CSE 486/586 Solutions for Non-Graded Practice Problem Set 1

Acknowledgement: This problem set heavily contains the material developed and copyrighted by Prof. Indranil Gupta at Illinois.

1. Can we design a complete and accurate failure detector? Why or why not?

    a. For synchronous systems, the answer is yes. For asynchronous system, the answer is no because it is impossible to distinguish delay/loss of messages from faulty processes.

2. Calibrations on a recent version of the Linux operating system showed that on the client side, there is a delay of at least 1.3 ms for a packet to get from an application to the network interface, and a similar minimum delay for the opposite path (network interface to application buffer). The corresponding minimum delays for the server are 0.15 ms and 0.35 ms respectively. What would be the accuracy of a run of the Cristian’s algorithm between a client and server, both running this version of Linux, if the round trip time measured at the client is 5 ms?

    a. The earliest point at which the server could have placed the time in message mt was min1 = 1:3 + 0:35 = 1:65 ms after the client dispatched message mr. The latest point at which the server could have done this was min2 = 0:15 + 1:3 = 1:45 ms before message mt arrived at the client. The time by the server’s clock when mt arrives at the client is therefore in the range: [t +min2; t + RTT - min1], where t is the time placed by the server in mt. The width of this range is (RTT - min1 - min2) = 1.9 ms, and the accuracy is thus: (RTT- min1 - min2)/2 = 0:95ms.

3. Problem 14.2 from the textbook.

    a. It is undesirable to set the clock back to right time because a process may implicitly depend on monotonically increasing values for time. For example, the make command checks if recompilation is necessary based on the the update timestamps of its dependencies.

    Current time according to the clock: 10:27:54.0 (hr:min:sec)
    Current actual time: 10:27:50.0
    After 8 seconds both clocks should read: 10:27:58.0

    Therefore, to adjust the clock we need to gain only 4 seconds in the software clock versus 8 seconds in the hardware clock. So, we need to move at 4/8 = 0:5 the pace of the hardware clock over the next 8 seconds.

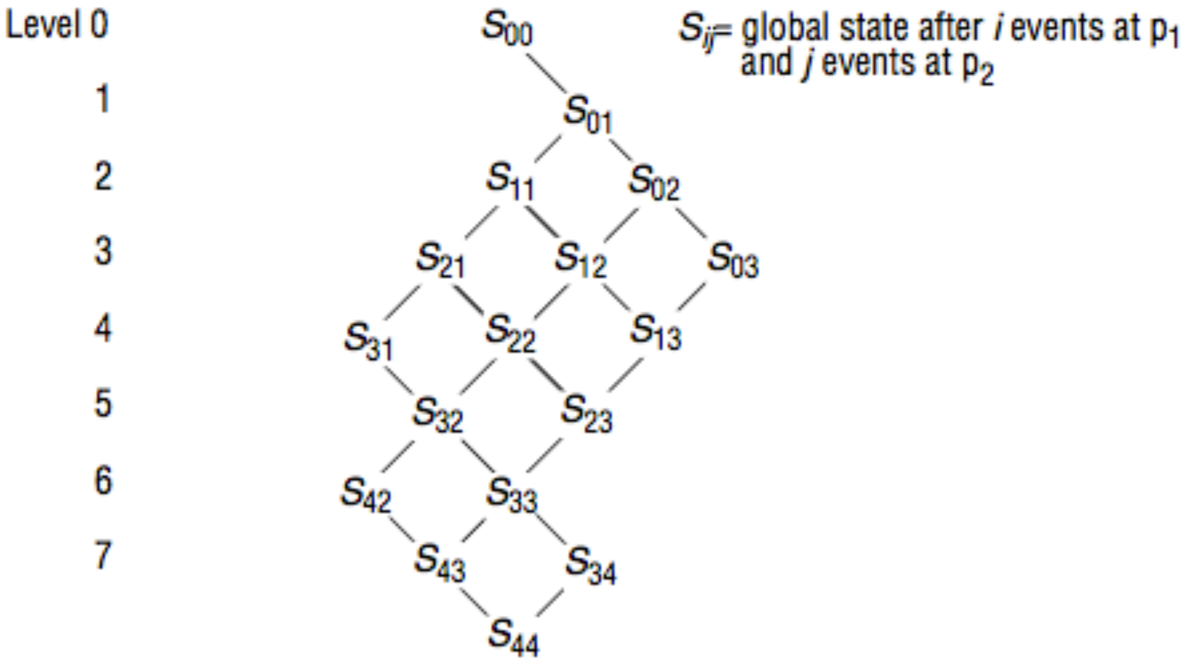
4. Problem 14.7 from the textbook.

    a. Let *a* = *Ti-2 – Ti-3* = 23.48 - 13.43 = 10.05; *b* = *Ti-1 – Ti* = 25.7- 15.725 =9.975. Then the estimated offset *oi* = (a+b)/2 = 10.013s, with estimated accuracy = ±di ⁄ 2 = ± (a-b)/2 = 0.038s (answers expressed to the nearest millisecond).

