SOLUCIONES ACTIVIDADES UNIDAD 9

1.- a) *¿A qué valor establecerá su reloj local?*

Según la fórmula del algoritmo de Cristian, el cliente debería sumar al valor del reloj retornado por el servidor la mitad de la diferencia entre el instante de recepción de la respuesta (T1) y el instante de envío de su petición (T0).

En este caso la diferencia entre T1 y T0 vale 13 centésimas de segundo; es decir, 130 ms.

Por tanto, habrá que sumar 130/2 = 65 ms al valor retornado por el servidor. Con ello, el valor al que debería fijarse el reloj del cliente en el instante de recepción de la respuesta sería 11:01:04,830 + 0,065 --> 11:01:04,895.

b) *Se tendrá que suspender el avance del reloj local del cliente durante algún tiempo? ¿Por qué?*

Esa acción no resulta necesaria en este ejemplo, pues el valor al que debemos establecer el reloj (11:01:04,895) es superior al que marcaba el reloj en el instante en que se recibió la respuesta (11:01:04,880). Por tanto, basta con avanzar el reloj local en el receptor 15 ms. Sólo en caso de que el reloj del cliente hubiera tenido un valor superior al que recomendase el algoritmo de Cristian se tendría que haber "parado" durante cierto tiempo el reloj del cliente o bien "ralentizarlo" para que avanzara más despacio hasta que se compensara esa diferencia de tiempo.

*c) Si además de lo dicho en el apartado anterior se supiera que el mensaje de petición necesitó **50 ms** en ser transmitido y el de respuesta **80 ms**, ¿se podría afirmar que la sincronización efectuada por este algoritmo fue precisa? ¿Por qué?*

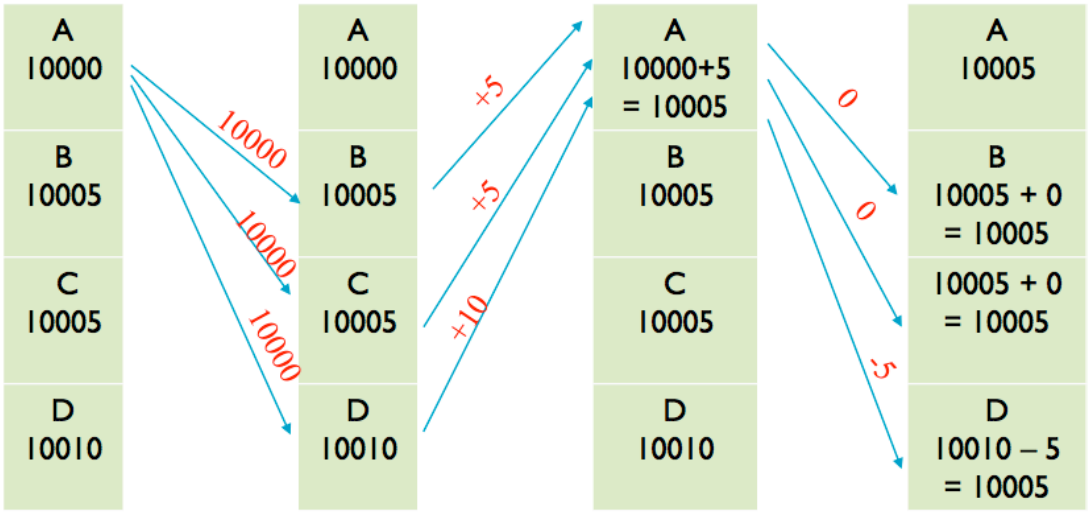
La sincronización efectuada en el apartado "a" asumía que ambos mensajes necesitaron 65 ms para transmitirse, respectivamente. Sin embargo, en el mensaje de petición se tardó algo menos (15 ms menos), mientras que en el de respuesta se tardó algo más (unos 15 ms más).

