1: Describa los siguiente items y técnicas de los sistemas distribuidos:
• ***1.1: Relojes.***

Los computadores pueden disponer de su propio reloj físico, estos relojes son dispositivos electrónicos que cuentan las oscilaciones que ocurren en un cristal a una frecuencia definida, y que normalmente dividen esta cuenta y almacenan el resultado en un registro contador. Los dispositivos de reloj pueden estar programados para generar impulsos a intervalos regulares con el fin de que, por ejemplo, pueda ser implementada la división de tiempo.

El sistema operativo lee el valor del reloj hardware del nodo, la escala y añade una compensación para producir un reloj software que mide aproximadamente el tiempo real, físico para el proceso.

Cuando el tiempo real en un marco de referencia absoluto es t, Ci (t) es la lectura del reloj software. En general, el reloj no es suficientemente preciso, y por tanto Ci (t) difiere de t. sin embargo, si Ci se comporta suficientemente bien, podemos utilizar su valor para marcar el tiempo de cualquier evento en Pi. Hay que indicar que eventos sucesivos corresponderán con marcas de tiempo diferentes solo si la resolución del reloj, el período entre actualización del valor del reloj, es mas pequeño que el intervalo de tiempo entre eventos sucesivos. La tasa a la que ocurren los sucesos depende de factores tales como la longitud del ciclo de instrucción del procesador.

Fuente: http://www.itistmo.edu.mx/Pag%20Informatica/APUNTES_archivos/page0020.htm

• ***1.2: Sesgo y deriva de reloj.***

Los relojes de los computadores, como los otros, tienden a no estar en perfecto acuerdo. La diferencia instantánea entre las lecturas de dos relojes cualesquiera se llama **sesgo**. También los relojes basados en cristal utilizados en los computadores están sujetos a derivas de reloj, que significa que ellos cuentan el tiempo a diferentes ritmos, y por lo tanto divergen. Los osciladores subyacentes están sujetos a variaciones físicas, con la consecuencia que sus frecuencias de oscilación difieren. Aun más la misma frecuencia de los relojes varía con la temperatura. Existen diseños que intentan compensar misma frecuencia de los relojes. Existen diseños que intentan compensar esta variación, pero no lo pueden eliminar. La diferencia en el período de oscilación entre dos relojes puede ser extremadamente pequeña, pero la diferencia acumulada sobre muchas oscilaciones conduce a una diferencia observable en los contadores registrados por los relojes, independientemente de con que precisión fueron inicializados.

Un ritmo **deriva de reloj** es el cambio en la compensación entre el reloj de referencia nominal perfecto por unidad de tiempo medido, por el reloj de referencia. Para relojes ordinarios basados en un cristal de cuarzo, suele ser de aproximadamente 10-6 segundos /segundo, dando una diferencia de 1 segundo cada 1.000.000 segundos, o 11,6 días. El ritmo de deriva de los relojes de cuarzo de alta precisión es de aproximadamente 10-7 o 10-8.

Fuente: http://www.itistmo.edu.mx/Pag%20Informatica/APUNTES_archivos/page0020.htm

• 1.3: Tiempo Universal Coordinado.

Los relojes de computadora pueden sincronizarse con fuentes externas de tiempo de gran precisión. Los relojes físicos más precisos utilizan osciladores atómicos, cuyo ritmo de deriva es aproximadamente de una parte en 1013. La salida de estos relojes atómicos se utiliza como estándar para el tiempo real transcurrido, conocido como tiempo atómico internacional.

UTC es un estándar internacional de cronometraje. Esta basado en el tiempo atómico, aunque ocasionalmente se inserta un salto de un segundo o más raramente, borrado, para mantenerse en sintonía con el tiempo astronómico. Las señales se sincronizan y difunden regularmente desde las estaciones de radio terrestre y los satélites, que cubren muchas partes del mundo.

Hay disponibles receptores comerciales. Comparados con UTC perfecto, las señales recibidas desde las estaciones terrestres tienen una precisión de milisegundos, dependiendo de la estación utilizada. Las señales recibidas desde GPS tiene una precisión de alrededor de 1 microsegundo. Los computadores con receptores adjunto pueden sincronizar sus relojes con estas señales de tiempo. Los computadores pueden también recibir el tiempo con una precisión de unos pocos milisegundos sobre una línea telefónica con organizaciones tales como el Instituto Nacional para los Estándares y Tecnología en USA.

Fuente: http://www.itistmo.edu.mx/Pag%20Informatica/APUNTES_archivos/page0020.htm

• 1.4: Sincronización de relojes físicos.

