SOLUTIONS ACTIVITIES UNIT 9 --> Activities 1 to 9

Activity 1

a) The client, to which value sets its local clock?

$\mathbf{C_s}$= 11:01:04,830 $\mathbf{T_0}$ = 11:01:04,750 $\mathbf{T_1}$ = 11:01:04,880

$\mathbf{C} = C_s + (T_1 - T_0)/2$ = 11:01:04,830 + (11:01:04,880 - 11:01:04,750)/2 = 11:01:04,895

b) Will the client's local clock have to suspend its progress for a while? Why?

No, the client's local clock won't have to suspend its progress for a while. As $C > C_c = T_1$, the update simply consists in settting the clock of the client to C, that is $C_c = C$.

c) If in addition we know that the request message needed to be transmitted 50ms and its response 80ms, could we claim that the synchronization made by Cristian algorithm was accurate? Why?

No, we would not be able to claim that the synchronization was accurate. Cristian's algorithm assumes that the sending time of both messages (request and answer) is practically the same, and as in this case that condition is not fulfilled, the adjustement wouldn't be right.

In fact, Cs would be 4'830 + 0'080 = 4'910, i.e. 15ms more than what the client calculated. So the client has been adjusted to 15ms less than required. However, if we keep on with the synchronization, at the end this will be more or less accurate.

<u>Activity 2.</u> In a distributed system with 4 nodes, we employ Berkeley algorithm for synchronizing their clocks. One of the nodes (node A) acts as the server of the algorithm, whereas the others (nodes B, C and D) act as clients. Let's suppose that all nodes have clocks that indicate the number of ticks occurred from the same temporary base; and that at one point they have the following value for their clocks: C_A=10000, C_B=10005, C_C=10005 and C_D=10010 (being C_n the clock of node n). At that point the server (node A) initiates the Berkeley algorithm.

If we assume that it takes *10 ticks* each time a message is sent, from its sending to its reception, which is the final value of the clocks of each node?

$\mathbf{T_0}$	10.000
$\mathbf{D_B}$	10.015 – 10.000 = 15
$\mathbf{D_C}$	10.015 – 10.000 = 15
$\mathbf{D_D}$	10.020 – 10.000 = 20
$\mathbf{T1_i}$	10.020
$\mathbf{D_B'}$	15 – (10.020 – 10.000)/2 = 5
$\mathbf{D_C'}$	15 – (10.020 – 10.000)/2 = 5
$\mathbf{D_D'}$	20 – (10.020 – 10.000)/2 = 10
D	(5 + 5 + 10 + 0)/4 = 5
$\mathbf{A_B}$	5 – 5 = 0
$\mathbf{A_C}$	5 – 5 = 0
$\mathbf{A_D}$	5 – 10 = -5
$\mathbf{C_A}$	C_{A0} + D + (Msg. delay * No. Messages) = 10.000 + 5 + (10*3) = **10.035**
$\mathbf{C_B}$	C_{B0} + A_B + (Msg. delay * No. Messages) = 10.035 + 0 + (10*3) = **10.035**
$\mathbf{C_C}$	C_{C0} + A_C + (Msg. delay * No. Messages) = 10.035 + 0 + (10*3) = **10.035**
$\mathbf{C_D}$	C_{D0} + A_D + (Msg. delay * No. Messages) = 10.040 + (-5) + (10*3) = **10.035**

Correction: In fact, C_D receives that it needs to update -5, and **D will stop its clock 5 ticks**, so it will **not decrement it.** At 10.040 all clocks will be synchronized (and then D clock will be able to keep on working).

<u>Activity 3.</u> Discuss which effect will have on Berkeley's algorithm its usage on a local network with 10 computers and where it occurs the following cases:

a) ***One of the computers has a faulty clock that is systematically delayed one minute per each hour passed. The rest of nodes have "normal" clocks that do not suffer any relevant forward moves or delays during their behavior.***

When the algorithm detects the unusually large difference on that client's clock, the provided value from that client will not be taken into account when performing the calculations.

b) ***None of the computers have a correct clock (i.e. accurate clock) but their defects that they might have are compensated between them.***

As the objective of Berkeley's algorithm is not to synchronize the "real" time for every node, but to reach an agreement between nodes, it does not matter if the time of the nodes is not the real one, as long as every client has the same time set.