El ajuste que realizó el algoritmo se hizo asumiendo que el servidor leyó su reloj local cuando en el reloj del cliente la "hora" era 11:01:04,815. Sin embargo, según se nos dice ahora en los tiempos de propagación de los mensajes de petición y respuesta, el instante en el reloj del cliente cuando el servidor leyó su reloj local fue 11:01:04,800 (es decir, 11:01:04,750 + 50ms) . Con ello, el cliente debería haber avanzado su reloj sólo:

11:01:04,830 – 11:01:04,800 = 30ms

Es decir, tendría que haber avanzado 30ms, en vez de los 15ms calculados en el primer apartado. Sin embargo, aunque no parezca demasiado precisa esta sincronización, conforme se va repitiendo el proceso de sincronización estos pequeños desajustes se van subsanando. El algoritmo de Cristian basa su precisión en un comportamiento estable de los canales de comunicación proporcionando retardos predecibles y relativamente estables.

2.- *a) Explique gráficamente cuáles son los diferentes pasos del algoritmo y qué mensajes se envían entre los nodos.*

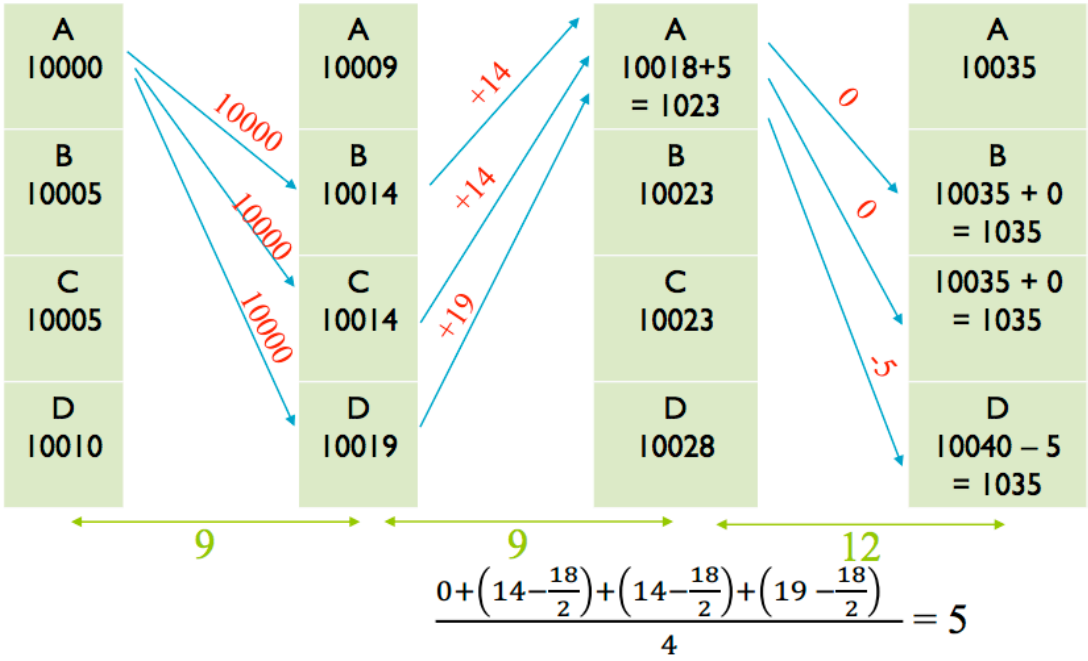


$$\frac{0+5+5+10}{4} = 5$$

En el primer paso del algoritmo, el nodo A (que actúa de servidor) difunde el valor de su reloj (en este caso, C_A=10000). Cada cliente responde con la diferencia entre su reloj local y el del servidor. Así, los nodos B y C responden con "+5", mientras que el nodo D responde con "+10". Tras obtener las respuestas, el nodo A calcula la diferencia media (como se muestra en la figura), obteniendo que el valor medio de las diferencias entre todos los relojes de los nodos es "5". Por tanto, el nodo A deberá actualizar su reloj en 5 unidades (como se muestra en la tercera columna de la figura). Y además el nodo A contesta a todos los clientes con la diferencia que debe aplicar cada uno para actualizar su reloj local. En este caso, como los nodos B y C ya tenían esas "5" unidades de más respecto al reloj de A, entonces no tienen que modificar sus relojes. Por su parte, el reloj del nodo D iba 10 unidades adelantado respecto al reloj de A, por lo que ahora debería "retroceder" 5 unidades.