Para conocer en qué hora del día ocurren los sucesos, es necesario sincronizar los relojes de los procesos **Ci**, con una fuente de tiempo externa autorizada. Esto es la sincronización externa.
Si están sincronizados con otro con un grado de precisión conocido, entonces podemos medir el intervalo entre dos eventos que ocurren en diferentes computadores llamando a sus relojes locales, incluso aunque ellos no estén necesariamente sincronizados con una fuente externa de tiempo. Esto es sincronización interna. Definimos estos dos modos de sincronización más detalladamente como sigue, sobre un intervalo de tiempo real.

Sincronización externa: para una sincronización dada D>0, y para fuente S tiempo UTC, /Si(t) – Ci(t)<D, para i= 1,2 N y para todos los tiempos reales t en I. otra forma de decir esto es que los relojes Ci son precisos con el límite D.

Sincronización interna: para una sincronización dada D>0, /Ci(t) – Cj(t)/<D, para i= 1,2 N y para todos los tiempos reales t en I. otra forma de decir esto es que los relojes Ci concuerdan con el limite D.

Los relojes que están sincronizados internamente no están necesariamente sincronizados externamente, puesto que pueden desplazarse colectivamente desde una fuente de tiempo externa incluso aunque estén de acuerdo entre sí. Sin embargo, se deduce de las definiciones que si el sistema está sincronizado externamente con un límite D entonces el mismo sistema está sincronizado internamente con un límite de 2D.

Fuente: http://www.itistmo.edu.mx/Pag%20Informatica/APUNTES_archivos/page0020.htm

2: Describa los siguientes procedimientos:

• 2.1: Sincronización en un sistema síncrono.

El de sincronización interna entre dos procesos en un sistema distribuido síncrono es el caso más simple posible. En un sistema síncrono, se conocen los límites para el ritmo que deriva de los relojes, el máximo retardo de transmisión de mensajes y el tiempo para ejecutar cada paso de un proceso.

Un proceso envía el tiempo t de su reloj local a otro en un mensaje m. En principio el proceso receptor podría tener su reloj en el tiempo $t + T_{trans}$, donde T_{trans} es el tiempo preciso para transmitir m entre ellos. Los dos relojes podrían coincidir. Desafortunadamente T_{trans} esta sujeto a variaciones y es desconocido, a pesar de todo, hay siempre un tiempo mínimo de transmisión min. que podría obtenerse si no se ejecutara ningún otro proceso y no existiera mas tráfico en la red; min. puede ser medido o estimado de forma conservadora.

En un sistema síncrono, por definición hay también un limite superior max del tiempo tomado para transmitir cualquier mensaje. Sea la incertidumbre en el tiempo de transmisión del mensaje u, donde u = (max – min). En general, para un sistema síncrono, el limite optimo que puede ser conseguido en el sesgo de reloj cuando sincronizamos N relojes es u(1 – 1/N).

La mayoría de los sistemas distribuidos encontrados en la práctica son asíncronos: los factores que conducen a retardos de los mensajes no están limitados en su efecto, y no hay limite superior max en los retardos de transmisión de mensajes.

• 2.2: Método de Cristian para sincronizar relojes.

Cristian (1989) sugirió la utilización de un servidor de tiempo, conectado a un dispositivo que recibe señales de una fuente UTC, para sincronizar computadores externamente. Bajo solicitud, el proceso servidor S proporciona el tiempo de acuerdo con su reloj.

Cristian observó que aunque no hay límite superior en los retardos de transmisión de mensajes en un sistema asíncrono, los tiempos de ida y vuelta de los mensajes intercambiados entre cada par de procesos son a menudo razonablemente cortos. El describe el algoritmo como probabilístico: el método consigue sincronización sólo si los tiempos de ida y vuelta entre el cliente y el servidor son suficientemente cortos comparados con la precisión requerida.

Un proceso *p* solicita el tiempo en un mensaje m_r, y recibe el valor del tiempo *t* en un mensaje m_t .el proceso *p* registra el tiempo total de ida y vuelta T_{round} tomando para enviar la solicitud m_r y escribir la respuesta m_t . Se puede este tiempo con precisión razonable si su ritmo de deriva de reloj es pequeño.

Una estimación sencilla del tiempo al que *p* debe fijar su reloj es $t + T_{round}/2$, que supone que el tiempo transcurrido se desdobla igualmente antes y después de que S coloque t en m_r . Esto es normalmente una suposición razonable de precisión, a menos que los dos mensajes sean transmitidos sobre redes diferentes. Si el valor del tiempo mínimo de transmisión *min* es conocido o puede ser estimado conservativamente, entonces podemos determinar la precisión de este resultado como sigue.