5. Problem 14.15 from the textbook.

    a.

The lattice of global states for the execution of Figure of exercise 11.15



6. In the figure below, if all processes start with sequence numbers or vector timestamps (as applicable) containing all zeroes, for each of the following algorithms, mark the timestamps at the point of each multicast send and each multicast receipt. Also mark multicast receipts that are buffered, along with the points at which they are delivered to the application.

a. FIFO ordering algorithm discussed in class

i. P1:

Send, [1, 0, 0]

Send, [2, 0, 0]

Accept, [2, 0, 1]

Send, [3, 0, 1]

Accept, [3, 0, 2]

Accept, [3, 1, 2]

ii. P2:

Accept, [0, 0, 1]

Buffered as [2, 0, 1], Timestamp still [0, 0, 1]

Send, [0, 1, 1]

Accept, [1, 1, 1]

Accept, [2, 1, 1] (Previously buffered)

Accept, [2, 1, 2]

Accept, [3, 1, 2]

iii. P3:

Accept, [1, 0, 0]

Send, [1, 0, 1]

Accept, [2, 0, 1]

Accept, [2, 1, 1]

Send, [2, 1, 2]

Accept, [3, 1, 2]

b. Causal ordering algorithm discussed in class

i. P1:

Send, [1, 0, 0]

Send, [2, 0, 0]

Accept, [2, 0, 1]

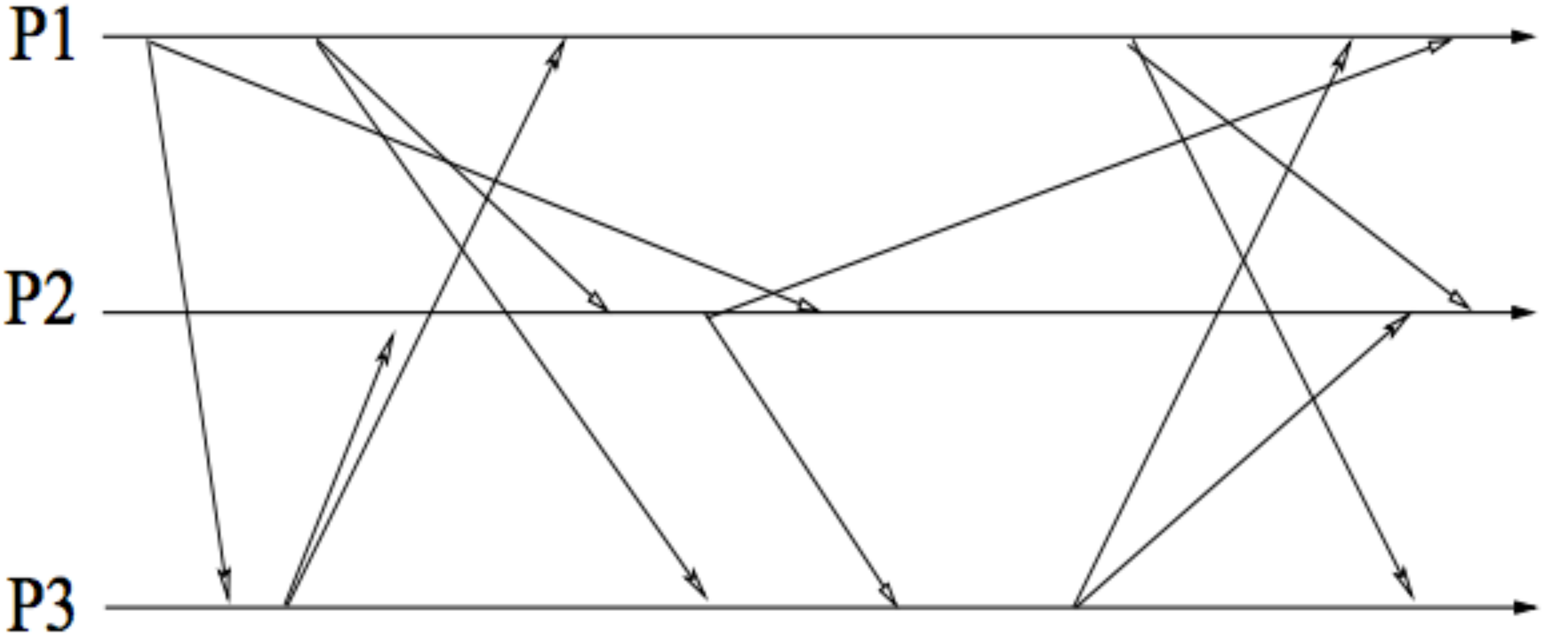
Send, [3, 0, 1]

Buffered as [2, 1, 2], Timestamp still [3, 0, 1]

Accept, [3, 1, 1]

Accept, [3, 1, 2] (Previously buffered)

ii. P2:
Buffered as [1, 0, 1], Timestamp still [0, 0, 0]
Buffered as [2, 0, 0], Timestamp still [0, 0, 0]
Send, [0, 1, 0]
Accept, [1, 1, 0]
Accept, [1, 1, 1] (Previously buffered)
Accept, [2, 1, 1] (Previously buffered)
Accept, [2, 1, 2]
Accept, [3, 1, 2]
iii. P3:
Accept, [1, 0, 0]
Send, [1, 0, 1]
Accept, [2, 0, 1]
Accept, [2, 1, 1]
Send, [2, 1, 2]
Accept, [3, 1, 2]



7. Consider messages C1 ~ C3 in Figure 15.11 of the textbook. Suppose that the boxes indicate multicast message receipt events, not delivery events. How would you deliver the messages for each process in order to preserve,

a. Total ordering?