Nota: aquí en la figura hemos puesto que el reloj del nodo D se decrementa 5 unidades, para ver visualmente que todos los relojes obtendrían el mismo valor tras aplicar el algoritmo. Pero recordemos que los relojes no se pueden retrasar. Así que en realidad, tras comunicarle al nodo D que debe actualizar su reloj en "-5", dicho nodo detendrá su reloj durante 5 unidades de tiempo. De este modo, cuando pasen 5 unidades, los relojes de A, B y C tendrán el valor 10010 (es decir, 10005 de aplicar el algoritmo de Bekerley más 5 unidades de tiempo que han pasado), y el reloj de D también tendrá el valor 10010 (es decir, 10010 que ya tenía y ha estado 5 unidades de tiempo detenido).

b) Si asumimos que han transcurrido 30 ticks durante todos esos pasos, ¿cuál será el valor final del reloj de cada nodo?



Asumiremos que los 30 ticks indicados se han repartido conforme se muestra en la figura, es decir, 9 ticks para el envío y recepción del valor del reloj del nodo A y de las diferencias de los relojes de los otros nodos; y 12 ticks para el cálculo de la media y el envío de la diferencia a aplicar a los relojes de los nodos.

Con ello, en la segunda columna, cuando los nodos reciben el valor del reloj de A, hay que tener en cuenta que habrán pasado 9 ticks, por lo que la diferencia de sus relojes respecto a C_A es superior al caso anterior en 9 unidades.

En la tercera columna, se ha reflejado el cálculo de la media que realiza el nodo A y la actualización de su reloj. Además, vemos también el valor de los relojes de los otros nodos, tras haber pasado 9 ticks.

Respecto al cálculo de la media, el nodo A sabe que desde que envió el valor de su reloj hasta que ha recibido la respuesta por parte de los otros nodos han pasado 9+9= 18 ticks. Además, asume que ambos mensajes (envío de su reloj y envío de la respuesta) han tardado lo mismo, por lo que los otros nodos calcularon la diferencia de sus relojes respecto a CA en 18/2. Por ello, a la hora de calcular el valor medio de la diferencia de los relojes debe restar el tiempo que transcurrió entre que él envió su mensaje y los nodos lo recibieron (y en ese momento calcularon su diferencia). Como se observa, el valor medio obtenido es el mismo que en el caso anterior, es decir, 5 unidades de tiempo.

Finalmente, en la cuarta columna se muestra que, aunque pasen 12 ticks hasta que reciben los clientes la diferencia a aplicar, el algoritmo ha calculado bien el número de ticks a aplicar y todos pueden sincronizar sin problemas sus relojes. Por otro lado, al igual que se ha explicado en el apartado anterior, para el nodo D la solución real implica detener su reloj 5 unidades de

tiempo, por lo que al cabo de 5 ticks más, todos los relojes tendrían el mismo valor. Es decir, en el instante 10040 todos tendrían el mismo valor en sus relojes.

3.- *a) Uno de los ordenadores tuviera un reloj defectuoso que sistemáticamente se retrasara un minuto por cada hora transcurrida. El resto de los nodos tienen relojes "normales" que no sufren ni avances ni retrocesos importantes en su progresión.*

Como el algoritmo de Berkeley calcula las medias entre los valores de todos los relojes, el error introducido implicaría retrasar el reloj 1/60 por cada unidad transcurrida. Si el resto de los ordenadores tienen un reloj "normal" lo que avance o se retrase de más cada uno de ellos se compensará con lo que hagan los demás. Por ello, el efecto neto de ese retraso sería que el valor calculado sufriría un retraso de 1/600 por cada unidad transcurrida en cada uno de los ajustes efectuados. Aun así, todos los nodos obtendrían un valor similar en cada ajuste. Es decir, los relojes de cada uno de ellos se seguirían sincronizando y todos tendrían valores muy próximos tras cada ajuste.