El instante más temprano en el que S podría haber colocado el tiempo en m_t fue *min* después de que *p* enviara m_r . El punto más tarde al que podría haberse hecho esto sería *min* antes de que m_t llegue a *p* . El tiempo del reloj de S cuando el mensaje de respuesta llega estará en el rango $|t + min, t + T_{round} - min|$.

La anchura de este rango es $T_{round} - 2min$, por lo que la precisión es

$$\pm (T_{round}/2 - min).$$

Se puede tratar con la variabilidad en alguna medida haciendo varias solicitudes a S y tomando el valor mínimo de T_{round} para conseguir una estimación más precisa. Cuanto más grande es la precisión requerida, más pequeña es la probabilidad de conseguirla.

DISCUSIÓN DEL ALGORITMO DE CRISTIAN: el método de Cristian sufre el problema asociado con todos los servicios implementados por un servidor único, por los que el único servidor de tiempo puede caer y hacer que la sincronización sea imposible temporalmente. Cristian sugirió que el tiempo debería ser proporcionado por un grupo de servidores de tiempo sincronizados, cada uno con un receptor para señales de tiempo UTC.

Hay que tener en cuenta que un servidor de tiempo erróneo que responda con valores espurios (erróneos) de tiempo, o un servidor de tiempo impostor que responde con tiempos erróneos deliberadamente, podrían causar muchos daños en un sistema de computadores. Estos problemas fueron más allá del alcance del trabajo descrito por Cristian |1989|, que supone que las señales de tiempo externo se auto comprueban. Cristian y Fetzer |1994| describen una familia de protocolos probabilísticos para la sincronización de relojes internos.

El problema de tratar con relojes defectuosos se aborda paralelamente por el algoritmo de Berkeley.

• 2.3: Algoritmo de Berkeley.

Cristian y Zatty |1989| describieron un algoritmo para sincronización interna que ellos desarrollaron para colecciones de computadores ejecutando el UNIX Berkeley. En este algoritmo se elige, un computador coordinador para actuar como maestro. A diferencia del protocolo de Cristian, este computador consulta periódicamente a los otros computadores cuyos relojes están para ser sincronizados, llamados esclavos. Los esclavos le devuelven sus valores de reloj. El maestro estima sus tiempos locales de reloj observando los tiempos de ida y vuelta. El balance de las probabilidades es que este promedio contrarresta las tendencias de los relojes individuales a funcionar rápido o lento. La precisión del protocolo depende de un tiempo de ida y vuelta máximo nominal entre el maestro y los esclavos. El maestro elimina cualquier lectura adicional asociado a tiempos más grandes que el máximo.

En lugar de reenviar el tiempo actual actualizado a los demás computadores, que introduciría una nueva imprecisión debido al tiempo de transmisión del mensaje, el maestro envía la cantidad que precisa el esclavo para hacer su ajuste.

El algoritmo elimina las lecturas de relojes defectuosos. El maestro toma un promedio tolerante a fallos. Esto es, se elige un subconjunto de los relojes que no difieran entre ellos más de una determinada cantidad, y el promedio se toma de las lecturas de sólo estos relojes

• 2.4: Protocolo del tiempo de red.

El método de Cristian y el algoritmo de Berkeley están pensados principalmente para utilizar en intranets. El Protocolo de Tiempo de Red (Network Time Protocol, NTP) define una arquitectura para un servicio de tiempo y un protocolo para distribuirla información del tiempo sobre Internet.

Los objetivos principales de diseño de NTP son los siguientes:

Proporcionar un servicio que permita a los clientes a lo largo de Internet estar sincronizados de forma precisa a UTC: NTC emplea técnicas estadísticas para el filtrado de los datos de tiempo y discrimina entre la calidad de los datos de tiempo de los diferentes servidores.

Proporcionar un servicio fiable que pueda sobrevivir a pérdidas largas de conectividad: los servidores pueden reconfigurarse para continuar proporcionando el servicio si uno de ellos llega a ser inalcanzable.

Permitir a los clientes resincronizar con suficiente frecuencia para compensar las tasas de deriva encontradas en la mayoría de los computadores.

Proporcionar protección contra la interferencia con el servicio de tiempo, ya sea maliciosa o accidental.

