Homework 1 (Time, Synchronization and Global State) - 100 Points

CS425 Distributed Systems, Fall 2009, Instructor: Klara Nahrstedt

Out: Thursday, September 3, **Due Date**: Thursday, September 17

Instructions: (1) All problem numbers below refer to the 4th edition of the textbook by Colouris, Dollimore and Kindberg. (2) Please, hand in hardcopy solutions that are typed (you may use your favorite word processor). We will not accept handwritten solutions. Figures and equations (if any) may be drawn by hand. (3) Please, start each problem on a fresh sheet and type your name at the top of each sheet. (4) Homework will be due at the **beginning of class** on the day of the deadline.

Relevant Reading for this Homework: Sections 11.1-11.5

1. (**10 Points**) At 10:27:540 (hr, min, 1/100 sec.), server **B** requests time from the time-server **A**. At 10:27:610, server **B** receives a reply from timeserver **A** with the timestamp of 10:27:375.

    a. (**5 Points**) Find out the drift of **B**'s clock with respect to the time-server **A**'s clock (assume there is no processing time at the time-server for time service).

Answer 1a: RTT: Reply – Request = 610-540 = 70 1/100sec.
Adjusted local time: Server + RTT/2 = 375 + 35 = 410 1/100sec.
Drift: Adjusted local time – local time = 410 – 610 = -200 1/100sec. = **-2 sec**

    b. (**2 Points**) Is **B'**s clock going too fast or too slow? If the answer is yes, by how much is the clock going too fast or too slow?

Answer 1b: B's block is running 2 seconds **too fast**.

    c. (**3 Points**) How should **B** adjust its clock?

Answer 1c: To adjust **B**'s clock we cannot just set the time back 2 seconds or we lose our monotonicity condition (The condition that the clock always moves only forward), which could result in events appearing as if they occurred in the future and a variety of issues in software relating to this problem. For **B** to adjust its clock safely, it should decrease the rate at which updates are given to its clock (at least in software) thus "slowing down" the time until it is caught up with the server's time.

2. (**5 Points**) In the symmetric mode of synchronization in NTP, suppose you are given that server **A** and server **B** are connected by a **symmetric channel**, i.e., the channel shows the same (but unknown) message delay both ways, i.e., the **A** to **B** delays is the same as **B** to **A**

delay. Show using the equations for analyzing the NTP protocol, that under this situation, one can estimate the clock skew accurately (i.e., with an error of 0).

Answer: If the channel is symmetric, then t = t', and
$o = o_i + (t' - t)/2 = o_i$
That is, the actual offset (clock skew) is perfectly estimated (i.e., the error (t'-t)/2 = 0).

3. (**20 Points**) Consider Figure 1 that shows four processes ***(P1, P2, P3, P4)*** with events ***a, b, c,*** … and messages communicating between them. Assume that initial logical clock values are all initialized to ***0***.

   a. (5 Points) List the **Lamport timestamps** for each event shown in Figure 1. Assume that each process maintains a logical clock as a single integer value as a Lamport clock. Provide timestamps for each labeled event.

   b. (10 Points) List the **Vector Clock timestamps** for each event shown in Figure 1. Provide timestamps for each labeled event.

   c. (5 Points) Is there the potential for a causal violation? Explain why.

