CS60002: Distributed Systems
Spring 2025
Assignment – 1

Due: February 5, 2025 (submit in moodle)
Answers must be typed mostly and finally a single pdf document should be submitted. You can insert scans of hand-drawn pictures if really needed.

1. Write clear and structured pseudocode for all algorithms, in the form discussed in class.
2. Use concise yet descriptive variable and function names.
3. Avoid implementation-specific syntax (e.g., Python, C++).
4. Include comments in the pseudocode to explain key steps.
5. Beside the variables mentioned in a problem (if any), you can add any other variable you want.
6. If you make any other assumptions (over and above what is given) in a problem, please write them clearly at the beginning of your answer.

1. Consider a system with a clock that runs slow with a drift rate of 10^{-5}. Cristian's algorithm is used to synchronize the clock from a time server every 30 minutes. To estimate the delay in getting the time, a simple algorithm is used – the system just measures the round trip delay and divides it by 2. The maximum delay of the link between the system and the time server is 2 milliseconds (one way). The processing time at the client after receiving the time from the server is 1 millisecond. Assume that the time server's clock has zero drift. What is the maximum possible difference between the two clocks (the time server and the client system) just before and just after a synchronization? Show all calculations.

   [Solution sketch]
   Just after the synchronization:
   If the processing time was 0, as per class, the max difference would be 1 millisec. If we assume that the client's clock is set at the end of the processing time, and because the server has no drift, the difference due to drift of client is to be considered to see if it has any effect. Consider a scenario: Client's request takes 2 ms to reach server, the server sends x, and the message reaches the client in 0 ms. Following class, client sets, at the end of 1 ms processing delay, its time to (x + 1) ms. For the 1 ms processing time, the server's clock also advances by 1 ms, so becomes (x+1) ms. So there is no effect and no difference (the drift during the processing time is offset by the clock being set at the end). On the other hand, if the request took 0ms to reach server, and server response took 2 ms to reach client, client will still set its time at the end of processing delay to (x+1) ms, but server would have been at (x+3) ms (the 2 ms message travel time plus the processing delay of 1 ms)

   You may still have 1-2 questions here. Let me see if anyone asks.

   Just before the synchronization:
   It will be the (max difference just after sync) + drift in 30 minutes = 2 ms + 12 seconds.

2. Suppose that 3 nodes always have their clocks synchronized to within 1 second of each other. A simple global snapshot algorithm is proposed in which each node takes its local snapshot at the hour, i.e., at 9 AM, 10 AM, ... as per its local clock; no other messages are exchanged. If we take a global state to be the the set of local snapshots of the 3 nodes taken at say 10 AM, is it be

guaranteed to be a consistent global state? Justify your answer clearly (Prove if yes, give a counterexample if no).

[Solution sketch] No. Say Node 1's clock is 1 second ahead of node 2. Node 1 takes snapshot at 10 am, then sends a message to Node 2. The message tales less than 1 second to reach node 2. Node 2 has not reached 10 am yet, so when it reaches 10 am and takes the snapshot, the receive of the message is included. This will be an orphan message, so not consistent.

3. Write a distributed algorithm for constructing a spanning tree rooted at a specific node in an asynchronous, reliable system. The root node will initiate the algorithm. At the termination of the algorithm, each node should know its parent and children set in the spanning tree. In addition, the root node should know that the spanning tree has been constructed (i.e., the algorithm has terminated). The model is asynchronous, reliable communication, arbitrary topology. **(*For your practice, already discussed in class, this will not be graded*)**.

   [Solution sketch] Whole idea done in class.

4. Suppose you are given a connected, undirected graph with a special node *i* and an upper bound *T* on the message delay over any link. Design a global state collection algorithm that does not rely on FIFO channels and does not use piggybacking. At the end of your algorithm, the global state should be available at node *i* and *i* should know that state collection is complete. Analyze its message complexity. Clocks in the nodes are not synchronized, but you can assume that they have no drift. Assume that every process knows an upper bound on the no. of processes in the system, and processing times at nodes are negligible. Communication is reliable.

   [Solution sketch] Use Chandy-Lamport, with the following small change. On deciding to take a local snapshot, just stop sending messages on all outgoing channels for T time (or slightly larger than T). This ensures all pre-marker messages reach and channels become empty. Then send the marker. Then again don't send any messages for T time (ensures no message sent after the marker reaches before the marker). Then resume normally. This does not require any change in the receiver code.

5. Design a permission based distributed mutual exclusion algorithm in which a node will require 0 messages per critical section entry in the best case. However, in the worst case, it will require 2(n-1) message per critical section entry similar to Ricart-Agarwala's algorithm. Identify what the best and the worst cases are. The proposed algorithm need not maintain the fairness property of Ricart-Agarwala's algorithm.

   [Solution sketch] This Raicairol-Carvalho algorithm. Basically once a node receives a reply, it assumes that the sender node has given its permission to the receiver node until the sender requests again. Won't cause starvation, but will not be fair like Ricart-Agarwala in the sense that requests with higher timestamps can sometimes go in before requests with lower timestamps. Best case is when one node requests again and again, then it will require 2(n-1) messages the first time and then 0 messages per CS entry thereafter. Worst case is when all processes wants to enter CS.

6. Design an asynchronous distributed algorithm to find a BFS spanning tree from a designated node *X* to all other nodes in an undirected, connected graph. *X* will be the only node to initiate the algorithm, and no flooding should be used. At the end of the algorithm, each node should have a local variable *Level* whose value set to its shortest path length from *X*. The BFS spanning tree should be built level by level, i.e., nodes at any level x should be added to the tree only after

all nodes at level x-1 have been added to the tree (X should synchronize this). What is the message complexity of the algorithm? Do not worry about how *X* gets to know when all the values are correctly computed at the nodes (i.e., about detecting termination of the algorithm at X). Communication is reliable.

[Solution sketch] Will be circulated by TAs. Just a lot of details.

7. Design an asynchronous distributed algorithm to find a DFS spanning tree in an undirected, connected graph. Nodes have unique id. The algorithm may be started by more than one node acting as the initiator (root), each starting independently at any time. However, if the spanning tree building of one or more initiators overlap in time, finally only one DFS spanning tree should exist, specifically the one built by the initiator with the maximum id among all such initiators.

   [Solution sketch] See slides given with this.

8. Design a token based mutual exclusion algorithm that uses a rooted spanning tree to pass tokens along the edges of the spanning tree. The spanning tree is defined by a "*to_token*" variable at each node which points to its parent in the tree, the value for the initial root is set to itself. The token is initially at the root of the spanning tree built. The root is allowed to be changed over time if needed (while maintaining a proper rooted tree), but the underlying unrooted tree remains the same. You do not have to construct the spanning tree, assume that it exists (i.e., the *to_token* variables are set properly). If there is no pending request in the system, the token should stay with the node that used it last. A node that wants to use the token should have to request for it when it needs it. Each node in the tree should be in charge of requesting for and use of the token on behalf of all nodes in the subtree rooted at it, including itself. Use the ideas discussed in class to write the algorithm clearly. What do you think is the no. of messages needed for CS entry for your algorithm? Model is asynchronous, reliable, nodes have unique ids.

   [Solution sketch] See Raymond's tree-based mutual exclusion algorithm (paper given)