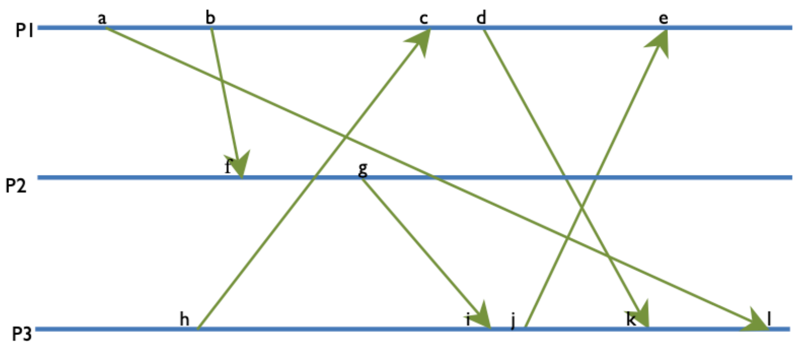
ACTIVITY 4

OBJECTIVES: To describe the synchronization algorithms for physical clocks.

Given the following sentences, update them as needed to make them all **TRUE**.

1. In the Cristian algorithm, if a client C asks the server its time at instant 20.000 (according to C clock), receives the server answer at instant 20.010 (according to C clock) and calculates that the new value for its clock must be 20.024, then you can deduce that the answer of the server has been ~~14~~ **20.019**. **False, the calculated delay is 5ms which is added to get the clock of 20.024, so substracting from it we get server time will be 20.019**
2. In the Berkeley algorithm, the node that acts as coordinator ~~never~~ **may** need to adjust its clock. Only the clocks of the other nodes that participate in the algorithm are adjusted. **False**
3. The Berkeley algorithm assumes that sending a message from node A to node B takes the ~~double~~ **same** time than the answer from B to A. **False, it uses the whole period of sending and receiving and then divides in two assuming they both last the same time.**
4. In the Cristian algorithm, the ~~average between the client clock and~~ the server clock is ~~calculated to~~ set the client time. **False, only server clock is taken into account**
5. In the Berkeley algorithm, each client can synchronize its clock at any time, ~~independently of when~~ **then all** the other nodes synchronize their clocks **too**. **False** In the original sentence, if we change Berkeley to Cristian, then we will make it TRUE
6. Suppose that six nodes use the Berkeley algorithm to synchronize their clocks. The node N_1 starts the algorithm when the values of the clocks of the six nodes are respectively N_1 = 2002, N_2 = 2008, N_3 = 2005, N_4 = 2010, N_5 = 2000 and N_6 = 2005 and it receives the responses from the other nodes to the same time. ~~Given these assumptions, node N_1 determines that it must advance its own clock in 30 units of time~~. **False, it cannot determine an advancement of 30 units as the maximum deviation from clocks is of 10 units** In fact, it will calculate the mean: (0+6+3+8+(-2)+3) / 6 = 3. So N_1 must advance 3 units of time
7. In the Berkeley algorithm, if we assume a total of N nodes, it is required to send a total 3*(N-1) messages. TRUE: If N includes the coordinator, this coordinator sends initially its clock to all other nodes (N-1), receives from them their differences (N-1 messages), and then sends to them what they have to adjust (N-1 messages). That makes a total of 3 * (N-1) messages.
8. The Cristian's algorithm requires ~~knowing a priori the average delay of the messages exchanged between the different nodes~~. **False, the delay is calculated approximately with the sending and receiving time.**
9. In the Cristian's algorithm, if a client C asks the server for its time at the instant 10.000 and the server answers 10.006 and this response arrives to C at instant 10.008, then node C must stop its clock for 2 units of time. FALSE: node C must adjust its clock to C = Cs + (T1 - T0)/2 = 10006 + (10008 - 10000)/2 = 10006 + 4 = 10.010. So it must **increment** in 2 units of time.