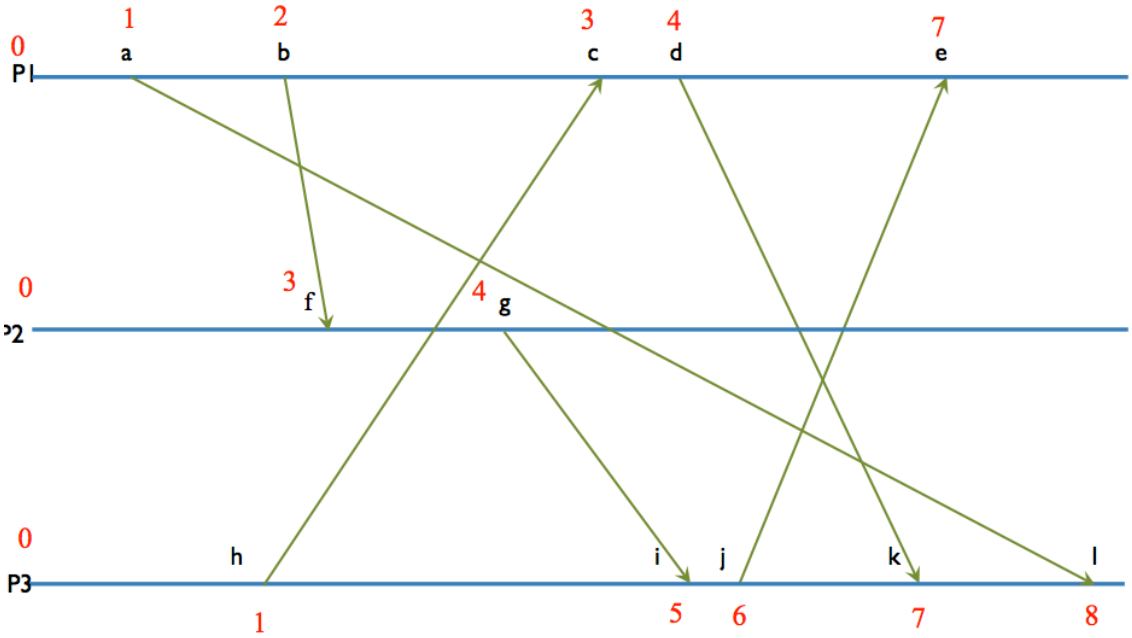
A pesar de no ser un algoritmo perfecto, pues no detecta ni descarta errores continuos por parte de alguno de los nodos participantes, el algoritmo de Berkeley sí que consigue mantener los relojes de un mismo sistema sincronizados. Eso (que todos los nodos tengan relojes sincronizados entre sí) se consideró más importante que el hecho de además lograr dicha sincronización también con el tiempo "real".

b) Ninguno de los ordenadores tuviese un reloj correcto pero los defectos que estos pudieran tener se compensaran entre sí.

En este segundo supuesto, además de lo obtenido en el primer apartado se podría conseguir que el valor mantenido en cada reloj esté razonablemente cerca de la hora real, si todos ellos fueron inicializados correctamente.

4.- *a) Valor del reloj lógico de Lamport para cada evento:*

a:	1	b:	C(a)+1 = 2	c:	max(C(h),C(b))+1=3
d:	C(c)+1=4	e:	max(C(d),C(j))+1=7	f:	max(0,C(b))+1=3
g:	C(f)+1=4	h:	1	i:	max(C(h),C(g))+1=5
j:	C(i)+1=6	k:	max(C(j),C(d))+1=7	l:	max(C(k),C(a))+1=8



b) Eventos concurrentes con el evento "i":

Serán concurrentes con "i" todos aquellos eventos desde los que no podamos encontrar un "camino dirigido" que termine o empiece en "i" y que los relacione con él. Hacia "i" encontramos dos "caminos":

a --> b --> f --> g --> i

h --> i

Desde "i" encontramos dos "caminos" más:

i --> j --> e

i --> j --> k --> l

Por tanto, los eventos que no hemos citado en ninguno de estos recorridos serán aquellos concurrentes con "i". En concreto, solo existirán los eventos "c" y "d" en el proceso P1 que cumplan tal condición. Ellos son los únicos concurrentes con "i".

c) Eventos concurrentes con el evento "e".

