majority of processes j.
12     return ∑_k max_j c_j[k]
13 upon receiving read(r) from j do
14     Send respond(r, c_i) to j
```

Figure 1: Moa's shared counter implementation

conflicts, i.e. the philosopher with the highest priority gets the respective fork when hungry and competing with a neighbor. Moa is not sure whether her protocol is correct and cannot find a way to calculate its complexity.

1. Can you help Moa to prove or disprove the correctness of her protocol by providing a proof or a counterexample?
2. If you prove that the protocol is correct please provide a complexity analysis of the protocol.
3. If you prove that the protocol is not correct, can you extend the protocol, by wrapping it around with another protocol leyer, without modifying Moa's part (you do not want to hurt Moa's feelings :)), to make it correct? What is the complexity of the algorithm?
4. Can you design another algorithm from scratch that solves the dinning philosophers problem that has better time complexity?

6. (10 points) Moa is building a database that is replicated on N machines. To access the database, a client accesses any of the replicas. The communication between clients and the replicas and between the replicas themselves is reliable, point-to-point, and FIFO-ordered. However, the communication delays can vary significantly. Moa has designed the following variant of a totally-ordered multicast algorithm to be used on her system:

   Each replica maintains Lamport's logical clock, and every inter-replica message is stamped with the unique id of the sender upon transmission. Whenever a replica receives a database

update message from a client, it brodcasts the update in a message to all replicas (including itself). Whenever a replica receives an update message from another replica (or from itself) it puts the message into a local queue, and acknowledges the reception of the update by sending an acknowledgment message to all other replicas (including itself). The replica applies an update from its local queue to its local database if and only if:

i) the update message has the lowest timestamp among the messages in the replica's local queue, and

ii) the update message has been acknowledged by a quorum of N

After a replica has applied the update to the local database, the update message and its acknowledgments are removed from the local queue. If later another acknowledgment arrives for the message, such a late acknowledgment is simply dropped from the queue.

Prove that the above algorithm implements totally-ordered causal multicast (i.e. satisfies the following requirements: 1. Reliability: Integrity, Validity, Agreement 2. Ordering: Causal, Total-order) or produce a counter example.