Problem 1

(1) Yes

If a non-faulty process sends m before m', then every non-faulty processes delivers m then m'

(2) No

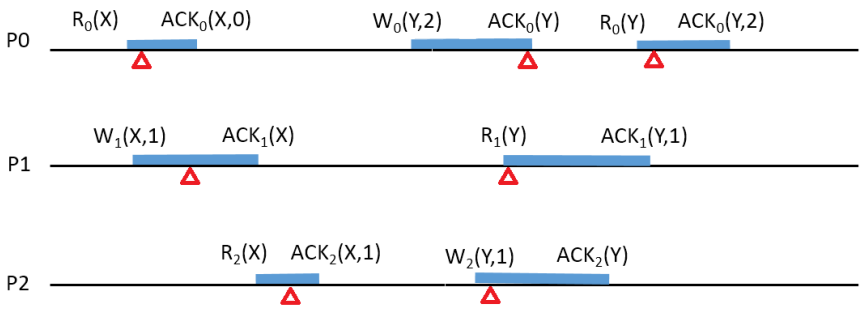
MC3 -> MC4 on P2, but MC4 is delivered before MC3 on P3

MC1 -> MC5 on P3, but MC5 is delivered before MC1 on P1

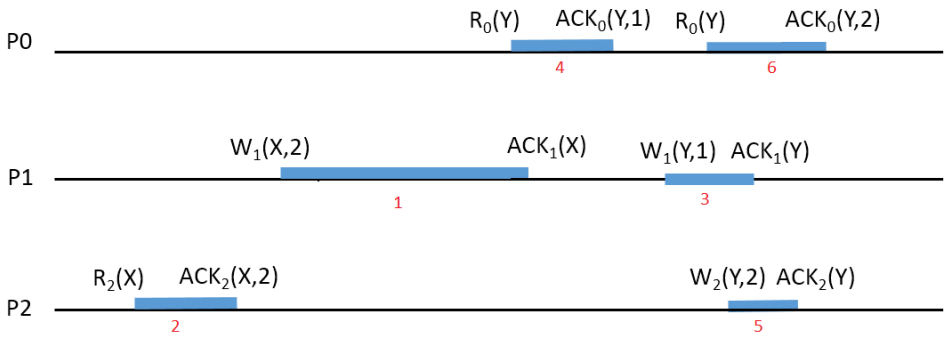
Either of the counter example will suffice.

Problem 2

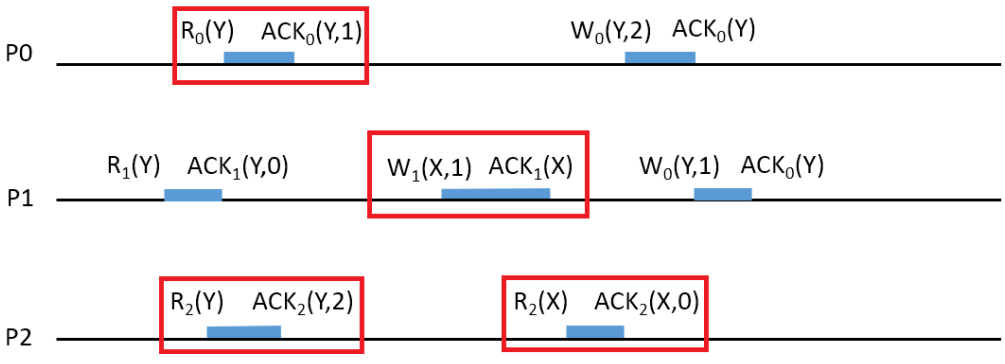
(a)



(b)