Activity 5. Given the next execution in a distributed system formed by 3 nodes...



a) ***Indicate for each of these events of sending and receiving that are shown in the figure, which is the value that Lamport's algorithm assigns for these events (i.e. the value of the Lamport's logical clock for these events). Assume that the execution begins at the left of the figure: all nodes start from time 0.***

a:	1	**b:**	2	**c:**	3
d:	4	**e:**	7	**f:**	3
g:	4	**h:**	1	**i:**	5
j:	6	**k:**	7	**l:**	8

b) ***Indicate which events are concurrent with event "i".***

The events that are concurrent with another one cannot be known with *Lamport's* algorithm. However, taking in account the information from the following exercise (activity 6): $c||i$ and $d||i$.

c) ***Indicate which events are concurrent with event "e".***

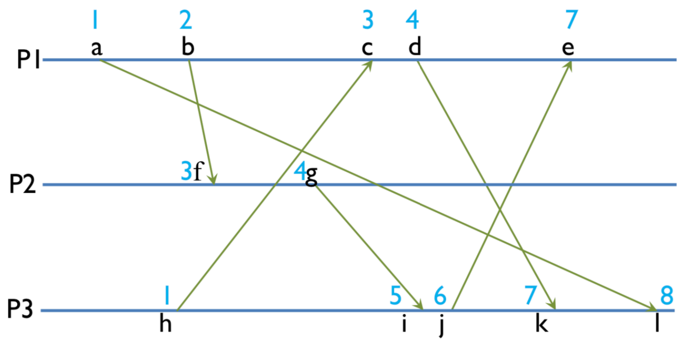
The events that are concurrent with another one cannot be known with Lamport's algorithm. However, taking in account the information from the following exercise (activity 6): $e||k$ and $e||l$.

d) ***Can you give a total order of the events? If so, which one?***

Yes, a total order can be stablished if the identification of the node is added as a suffix of the value that the algorithm assigns to each event. In this case, it would remain as follows:

a:	1.1	**b:**	2.1	**c:**	3.1
d:	4.1	**e:**	7.1	**f:**	3.2
g:	4.2	**h:**	1.3	**i:**	5.3
j:	6.3	**k:**	7.3	**l:**	8.3

a	1.1
h	1.3
b	2.1
c	3.1
f	3.2
d	4.1
g	4.2
i	5.3
j	6.3
e	7.1
k	7.3
l	8.3



Activity 6. Given the execution shown in previous activity:

a) ***Indicate which vector clock will be assigned each of the events shown in that execution, assuming that initially each process has a local clock with value [0,0,0].***

a:	$[1,0,0]$	**b:**	$[2,0,0]$	**c:**	$[3,0,1]$
d:	$[4,0,1]$	**e:**	$[5,2,3]$	**f:**	$[2,1,0]$
g:	$[2,2,0]$	**h:**	$[0,0,1]$	**i:**	$[2,2,2]$
j:	$[2,2,3]$	**k:**	$[4,2,4]$	**l:**	$[4,2,5]$

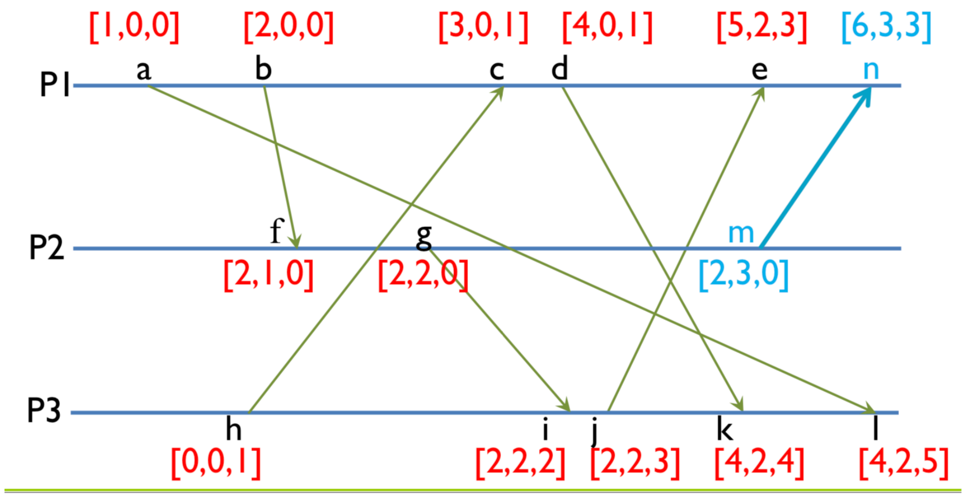
b) ***Using the vector clocks of the previous question, give a complete list of pair of events that are concurrent between them. For example, an element of that list will be the pair of events "a" and "h" and their respective vector clocks are [1,0,0] and [0,0,1].***