El servicio de NTP está proporcionado por una red de servidores localizados a través de Internet. Los servidores primarios están conectados directamente a una fuente de tiempo como un radio-reloj recibiendo UTC; los servidores secundarios están sincronizados con servidores primarios. Los servidores están conectados en una jerarquía lógica llamada subred de sincronización cuyos niveles se llaman estratos. Los servidores primarios ocupan el estrato 1: ellos están en la raíz. Los servidores del estrato 2 son servidores secundarios que están sincronizados directamente con los servidores primarios; los del estrato 3 están sincronizados con los del estrato 2; y así sucesivamente. Los servidores del nivel más bajo (hojas) se ejecutan en las estaciones de trabajo de los usuarios.

Los relojes de los servidores con números alto de estrato tienen tendencia a ser menos fiables que aquellos con números bajos de estrato, porque se introducen errores en cada nivel de sincronización.

La subred de sincronización se puede reconfigurar cuando los servidores llegan a ser inalcanzables o se producen fallos.

Los servidores NTP se sincronizan entre sí en uno de estos 3 modos: multidifusión, llamada a procedimiento y modo simétrico. El modo multidifusión está pensado para su uso en una LAN de alta velocidad. Uno o más servidores reparten periódicamente el tiempo a los servidores que se ejecutan en otros computadores conectados en la LAN, que fijan sus relojes suponiendo un pequeño retardo. Este método puede alcanzar sólo precisiones relativamente bajas, pero que no obstante son consideradas suficientes para muchos propósitos.

El modo de llamada a procedimiento es similar al funcionamiento del algoritmo de Cristian, un servidor acepta solicitudes de otros computadores, que el procesa respondiendo con su marca de tiempo. Este modo es adecuado donde se requieren precisiones más altas que las que se pueden conseguir con multidifusión, o donde la multidifución no viene soportada por hardware.

El modo simétrico está pensado para su utilización por servidores que proporcionan información del tiempo en LANs y por los niveles más altos de la subred de sincronización. Un par de servidores operando en modo simétrico intercambian mensajes llevando información del tiempo. Los datos del tiempo son retenidos como parte de una asociación entre los servidores que se mantiene con el fin de mejorar la precisión de su sincronización en el tiempo.

En todos los modos, los mensajes se entregan de modo no fiable, utilizando el protocolo de transporte estándar Internet UDP. En el modo de llamada a procedimiento y en el simétrico, los procesos intercambian pares de mensajes. Cada mensaje lleva marcas de tiempo de los sucesos del mensaje reciente: los tiempos locales cuando el mensaje anterior NTP entre el par fue enviado y recibido y el tiempo local cuando el mensaje actual fue transmitido. El receptor del mensaje NTP anota el tiempo local cuando el recibe el mensaje. En el modo simétrico, a diferencia del algoritmo de Cristian puede haber un retardo no despreciable entre la llegada de un mensaje y el envío del siguiente. También, se pueden perder mensajes, pero las tres marcas de tiempo llevadas por cada son sin embargo válidas.

Por cada par de mensajes enviados entre dos servidores NTP calcula una compensación **o** , que es una estimación de la deriva actual entre los dos relojes, y un retardo $\mathbf{d_i}$, que es el tiempo total de transmisión para los dos mensajes. Si el verdadero desplazamiento del reloj **B**

con relación al de A es **o**, y si los tiempo de transmisión actuales para **m** y **m'** son **t** y **t'** relativamente, entonces tenemos:

$T_{i-2} = T_{i-3} + t + \mathbf{o}$ y $T_i = T_{i-1} + t' + \mathbf{o}$ Esto conduce a: $d_i = t + t' = T_{i-2} - T_{i-3} + T_i - T_{i-1}$

Los servidores NTP aplican un algoritmo de filtrado de datos a pares sucesivos $< \mathbf{o_i, d_i} >$, que estima la deriva **o** y calculan la calidad de esta estimación como una cantidad estadística llamada *el filtro de dispersión*. Un filtro de dispersión relativamente alto implica datos relativamente poco fiables. Los ocho pares $< \mathbf{o_i, d_i} >$ más recientemente retenidos. Como con el algoritmo de Cristian, se elige como estimación de **o** el valor $\mathbf{o_j}$ que corresponde con el mínimo valor $\mathbf{d_i}$.

El valor del desplazamiento derivado de la comunicación con una única fuente no se utiliza necesariamente en sí mismo para controlar el reloj local. En general un servidor NTP se conecta en los intercambios de mensajes con varios de sus iguales, o colegas. Además del filtrado de datos aplicados a los intercambios con cada igual, NTP aplica un algoritmo de selección de colega. Éste examina los valores obtenidos de los intercambios con cada uno de los distintos colegas, buscando los valores relativamente menos fiables. La salida de este algoritmo puede provocar que un servidor cambie el colega que utilizaba inicialmente para la sincronización.