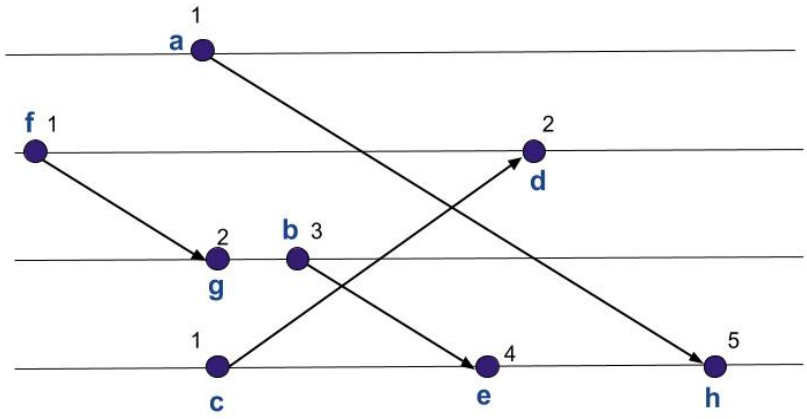


(c)



Problem 3

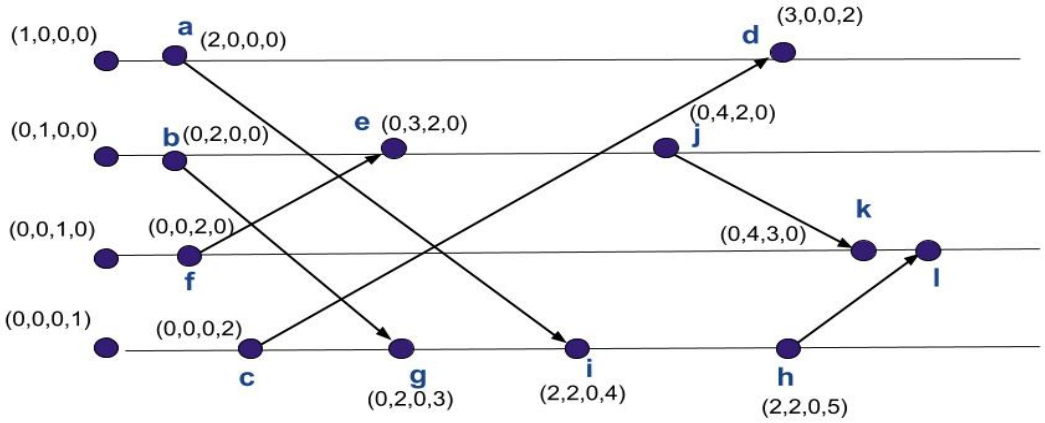
(a)



(b)

{a,b,e,g}

(c)



(d)

Yes,

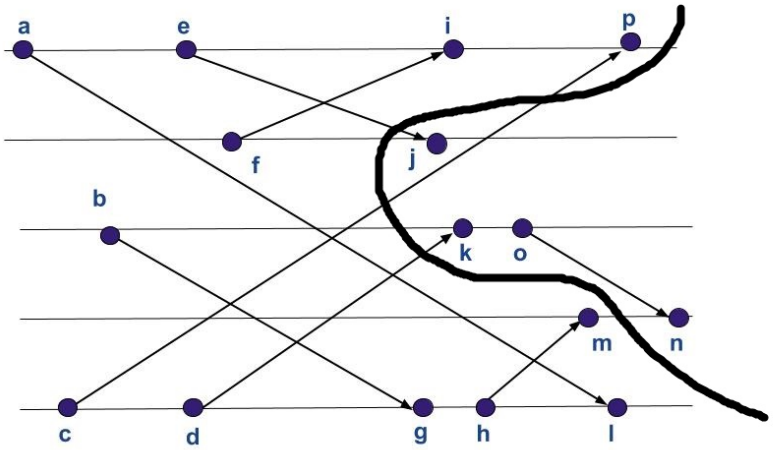
$$V_j \not\leq V_h \; and \; V_j \not\geq V_h$$

Problem 4

(a) Server B. Because this server incurs the minimum expected skew(200/2 - 95)=5 ms