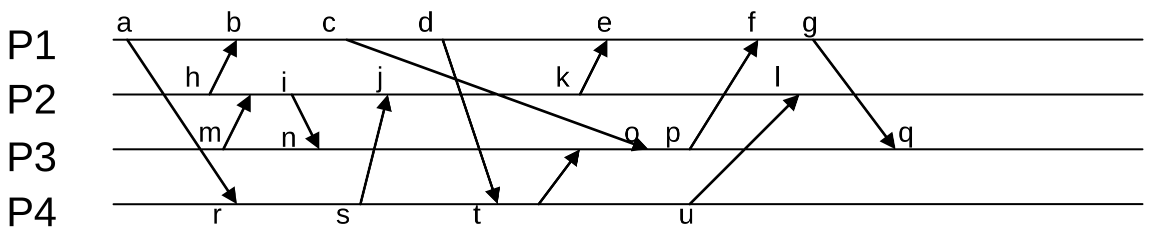
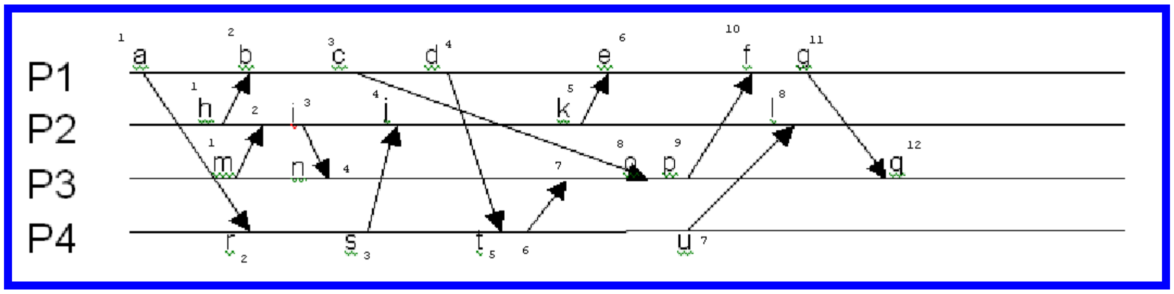
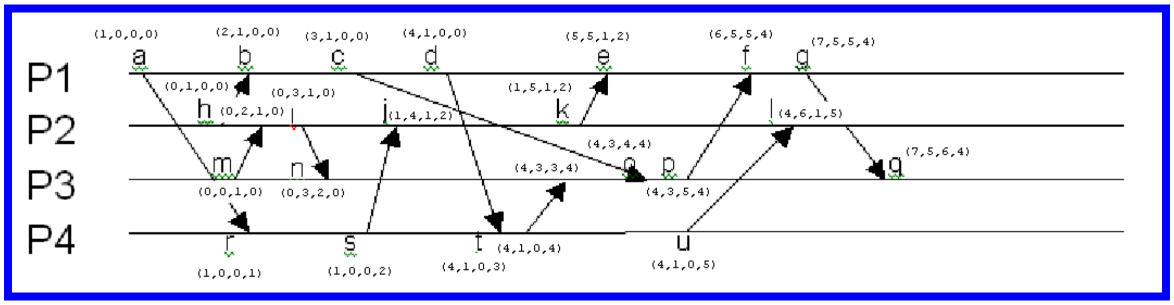


Figure 1: Four *Processes P1, P2, P3, P4 run events a,b,c,d,…. to send and receive messages*

Answer 3a&3b: Answers for 3a and 3b are in **Figure 3**.

Answer 3c: Yes, there is a potential causality violation. Specifically from point C to point O, the message passed holds (3,1,0,0) as the vector timestamp but the local vector timestamp on **message arrival** at point O is (4,3,3,4). Because the vector timestamp passed in the message is less than the local vector timestamp on arrival there is potential for a causal violation.





Event	Lamport Time Stamp	Vector Time Stamp
a	1	(1,0,0,0)
b	2	(2,1,0,0)
c	3	(3,1,0,0)
d	4	(4,1,0,0)
e	6	(5,5,1,2)
f	10	(6,5,5,4)
g	11	(7,5,5,4)
h	1	(0,1,0,0)
i	3	(0,3,1,0)
j	4	(1,4,1,2)
k	5	(1,5,1,2)
l	8	(4,6,1,5)
m	1	(0,0,1,0)
n	4	(0,3,2,0)
o	8	(4,3,4,4)
p	9	(4,3,5,4)
q	12	(7,5,6,4)
r	2	(1,0,0,1)
s	3	(1,0,0,2)
t	5	(4,1,0,3)
u	7	(4,1,0,5)

Figure 3: *Answer for Problem 3a and 3b***.**

4. (**35 Points**) Consider **Figure 2** showing three processes ***P1, P2, P3***.

   a. (5 Points) Is the run <e_1^1, e_2^2, e_2^3, e_3^2 e_2^4, e_3^3> a **linearization** of events? Explain why or why not.

Answer 4a: (One reason is sufficient) No, the run is **not a linearization** of events. One of the reason is that the event e_2^4 (effect) is before e_3^3 event (cause) which violated the happen-before relation. Linearization is a sequence of events that is consistent with happen-before relation. $e_3^3 \rightarrow e_2^4$ is in a happen-before relation, and so if it were linearization run, e_2^4 would have been before e_3^3.