Los colegas con número de estrato más bajo son menos favorecidos que aquellos en el estrato más alto porque están próximos a las fuentes de tiempo primarias. Aquellos con la dispersión de sincronización más baja son favorecidos relativamente. Ésta es la suma de las dispersiones del filtro medidas entre el servidor y la raíz de la subred sincronización.

NTP emplea un modelo de bucle de bloqueo de fase que modifica la frecuencia de actualización del reloj de acuerdo con las observaciones de su tasa de deriva.

Fuente: http://exa.unne.edu.ar/depar/areas/informatica/SistemasOperativos/MonogSO/TIEMEG02.htm

3: Describa los siguientes conceptos relacionados con tiempo y relojes lógicos:
• ***3.1: Relojes lógicos.***

Los sucesos están ordenados de forma única por los tiempos mostrados en el reloj lógico. Puesto que no podemos sincronizar perfectamente los relojes a lo largo de un sistema distribuido, no podemos usar, en general, el tiempo físico para obtener el orden de cualquier par arbitrario de sucesos que ocurran en él.

Podemos utilizar un esquema que es similar a la causalidad física, pero que se aplica en los sistemas distribuidos, para ordenar algunos de los sucesos que ocurren en diferentes procesos. Esta ordenación está basada en dos puntos sencillos e intuitivamente obvios.

Cuando se envía un mensaje entre procesos, el suceso de enviar el mensaje ocurrió antes del de recepción del mismo.

Lamport llamó a la ordenación parcial obtenida al generalizar estas dos relaciones la realización suceder antes.

La relación captura un flujo de información entre dos eventos. Nótese, sin embargo, que en principios la información puede fluir de formas distintas de la de paso de mensajes. Por ejemplo, si Pérez presenta un mandato a su proceso para que envíe un mensaje, acto seguido telefonea a Gómez, quien ordena a su proceso que envíe otro mensaje, luego el envío del primer mensaje claramente sucedió antes que el segundo. Desafortunadamente, como no se han enviado mensajes de red entre los procesos que los emitieron, no podemos modelar este tipo de relaciones en nuestro sistema.

Otro punto a señalar es que aun produciéndose la relación sucedió antes entre dos sucesos, el primero podría o no haber causado realmente el segundo. Un proceso podría recibir un mensaje y consecuentemente enviar otro mensaje, pero no que él emite cada cinco minutos en cualquier caso y no tiene ninguna relación específica con el primer mensaje. No se ha supuesto ninguna causalidad real, pero la relación debe ordenar estos sucesos.

Lamport inventó un mecanismos simple por el que la relación “sucedió antes” puede capturarse numéricamente, denominado **reloj lógico**. Un reloj lógico de Lamport es un contador software que se incrementa monótonamente, cuyos valores no necesitan tener ninguna relación particular con ningún reloj físico.

Fuente: http://www.itistmo.edu.mx/Pag%20Informatica/APUNTES_archivos/page0020.htm

• 3.2: Relojes lógicos totalmente ordenados.

Algunos pares de sucesos distintos, generados por diferentes procesos, tienen marcas de tiempo Lamport numéricamente idénticas .Se puede crear orden total sobre los sucesos , uno, para el que todos los pares de sucesos distintos están ordenados (teniendo en cuenta identificadores de procesos en los que ocurren los sucesos).

E es un suceso que ocurre en Pi
Tiempo local Ti.
E' es un suceso que ocurre en Pj
Tiempo local Tj.
Se deben definir las marcas de tiempos globales para dichos sucesos como: (Ti, i) y (Tj, j); definiendo (Ti, i) es menor o igual (Tj,j), si y sólo si , Ti es menor que Tj ò Ti es igual a Tj, siendo i menor que j.

Lamport utilizó este tipo de ordenación para la entrada de procesos en una sección crítica.

Fuernte: http://exa.unne.edu.ar/depar/areas/informatica/SistemasOperativos/MonogSO/TIEMEG02.htm

• 3.3: Relojes vectoriales.

Mattern y Fidge desarrollaron relojes vectoriales para vencer la deficiencia de los relojes de Lamport, del hecho que no podemos deducir que un reloj vectorial para un sistema de N procesos es un vector de N enteros.

Cada proceso mantiene su propio reloj vectorial Vi, que utiliza para colocar marcas de tiempo en los sucesos locales. Como las marcas de tiempo de Lamport, cada proceso adhiere el vector de marcas de tiempo en los mensajes que envía al resto, y hay unas reglas sencillas para actualizar los relojes.