Hay que emplear los mismos criterios vistos en el apartado anterior, ahora sobre el evento "e". Hacia "e":

a --> b --> c --> d --> e

a --> b --> f --> g --> i --> j --> e

h --> i --> j --> e

No hay ningún camino que parta de "e" y lo relacione con otros eventos.

Por tanto, los únicos eventos que no se han citado en todos los recorridos listados arriba serían los eventos "k" y "l" en el proceso P3. Esos dos serían los únicos concurrentes con "e".

d) ¿Puede proporcionar un orden total de los eventos? Si es así, ¿cuál?

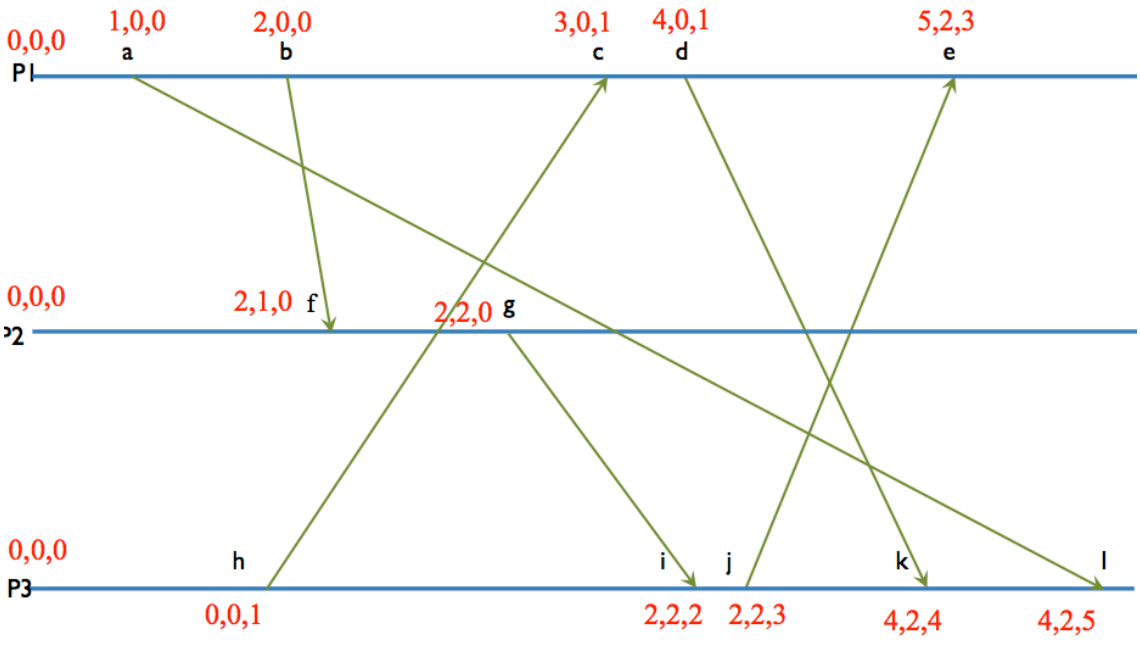
Si se añade el número de nodo como sufijo a los relojes lógicos de Lamport, entonces no se podrá obtener nunca dos valores iguales (pues en el mismo nodo o proceso los relojes lógicos ya proporcionaban valores diferentes; y en el caso de distintos nodos, si sus relojes lógicos eran iguales, al añadir el sufijo con el número de nodo, ahora ya serán diferentes).

El orden total en este caso es (se indica entre paréntesis el evento al que hace referencia):

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

5.- a) Relojes vectoriales de los eventos:

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) Listado de pares de eventos concurrentes entre sí:

Basándonos en los relojes vectoriales asignados a cada evento, podremos decir que dos eventos A y B son concurrentes cuando, comparando los relojes vectoriales componente por componente, encontremos algunas posiciones en las que el valor de A sea menor que el de B y en otras posiciones ocurra lo contrario.