Another reason might be that the linearization does not have a complete set of all events in the global history of Figure 2.

   b. (5 Points) Is the cut, shown by curve **X,** a **consistent cut**? Why?

Answer 4b: No. For a cut to be consistent, if e_i is in the cut and e_j happens before e_i, then e_j must be in the cut. In this example, e_2^4 (receive event) is in the cut, but the corresponding send event (e_3^3) that happens before it is not in the cut.

   c. (10 Points) Determine **two other consistent cuts** in Figure 2 and specify their **frontiers**.

Answer 4c: See Figure 4 for two additional consistent cuts. Cuts **Y** and **Z** are consistent cuts. The frontier for Cut **Y** is *<e_1^1, e_2^2, e_3^1>*, The frontier for Cut **Z** is *<e_1^2, e_2^3, e_3^2>* .

   d. (15 Points) In the **Figure 2,** determine all the events that **happen before** event ***e_2^4*** (as per Lamport's Happened-Before relation). Also determine the events that are concurrent with event ***e_2^4***.

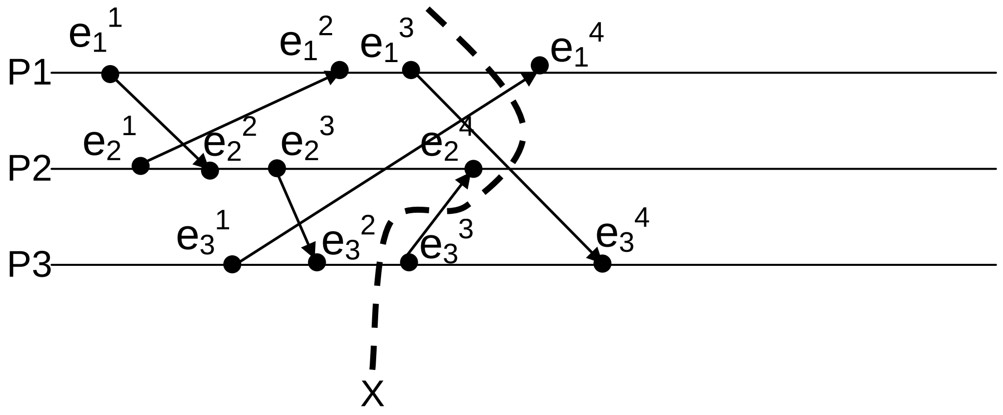


Figure 2: *Three Processes run events to send and receive messages with a Cut X through the processes.*

Answer 4d: Events that happened before event e_2^4: e_1^1, e_2^1, e_2^2, e_2^3, e_3^1, e_3^2, e_3^3.
Events that are concurrent with event e_2^4: e_1^2, e_1^3, e_1^4, e_3^4

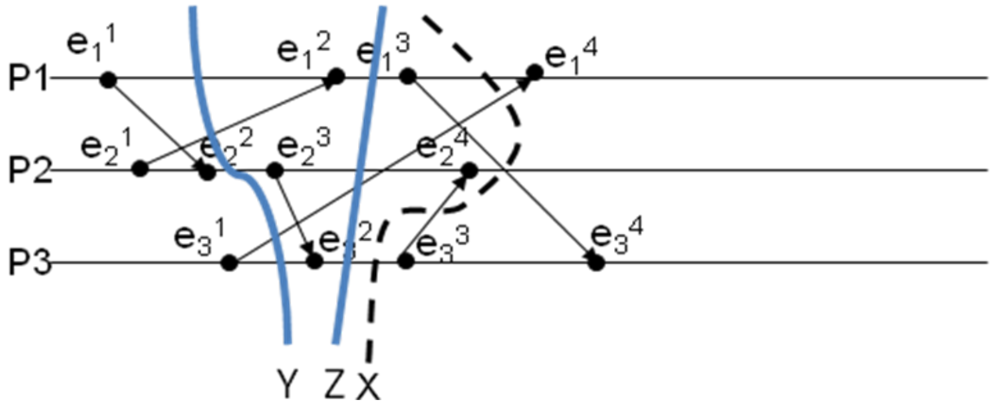


Figure 4: *Answer to Problem 4c – two additional consistent cuts Y and Z.*

5. (**10 Points**) ***a, b,*** and ***c*** are events and no two events belong to the same process. Prove or disprove (give counter-example) the following:

   a. (5 Points) ***a*** is concurrent with ***b* and *b*** is before ***c*** implies that ***a*** is before ***c***.