(Multiple variants are possible.) Let's call the first multicast messge from P1 M1, the second message from P1 M2, and the first message from P3 M3. One total ordering could be delivering M1, M2, & M3 in that order at every process.

b. FIFO ordering?

(Multiple variants are possible.) In order to preserve FIFO order, M1 and M2 should be delivered in that order. One possibility is again delivering M1, M2, and M3 at every process.

c. Causal ordering?

(Multiple variants are possible.) M1 should be delivered before M2; M1 should also be delivered before M3. There is no ordering guarantee necessary between M2 and M3. As long as each process follows this, it can be an answer.

d. Total and causal ordering?

e. The key is find one causal ordering and impose it on all processes. One possibility is to deliver M1, M3, and M2 in that order at every process.

8. Problem 15.20 from the textbook.

a. To RTO-multicast (reliable, totally-ordered multicast) a message *m,* a process attaches a totally-ordered, unique identifier to *m* and R-multicasts it. Each process records the set of message it has R-delivered and the set of messages it has RTO-delivered. Thus it knows which messages have not yet been RTO-delivered. From time to time it proposes its set of not-yet-RTO-delivered messages as those that should be delivered next. A sequence of runs of the consensus algorithm takes place, where the *k'th* proposals (k = 1, 2, 3, ...) of all the processes are collected and a unique decision set of messages is the result. When a process receives the *k'th* consensus decision, it takes the intersection of the decision value and its set of not-yet-RTO-delivered messages and delivers them in the order of their identifiers, moving them to the record of messages it has RTO-delivered. In this way, every process delivers messages in the order of the concatenation of the sequence of consensus results. Since the consensus results given to different correct processes are identical, we have a RTO multicast.

9. Explain why the impossibility of consensus proof (proofs of Lemmas 2 and 3 in the FLP paper) would break if the system were synchronous. Specifically, give at least one statement in the proof that may not hold in a synchronous system.

a. Lemma 2 is not violated in a synchronous system. Lemma 3, however, uses the assumption that there maybe arbitrary message delays. While this is true in an asynchronous system, message delays are bounded in a synchronous system.

10. Problem 15.4 from the textbook.

a. Process *A* sends a request *rA* for entry then sends a message *m* to *B.* On receipt of *m, B* sends request *rB* for entry. To satisfy happened-before order, *rA* should be granted before *rB* . However, due to the vagaries of message propagation delay, *rB* arrives at the server before *rA* , and they are serviced in the opposite order.

11. In a group of 10 processes using the Ricart and Agrawala's algorithm for mutual exclusion, all the 10 processes simultaneously (in physical time) generate a request to enter the same critical section. What is the worst-case number of messages that the algorithm might transmit? (Calculate this number after everyone has exited their critical section.) You can assume that each process generates only one request for the critical section.

a. Each of the 10 processes will send a request to the other 9 processes. Total number of entry request messages: 10 × 9 = 90. At first each process will reply to the other processes with higher priority than itself. Total messages in this stage: 9×(9+1)/2 = 45. Once the highest priority process exits the critical section, it will reply to the 9 queued requests – which enables the process with the second highest priority enter the critical section. When it is done, it replies to its 8 queued requests and the cycle goes on. Therefore, the total number of messages in this stage: 9×(9+1)/2 = 45. So, the worst case number of messages: 90 + 45 + 45 = 180

12. You are given a problem called 2-Leader Election that has the following Safety and Liveness requirements:

Safety: For each non-faulty process p, p's elected = a set of two processes, OR = NULL.

Liveness: For all runs of election, the run terminates AND for each non-faulty process p, p's elected is not NULL.

Modify the Bully Algorithm described in lecture to create a solution to the 2-Leader Election problem. You may make the same assumptions as the Bully Algorithm, e.g., synchronous network. Briefly discuss why your algorithm satisfies the above Safety and Liveness, even when there are failures during the algorithm's execution.

b. (Other ways might be possible) One possible solution: Nodes receiving zero or one reply in response to an election message become the coordinators and send out the coordinator message. Nodes receiving the coordinator message set their 'elected' set only when they receive coordinator message from 2 processes. One special case to handle is that when one of the coordinators detect that the other coordinator is failed. In that case it can instruct a non-faulty process with a lower id to initiate a new election.