El evento “a” ([1,0,0]) es concurrente solo con “h” ([0,0,1]).

El evento “b” también pues puede observarse que todos los demás (a excepción de “a”, que fue causa de “b”) tienen un valor igual o superior a los que presenta “b” en todas sus componentes.

El evento “c” [3,0,1] es concurrente con “f” [2,1,0], “g” [2,2,0], “i” [2,2,2] y “j” [2,2,3].

El evento “d” [4,0,1] es concurrente con “f” [2,1,0], “g” [2,2,0], “i” [2,2,2] y “j” [2,2,3].

El evento “e” [5,2,3] es concurrente con “k” [4,2,4] y “l” [4,2,5].

El evento “f” [2,1,0] es concurrente con “c” [3,0,1] , “d” [4,0,1] y “h” [0,0,1].

El evento “g” [2,2,0] es concurrente con “c” [3,0,1] , “d” [4,0,1] y “h” [0,0,1].

El evento “h” [0,0,1] es concurrente con “a” [1,0,0], “b” [2,0,0], “f” [2,1,0] y “g” [2,2,0].

El evento “i” [2,2,2] es concurrente con “c” [3,0,1] y “d” [4,0,1].

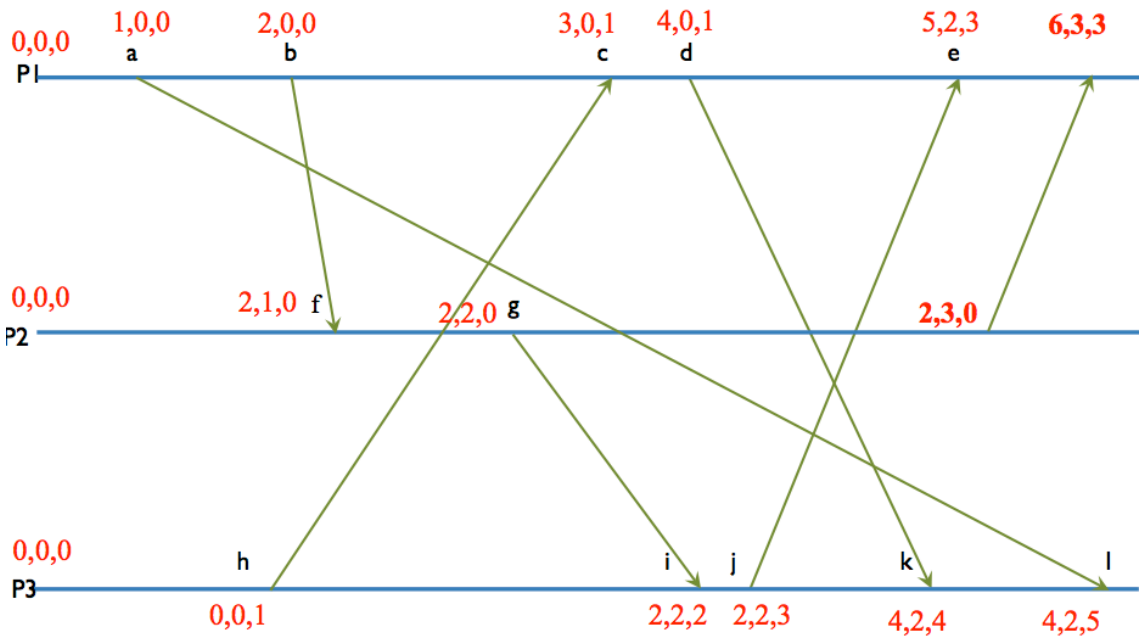
El evento “j” [2,2,3] es concurrente con “c” [3,0,1] y “d” [4,0,1].

El evento “k” [4,2,4] es concurrente con “e” [5,2,3].

El evento “l” [4,2,5] es concurrente con “e” [5,2,3].

c) Como el evento de envío se da en P2, dicho evento tendría los mismos valores en las componentes 1 y 3 que tuvo el evento “g” y el valor 3 en la componente 2 (por ser el tercer evento que ocurre en ese proceso P2). Por tanto, su valor sería [2,3,0].