(b) $\frac{RTT}{2} - min = 5ms$

Problem 5



Problem 6

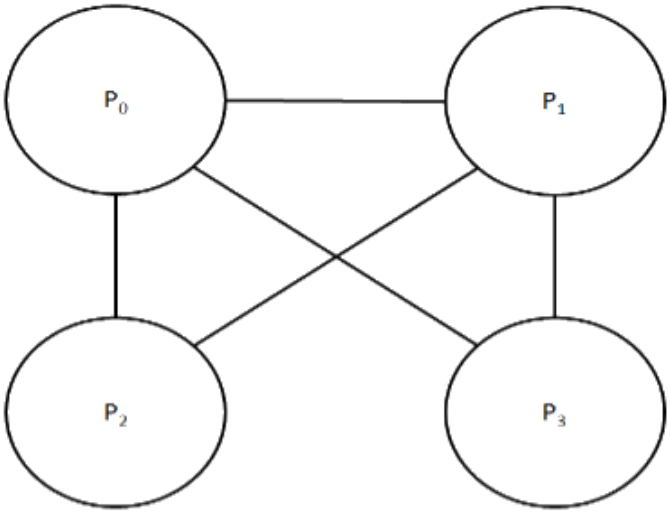
True

True

False

False

Problem 7



v := my input
at each round 1 through round x:
  if I have not yet sent v then send v to all directly connected processes
  wait to receive messages for this round
  v := minimum among all received values and current value of v
  if this is round x then decide on v

(a) Is it possible to achieve consensus with up to 1 crash failure using the above algorithm? Determine the smallest number of rounds x in the above algorithm, or explain why it is not possible.

Yes, 3 rounds. 2 rounds guaranteed failure-free => everyone sees same set of values (because longest path is length 2). Counterexample for just 2 rounds: in first round, P0 sends value to P2 then crashes. Then P3 cannot hear that value in time.

(b) Is it possible to achieve consensus with up to 2 crash failures using the above algorithm? Determine the smallest number of rounds x in the above algorithm, or explain why it is not possible.

No. Impossible to communicate => consensus is impossible. If P0 and P1 crash immediately, then P2 and P3 cannot communicate.

In the counter-example in part (a) above, consider the case when P_0 has input 0, and rest of the processes have input 1.

Problem 8

(a) Yes. There exists a valid permutation that also preserves per-process order:
$Write_0(X,5)$ $Write_1(Y,4)$ $Write_1(X, 2)$ $Read_1(X,2)$ $Read_0(X,2)$ $Read_0(Y,4)$

(b) Yes.
Linearizability is achievable if a permutation satisfies requirements for sequential consistency and also the real-time order. Since there is no constraint specified on real-time at which the operations are invoked, it is possible to construct an execution in which the order of operations in the above permutation is also their real-time order.

Problem 9

No. The magic variable only informs of the first crash in this asynchronous system. But there can be other subsequent crashes too, which are not detected by the magic variable. So the FLP impossibility result still applies. Thus, *exact* consensus is not guaranteed in finite time.

Problem 10

a) Answer: zero. Explanation: Suppose that processes 1 and 2 fail without sending any messages. Then the remaining 3 processes will receive same three values, and compute identical next state.

b) In the worst-case execution below, all processes are non-faulty.

Processes	P1	P2	P3	P4	P5
After round 1	2	2	2	4	4
After round 2	2	2	2	10/3	10/3
After round 3	2	2	2	26/9	26/9

P1, P2, P3 always receive value from each other and their state will stay at 2. P4 and P5 get values from P4, P5 and P3, their state decreases after each round.

The difference between the largest value (state) and smallest value is called *range* below.
The range is 5-1 = 4 initially. After 1 round, it is 4-2 = 2. (It is not possible to achieve a larger range than 2.)

After round 1, the range is 1/2 times the initial range (the initial range is 4, and after 1 round, it is 2).

After round 2, the range is 4/3 = 2/3 times the range after 1 round. (It is possible to show that the range can be no larger than 2/3 times the value after the previous round).

After 3 rounds, the range is 8/9 = 2/3 times the range after 2 rounds.

For r>1, after r rounds, the range is 2/3 times the range after r-1 rounds.

Solve for $2\left(\frac{2}{3}\right)^{r-1} < 0.15$

After r = 8 number of rounds, the difference is lower than 0.15.