Los vectores de marcas de tiempo tienen la desventaja, comparados con las marcas de tiempo de Lamport, de precisar una cantidad de almacenamiento y de carga real de mensajes que es proporcional a N, el número de procesos. Charron Bost mostró que, si somos capaces de decir N es inevitable. Sin embargo, existen técnicas para almacenar y transmitir cantidades más pequeñas de datos, a costa del procedimiento precisado para reconstruir los vectores completos. Raynal y Singha dan cuenta de algunas de estas técnicas. También describen la noción de relojes matricules, en que los procesos mantienen estimaciones de los vectores de tiempo de otros procesos así como las suyas propias.

Fuente: http://www.itistmo.edu.mx/Pag%20Informatica/APUNTES_archivos/page0020.htm

4: Describa los siguientes puntos:

• 4.1: Estados globales y cortes consistentes.

Es posible observar la sucesión de estados de un proceso individual, pero la cuestión de cómo establecer un estado del sistema, el estado de la colección de procesos, es más difícil de tratar.

El problema esencial es la ausencia de tiempo global. Debiéramos preguntarnos si podemos elaborar un estado global significativo de los estados locales registrados en diferentes tiempos reales. La respuesta es un sí matizado, pero para ver esto debemos presentar antes algunas definiciones.

Consideremos a un sistema general φ de N procesos $p_i (i = 1, 2..., N)$, cuya ejecución deseamos estudiar. Ocurre una serie de eventos en cada proceso, y podemos caracterizar la ejecución de un proceso por su historia:

$$historia(p_i) = h_i = <e^0_i, e^1_i, e^2_i, ...>$$

De forma semejante, podemos considerar cualquier prefijo finito de la historia del proceso:

$$h^k_i = <e^0_i, e^1_i, ..., e^k_i>$$

Cada evento es una acción interna del proceso (por ejemplo, la actualización de una de sus variables) o el envío o la recepción de un mensaje sobre los canales de comunicación que conectan los procesos.

Podemos registrar qué ocurrió con la ejecución de φ. Cada proceso puede registrar los sucesos que se producen allí, y la sucesión de estados por la que pasa. Denotamos con s^k_i el estado del proceso p_i inmediatamente antes que ocurra el suceso k, por lo que e^0_i es el estado inicial de p_i.

Podemos formar la historia global de φ como la unión de las historias individuales de los procesos:

$$H = h_0 \grave{E} h_1 \grave{E} .... \grave{E} h_{N-1}$$

Podemos tomar cualquier conjunto de estados de los procesos individuales para formar un estado global $(s_1, s_2, ..., s_N)$. Pero qué estados son significativos; esto es, ¿cuál de los estados de los procesos podrían haber ocurrido al mismo tiempo? Un estado global corresponde a los prefijos iniciales de las historias individuales de los procesos. Un *corte* de la ejecución del sistema es un subconjunto de su historia global que es la unión de los prefijos de las historias de los procesos:

$$C = h^{C1}_1 \grave{E} h^{C1}_1 \grave{E} .... \grave{E} h^{CN}_N$$

Consideramos los sucesos que ocurren en los procesos p_1 y p_2 mostrados en la figura 1. La figura representa dos cortes, uno con frontera $<e^0_1 , e^0_2>$ y otro con la frontera $<e^2_1 , e^2_2>$. El corte de más a la izquierda es *inconsistente*. Esto es porque p_2 incluye la recepción del mensaje m_1, pero en p_1 no incluye el envío del mensaje. Esto se considera como un *efecto* sin una *causa*. La ejecución actual no estuvo nunca en un estado global correspondiente a los estados del proceso en la frontera. Por el contrario, el corte de la derecha es *consistente*. Incluye tanto el envío como la recepción del mensaje m_1. Incluye el envío pero no la recepción del mensaje m_2. Esto es consistente con la ejecución actual; después de todo, el mensaje se tomó algún tiempo para llegar.

Un corte *C* es consistente si, para cada suceso que contiene, también contiene todos los sucesos que *sucedieron antes* del suceso.