Por su parte, el evento de recepción en P1 tendría el valor 6 en su componente 1 (pues sería el sexto evento ocurrido en P1) y el 3 en su componente 2 (pues solo ha habido tres eventos en P2 y el último de ellos fue el envío de ese mensaje, por lo que tal valor se llega a comunicar a P1), manteniendo el mismo valor que ya se dio en “e” en su componente 3. Así, su valor sería [6,3,3].



6.- No se podría utilizar, pues el algoritmo de Chandy y Lamport sólo funcionará si los canales utilizados realizan una entrega en orden FIFO y son fiables. En la traza mostrada en la actividad 3 no se respeta el orden de entrega FIFO pues P1 envía dos mensajes a P3 en los eventos "a" y "d", respectivamente, pero tales mensajes se entregan en orden contrario al que fueron enviados.

El orden FIFO resulta necesario para garantizar una correcta recolección del estado de los canales.

7.- *Explique qué traza seguiría este algoritmo de Chandy y Lamport y qué mensajes en tránsito recogería en cada canal.*

Como P1 recibe su primer mensaje MARCA antes del evento "c", anota entonces su estado local y pasa a "abrir" el registro de sus canales de entrada. Antes de que pueda difundir su mensaje MARCA hacia P2 y P3 recibe el mensaje que P3 envió en el evento "h" y lo anota como contenido del canal P3->P1. Inmediatamente después difunde su mensaje MARCA hacia P1 y P3 (antes de que se diera el evento "d").

A su vez, P3 recibe su primer MARCA antes del evento "j" según dice el enunciado. Dicho mensaje fue enviado por P2 y se habrá entregado después de que se diera el evento "i", pues tal MARCA fue difundida tras el evento "g" (según dice el enunciado) y los canales son FIFO. Tras haber recibido ese mensaje, P3 difunde un mensaje MARCA hacia P1 y P2. En P1 se entregará antes del evento "e" para respetar el orden FIFO en las entregas, en P2 se entregará en algún momento, pero como no hay otros eventos en la traza para ese proceso, poco importará cuándo suceda esto. Como resultado de esta difusión, ni P1 ni P2 llegarán a registrar ningún mensaje en tránsito dentro de los canales P3->P1, P3->P2.

Por último, la difusión de P1 (respetando el orden FIFO) llegará a P3 entre los eventos "k" y "l". Por ello, al ocurrir "k" en P3 éste anotará que el mensaje emitido por P1 en el evento "a"

estaba en tránsito en el canal P1->P3. Entre tanto, también llegará a P2 pero en ese proceso no se llegará a registrar ningún mensaje en tránsito.

Con todo esto, todos los procesos han llegado a recibir un mensaje MARCA por cada uno de sus canales tras haber recibido el que inició el algoritmo. Los únicos mensajes en tránsito que se llegan a registrar son:

1) Canal P3->P1, mensaje emitido en el evento "h".

2) Canal P1->P3, mensaje emitido en el evento "a".

Y los estados locales de cada proceso son:

1) P1: estado que mantenía justo antes del evento "c".

2) P2: estado que mantenía inmediatamente después del evento "g".

P3: estado que mantenía entre los eventos "i" y "j".

8.- Justificación de las afirmaciones:

1. El algoritmo de Bully falla si dos o más nodos inician el algoritmo de forma simultánea (es decir, si hay más de un iniciador). *JUSTIFICACIÓN*: *Si hay más de un iniciador, el algoritmo simplemente se repite (una vez por cada iniciador). Y en todos los casos se obtendrá el mismo líder, que es el nodo con el mayor identificador (asumiendo que no se han producido fallos en los nodos, o, al menos, que el de mayor identificador no ha fallado).*	F
El algoritmo de elección de líder sobre anillos falla si dos o más nodos inician el algoritmo de forma simultánea (es decir, si hay más de un iniciador). *JUSTIFICACIÓN*: *Si hay más de un iniciador, el algoritmo simplemente se repite (una vez por cada iniciador). Cada iniciador, cuando reciba el listado final, selecciona el identificador mayor. Si asumimos que no se han producido fallos en los nodos, o, al menos, que el de mayor identificador no ha fallado, todos los iniciadores existentes determinarán que el líder es el mismo.*	F
2. El algoritmo centralizado de exclusión mutua visto en clase limita la escalabilidad y la tolerancia a fallos porque el coordinador supone un cuello de botella y un punto de fallo único. *JUSTIFICACIÓN: Si el coordinador falla, la información sobre quién está en la sección crítica y quién está esperando a entrar se pierde.*	V