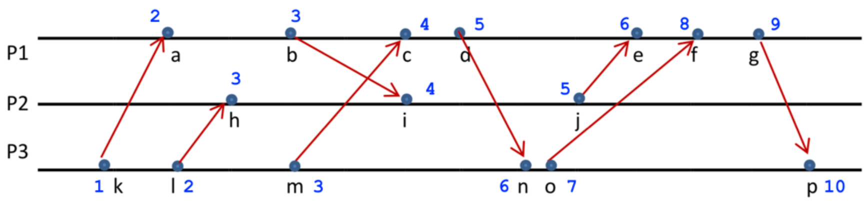
$a||h \quad b||h \quad c||f \quad c||g \quad c||i \quad c||j \quad d||f$

$d||g \quad d||i \quad \mathrm{d}||j \quad e||k \quad e||l \quad f||h \quad g||h$

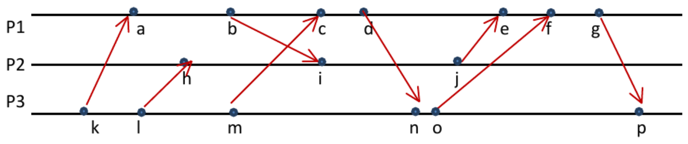
c) ***Imagine that after the trace shown in that previous execution, process P2 sends a message to process P1 and there is no other event in this system... which vector clocks will be assigned to the sending event and to the receiving event of that message?***

Event `m` (in P2) would have the vector clock $[2,3,0]$, and event `n` (in P1), would have the vector clock $[6,3,3]$.





Activity 7



a) Indicate the Lamport's Logical Clock value for each of these event

a:	2	**b:**	3	**c:**	4	**d:**	5
e:	6	**f:**	8	**g:**	9	**h:**	3
i:	4	**j:**	5	**k:**	1	**l:**	2
m:	3	**n:**	6	**o:**	7	**p:**	10

b) Indicate the relationships "happens -before" and what events are concurrent with each other

This information cannot be deduced from the information obtained using Lamport's algorithm, so in this case we have to use the information provided by the vector clocks' algorithm in question d):

Happens-before relation:

a →	b,c,d,e,f,g,i,j,n,o,p	**b** →	c,d,e,f,g,i,j,n,o,p	**c** →	d,e,f,g,n,o,p
d →	e,f,g,n,o,p	**e** →	f,g,p	**f** →	g,p
g →	p	**h** →	e,f,g,i,j,p	**i** →	e,f,g,j,p
j →	e,f,g,p	**k** →	a,b,c,d,e,f,g,h,i,j,l,m,n,o,p	**l** →	c,d,e,f,g,h,i,j,m,n,o,p
m →	c,d,e,f,g,n,o,p	**n** →	f,g,o,p	**o** →	f,g,p
p →	-				

Concurrent events:

a||h a||l a||m b||h b||l b||m c||h c||i c||j d||h d||i d||j e||n e||o h||m h||n h||o i||m i||n i||o j||m j||n j||o

c) Looking only at the values of Lamport's logical clocks, can we determine which events have happened before others? And what events are concurrent?

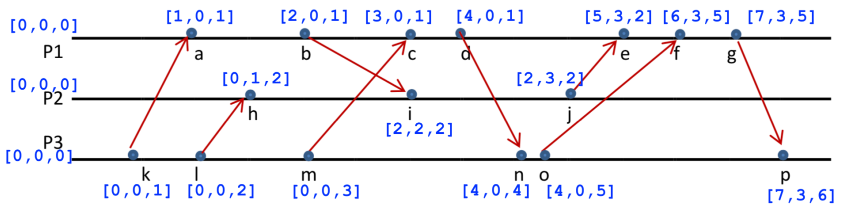
No, the values of Lamport's logical clocks are not enough to determine which events have happened before others or which are concurrent, because if C(a) < C(b) we cannot decide in which of the two cases we are in.

d) Indicate the Vector Clock value for each of these events

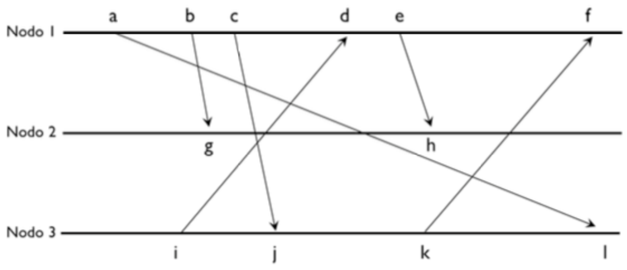
a:	[1,0,1]	**b:**	[2,0,1]	**c:**	[3,0,3]	**d:**	[4,0,3]
e:	[5,3,3]	**f:**	[6,3,5]	**g:**	[7,3,5]	**h:**	[0,1,2]
i:	[2,2,2]	**j:**	[2,3,2]	**k:**	[0,0,1]	**l:**	[0,0,2]
m:	[0,0,3]	**n:**	[4,0,4]	**o:**	[4,0,5]	**p:**	[7,3,6]

e) Looking only at the values of these clocks, can we determine which events have happened before others? And what events are concurrent?