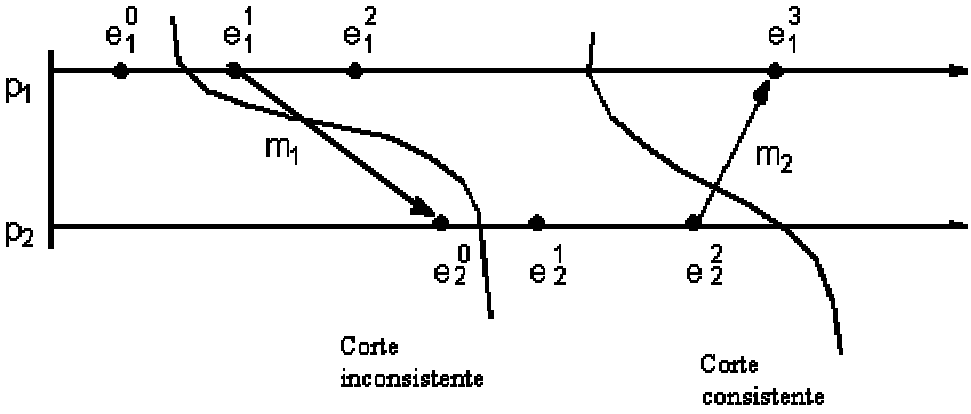

Figura 1. Cortes

Un *estado global consistente* es uno que corresponde con un corte consistente. Podemos caracterizar la ejecución de un sistema distribuido como una serie de transiciones entre los estados globales del sistema:

$S_0 \rightarrow S_1 \rightarrow S_2 \ldots$

Una *ejecución* es una ordenación de orden total de todos los sucesos en una historia global que es consistente con cada ordenación de la historia local. Una *linealización o ejecución consistente* es una ordenación de los sucesos en una historia global que es consistente con su relación sucedió antes ® en *H*. Una linealización es también una ejecución.

No todas las ejecuciones pasan a través de estados consistentes globales, pero todas las linealizaciones pasan sólo a través de estados globales consistentes.

• *4.2: Predicados de estado global, estabilidad, seguridad y vitalidad.*

Detectar una condición tal, como un bloqueo indefinido o una terminación equivale a evaluar un predicado de estado global. Un predicado de estado global es una función que pasa del conjunto de estados globales de los procesos en el sistema a verdadero, falso.

Un sistema que esta siendo bloqueado indefinidamente o un sistema que esta terminando es que todos son estables: una vez que el sistema entra en un estado, permanece en todos los estados futuros alcanzables desde ese estado. Por contraste, cuando monitorizamos o depuramos una aplicación a menudo estamos interesados en predicados no estables.

Dos nociones relevantes para los predicados de estado global: seguridad y vitalidad. Supongamos que hay una propiedad indeseable a que es un predicado del estado global del sistema: por ejemplo, a podría ser la propiedad de estar bloqueado indefinidamente. Sea S_0 el estado original del sistema. Seguridad con respecto a a es la aserción que a evalúa a falso para todos los estados S alcanzables desde S_0. A la inversa, sea b una propiedad deseable del estado global del sistema: por ejemplo la propiedad de alcanzar la terminación. Vitalidad con respecto a b es la propiedad que, para cualquier linealización L comenzando en el estado S_0, b se evaluara a verdadero para algún estado S_1 alcanzable desde S_0.

• 4.3: Algoritmo de Instantánea de Chandy y Lamport.

El objetivo del algoritmo es registrar un conjunto de estados de procesos y canales (una instantánea) para un conjunto de procesos P_i (i = 1,2,...n) tal que, incluso a través de una combinación de estados registrados que nunca podrían haber ocurrido al mismo tiempo, el estado global registrado sea consistente.

El algoritmo supone que:

· No falla ni los canales ni los procesos; la comunicación es fiable por lo que cada mensaje enviado es recibido intacto, exactamente una vez.

· Los canales son unidireccionales y proporcionan la entrega de los mensajes con ordenación FIFO.

· El grafo de los procesos y canales esta fuertemente conectado (hay un recorrido entre dos procesos cualquiera.)

· Cualquier proceso puede iniciar una instantánea global en cualquier instante.

· Los procesos pueden continuar su ejecución y enviar y recibir mensajes normales mientras tiene lugar la instantánea.

La idea esencial del algoritmo es como sigue. Cada proceso registra su estado y para cada canal entrante también un conjunto de mensajes enviados a él. El proceso registra, para cada canal, todos los mensajes que entraron después de que él registrara el estado y antes de que el emisor registrara su propio estado.

El algoritmo puede mediante el uso de mensajes de marcador especial, que son distintos de cualquiera de los otros mensajes que envían los procesos, y que los procesos podrán enviar y recibir mientras continúa su ejecución normal. El marcador tiene un papel dual: como un

aviso para que el receptor guarde su propio estado, sino lo ha hecho aun; y como un medio para determinar que mensajes incluir en el estado del canal.

El algoritmo se define mediante dos reglas, la regla de recepción del marcador y la regla del envío del marcador. La regla del envío del marcador obliga a los procesos a enviar un marcador después de haber registrado su estado, pero antes de que envíen cualquier otro mensaje.

