CS60002: Distributed Systems

Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur
Spring 2021

Test - 1
February 23, 2021
Time: 2 pm to 3-30 pm

Instructions (Read carefully before starting)

1. Write on plain paper (white). Final answer must all be handwritten, nothing should be typed.
2. At the end of the examination, scan all the pages in order in a **single** pdf. All the pages should be clearly readable in the pdf.
3. Name the pdf file in the following format: <your roll no>_<first name>.pdf. For example 16CS30001_Arijit.pdf
4. Mail the pdf file to pranav.23890@gmail.com. Note that the address is different from the one you sent the assignment to.
5. **The pdf file must reach the TA within 5 minutes of the end of the examination**. So make sure you leave enough time for scanning, mailing etc., and check that you are mailing to the correct address. Anything reaching the TA beyond the 5 minutes will incur (exponentially increasing with delay) penalty. The amount of penalty will be completely decided by me, you cannot negotiate with me, so do not submit late.

Answer ALL Questions

Unless otherwise mentioned, assume all systems are asynchronous, completely connected, with reliable links, and all nodes/processes have unique ids.

1. Suppose in Lamport's logical clock, each node i increments its clock by a fixed positive constant d_i on an event (instead of incrementing by 1)? Will the clocks work correctly if the d_i's are different for different i? Justify your answer. (5)
2. The k-exclusion problem is similar to the mutual exclusion problem, but here at most k nodes can be in the critical section at the same time, for some predefined constant k. Design an algorithm to achieve k-exclusion in a distributed system. If you start with any existing algorithm taught in class, assume that I know the algorithm and just list clearly what changes are needed to it. If you design any new algorithm, list the steps of the algorithm clearly, no need to write send/receive handlers for all messages. Argue in brief why no more than k nodes can be in the critical section at the same time in your algorithm. (5)

3. Prove that the consensus problem cannot be solved with three processes of which at most one of the process can be byzantine faulty (Prove anything you use from scratch, do not assume any known result). (5)
4. Consider a leader election algorithm for a synchronous unidirectional ring in which nodes send their ids around the ring similar to Lelann-Chang-Roberts algorithm. However, for a node with id *u*, any message initiated by *u* travels 1 hop every 2^u rounds (i.e., intermediate nodes will hold the message for 2^u rounds before forwarding it if it is to be forwarded). Write the complete algorithm (with any other changes if needed) so that it elects a leader at the end. What is its message and time complexity (give arguments/calculations to show how you got the values, no marks will be given without that)? (10)
5. Consider a synchronous, bidirectional ring. In a fault-free state, a token circulates in one direction only in the ring. A node may use the token when it gets it if it needs to, else it sends it to the next node in the ring. Nodes may exhibit crash fault (so the ring may be broken) and recover (and the ring may be formed again), but links are reliable and never go down. It is also given that at most one node can be in the crashed state at any time, the failure of a node can be instantaneously detected by its two neighbors in the ring (do not worry how), and message delays are zero.
    a. Design an algorithm for the case when it is assumed that the node holding the token never crashes. The token should continue to circulate in some manner to all non-failed nodes even when a node has crashed. Your algorithm must ensure that after a crashed node recovers, eventually the token must circulate in the ring normally (any one direction, clockwise or anticlockwise, is fine) if no further failures happen. Assume that the nodes are numbered from 1 to *n* in the clockwise direction. Write the code for a node *k* clearly, stating the messages first and then the receive/send rules for the messages. You can assume the existence of a predicate *failed(i)* which returns (do not worry how) *true* if node *i* has crashed, and *false* if *i* is alive (so any neighbor of a node *i* can just check the value of *failed(i)* to see if it has crashed or not, no messages need to be sent for this). (10)
    b. Now write clearly (list the steps, exact handlers not needed) how you will extend the above algorithm to handle the case when the node holding the token can also crash. You can assume that when a node holding the token crashes and then recovers, the token is wiped out (i.e., on recovery, the node will no longer have the token). (5)