Yes, we can determine which events happened before others and which are concurrent. If every component of Vector A is lower or equal than the respective component of Vector B and at least one component is stricly lower, we can conclude that event A happens before B. On the other hand if Vector A is not lower than Vector B according to that definition and Vector B is also not lower than Vector A, we can say that events A and B are concurrent.



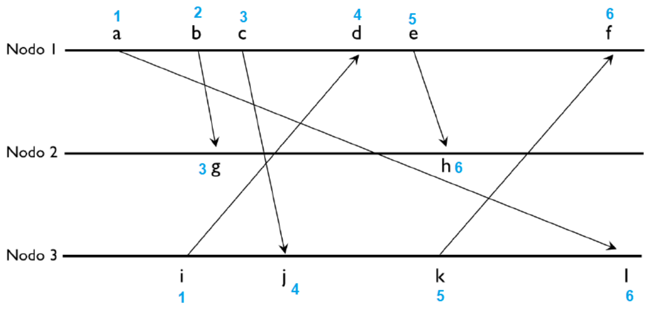
Activity 8. Given the following schedule:



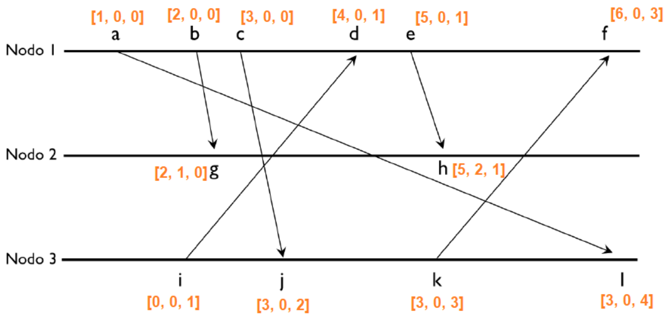
Indicate whether the following sentences are true (T) or false (F) and justify your answer.
The justification for every section is provided below as the complete procedure of the algorithms.

1. It holds that **c → l** and that **e || l** ***Justification:*** *Justified using vector clocks algorithm,* $[3,0,0] < [3,0,4]$ *which means that* **c → l**; $[5,0,1]$ *not* $< [3,0,4]$ *and* $[3,0,4]$ *not* $< [5,0,1]$ *which means that* **e || l**	**T**
2. The Lamport logical clock in **l** is equal to 6. ***Justification:*** *Justified using Lamport's algorithm.*	**T**
3. The vector clock in d is equal to $V(d) = [4,0,1]$ and the vector clock in j is equal to $V(j) = [3,1,2]$ ***Justification:*** *Justified using vector clocks algorithm,* $V(j) \neq [3,1,2]$	**F**

Lamport's algorithm procedure



Vector clocks algorithm procedure



Activity 9. Indicate whether the following sentences are true (T) or false (F). In case it is false, update the sentence to make it true.

1. We say that event "a" precedes event "b" (a -> b) if all nodes agree that they first see event "a" and then see event "b". **True.**

2. The *Lamport's* logical clocks guarantee that within the same node the values L(x) are always strictly increasing. **True.**

3. Two events "a" and "b" of different nodes ~~cannot~~ **can** have logical values associated with L(a) and L(b) such that $L(a) = L(b)$ **when using Lamport's algorithm**. **False.**

4. The vector clock associates a vector value to each event (e.g. for event "x" we speak of V(x)), so that if for any two events "a" and "b", if it holds that V(a)< V(b), then a -> b. **True.**

5. In vector clocks, at each node/**event** we need a vector of integer values, with one element for each possible ~~event~~ **node**. **False.**

6. In vector clocks, for any pair of distinct vectors V(a) V(b), **if** it holds that V(a) **not** < V(b) ~~or~~ **and** V(b) **not** < V(a), **it means that the events are concurrent**. **False.**