3. El algoritmo distribuido de exclusión mutua visto en clase utiliza relojes lógicos para ordenar algunas solicitudes de entrada a la sección crítica. *JUSTIFICACIÓN: El uso de relojes lógicos permite a un nodo A, que desea entrar en la sección crítica, desempatar cuando recibe una petición de entrada a la sección crítica por parte de otro nodo B. Al usar relojes lógicos, el nodo A puede determinar si su solicitud se envió antes o después de que el nodo B también enviase su solicitud. Y así puede determinar si debe enviar el mensaje de "OK" al nodo B (si es que B solicitó entrar antes que A), o bien no contestar a B (si es que A solicitó antes entrar).*	V
4. El algoritmo de Berkeley es uno de los algoritmos de exclusión mutua más eficientes. *JUSTIFICACIÓN: El algoritmo de Berkeley permite sincronizar relojes físicos.*	F
5. El algoritmo distribuido de exclusión mutua no necesita realizar ninguna acción en su protocolo de salida. *JUSTIFICACIÓN: Cuando un nodo sale de su sección crítica, debe enviar el mensaje de OK a los nodos que le habían solicitado entrar.*	F
6. El algoritmo de Cristian es un algoritmo de elección de líder muy eficiente. *JUSTIFICACIÓN: Es un algoritmo de sincronización de relojes físicos.*	F

9.- Justificación de las afirmaciones:

1. El algoritmo de Chandy-Lamport requiere que los canales no pierdan ni desordenen mensajes. JUSTIFICACIÓN: Se requiere que los canales sean fiables y en orden FIFO.	V
2. En general es problemático hacer que el reloj de un ordenador retroceda. *JUSTIFICACIÓN*: No se debe ir nunca "hacia atrás" en el tiempo.	V
3. En el algoritmo de Berkeley no se asume que exista un ordenador con un reloj preciso. *JUSTIFICACIÓN: Es en el algoritmo de Cristian donde sí hay un ordenador con un reloj muy preciso.*	V
4. El algoritmo de Berkeley sincroniza los relojes de los nodos de un determinado sistema distribuido, sin importarle la divergencia que pueda haber entre estos relojes y la hora "oficial". *JUSTIFICACIÓN: No se asume ningún reloj preciso y los relojes se sincronizan entre sí, por lo que los valores resultantes tras aplicar el algoritmo pueden diferir mucho de la hora oficial.*	V

5. El algoritmo de Cristian se utiliza para gestionar relojes lógicos. *JUSTIFICACIÓN: Se emplea para sincronizar relojes físico.*	F
6. El algoritmo de Cristian proporciona la base necesaria para implantar cualquier algoritmo descentralizado. *JUSTIFICACIÓN: Se trata de un algoritmo centralizado, pues el coordinador o servidor tiene toda la información global, ya que conoce la hora oficial del sistema.*	F
7. El algoritmo de Chandy y Lamport requiere que los canales de comunicación respeten un orden FIFO. *JUSTIFICACIÓN: Se requiere que si se envían dos mensajes m1 y m2 por el mismo canal, estos mensajes deben llegar siempre en ese mismo orden de envío (es decir, en orden m1 y m2).*	V
8. El algoritmo de Cristian permite identificar los mensajes en tránsito por cada uno de los canales de comunicación de un sistema distribuido. *JUSTIFICACIÓN: Esto lo hace el algoritmo de Chandy-Lamport*	F