Answer 5a: No, consider counter example in **Figure 5a**: *a* and *b* are concurrent, *b* $\rightarrow$ *c*; however *a* is also concurrent with *c.*

   b. (5 Points) ***a*** is concurrent with ***b* and *b*** is concurrent with ***c*** implies that ***a*** is concurrent with ***c***.

Answer 5b: No, consider counter example in **Figure 5b**: *b* is concurrent with both *a* and *c*, however, *a* $\rightarrow$ *c*.

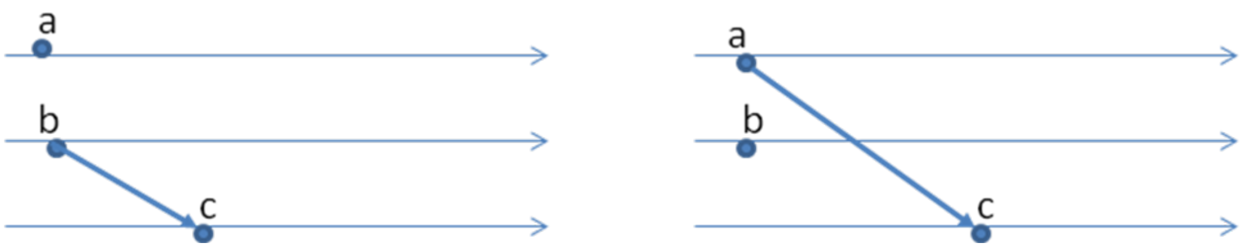


Figure 5 : *Counter Example for 5a (left) , and Counter Example for 5b (right)*

6. **(10 Points)** Give an example where the Lamport's clock algorithm comes short (i.e., the Lamport's algorithm cannot clearly conclude that event ***e*** happens before ***e'***, even those ***L(e) < L(e'),*** where ***L(e)*** is the Lamport's timestamp of the event e), and the vector clock algorithm concludes clearly that event ***e*** happened before ***e'*** or NOT.

Answer: Consider the following example of 2 processes (Figure 6). The Lamport timestamp 2 in P1 is clearly larger than Lamport timestamp 1 in P2, but the corresponding events are not in happen-before relation. On the other hand with vector timestamps (2,0) and (0,1), corresponding to those two events, we can clearly conclude that these two events are not in happen-before relation (since VT1[P1] > VT2[P1] and VT1[P2] < VT2[P2]).

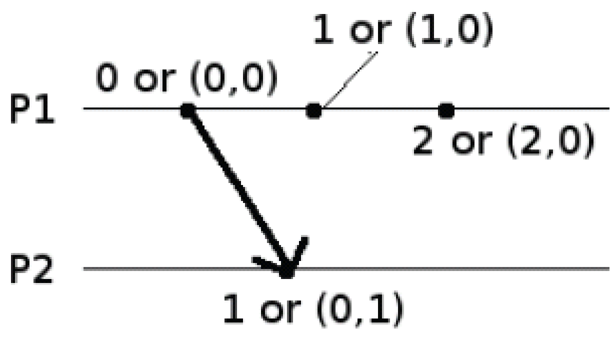


Figure 6: *Answer to Problem 6.*

7. **(10 Points)** If the FIFO channel assumption in the Chandy-Lamport algorithm is violated, then which step of the proof for the Chandy-Lamport algorithm given a consistent cut, breaks down?

Answer: The step of the proof that falls apart is: "If ej occurred before pj recorded its state, then ei must have occurred before pi recorded its state". Consider the **Figure 7** with P1 sending generic message **M** and snapshot marker message **SS**. If the FIFO channel assumption is violated, then it is possible that a message receive event (e3) is in the saved state of P2, even though the corresponding message send event (e2) is not in the saved state of P1. The resulting cut will contain the event e3 but exclude an event that happens-before it, thus is not consistent.

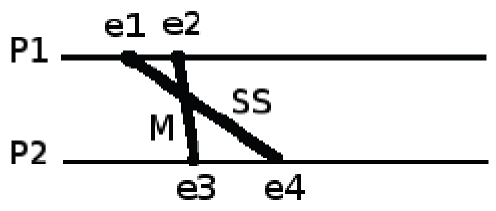


Figure 7: *Answer to Problem 7*.