La regla de recepción del marcador obliga a un proceso que no ha registrado su estado a hacerlo.

Cualquier proceso puede iniciar el algoritmo en cualquier instante. Actúa como si hubiera recibido un marcador (sobre un canal no existente) y sigue la regla de recepción del marcador. Por lo tanto registra su estado y comienza a registrar los mensajes que llegan sobre todos los canales entrantes.

Terminación del algoritmo de instantánea. Un proceso que ha recibido un mensaje marcador registra su estado en un tiempo finito y envía mensajes "marcador" sobre cada canal saliente en un tiempo finito. Si hay un recorrido de canales de comunicación y procesos desde un proceso hasta otro distinto del primer proceso, entonces registrara su estado un tiempo finito después que el primero haya registrado el suyo. Estamos suponiendo que el grafo de procesos y canales esta conectado fuertemente, se deduce que todos los procesos habrán registrado sus estados y los de los canales entrantes un tiempo finito después que algunos procesos hayan registrado inicialmente su estado.

Características del estado observado. El algoritmo de instantánea selecciona un corte de la historia de la ejecución. El corte, y por tanto el estado registrado por este algoritmo, es consistente.

Podríamos establecer una relación de alcanzabilidad entre el estado global observado y los estados inicial y final cuando se ejecuta el algoritmo. Sea Sis, una linealización del sistema tal y como se ejecuta. Sea S-inicio el estado global inmediatamente antes que el primer proceso registrara su estado; sea S-final el estado global cuando termina el algoritmo de **instantánea,** después de la última acción de registro de estado; y sea S-insi el estado global registrado.

Categorizados inicialmente todos los sucesos como sucesos **pre-instantánea o post-instantánea**. Un suceso pre-instantánea en el proceso es uno que sucedió antecede que registrara su estado; todos los demás sucesos son post-instantánea. Un suceso post-instantánea puede suceder antes que uno pre-instantánea, si los sucesos ocurren en diferentes procesos, ningún suceso post-instantánea puede suceder antes que otro pre-instantánea en el mismo proceso.

Para cada proceso el conjunto de sucesos que ocurrieron en él es exactamente el conjunto de sucesos que experimento antes de que registrara su estado.

Por tanto el estado de cada proceso en ese punto, y el estado de los canales de comunicación, es el estado global S-insi registrado por el algoritmo. No hemos desordenado

ninguno de los estados S-inicio o S-final con los que la linealización comienza y termina. Hemos establecido una relación de alcanzabilidad.

Estabilidad y alcanzabilidad del estado observado. La propiedad de alcanzabilidad del algoritmo de instantánea se utiliza para detectar predicados estables. En general, cualquier predicado no estable que nosotros establezcamos como *Verdadero* puede o no haber sido *Verdadero* en la ejecución actual cuyo estado global hemos registrado. Si un predicado estable es *Verdadero*, entonces podemos concluir que el predicado es *Verdadero* en el estado S-final, por definición un predicado estable que es *Verdadero* en un estado lo es también en cualquier otro estado alcanzable. Si el predicado se evalúa a *Falso*, entonces debe ser también *Falso* para S-inicio

Fuente: http://exa.unne.edu.ar/depar/areas/informatica/SistemasOperativos/MonogSO/TIEMEG02.htm

• 4.4: Depuración distribuida.

Se presenta ahora, el problema del registro de un estado global de un sistema de manera de poder hacer afirmaciones útiles, sobre si un estado transitorio, como opuesto a uno estable, ocurrió en la actual ejecución. Esto es lo que precisamos cuando depuramos un sistema distribuido. Un ejemplo es un sistema distribuido controlando un sistema de tuberías en una fábrica donde estamos interesados en si todas las válvulas (controladas por diferentes procesos) fueron abiertas al mismo tiempo. En este ejemplo no se pueden observar los valores de las variables o los estados de las válvulas simultáneamente. El reto es monitorizar la ejecución del sistema a lo largo del tiempo, para capturar información de la *traza* mas que una simple instantánea, por lo que podemos establecer a *posteriori* si la condición de seguridad requerida fue violada o pudo haberlo sido.

El algoritmo de instantánea de Chandy y Lamport recoge el estado de una forma distribuida, el algoritmo que se describirá (debido a Marzullo y Neiger [1991]) es centralizado. Los procesos observados envían sus estados a un proceso llamado monitor, que ensambla estados globalmente consistentes de los que recibe. Consideramos que el monito se encuentra fuera del sistema observando su ejecución.

El objetivo es determinar casos en los que dado un predicado de estado global f era sin duda alguna *verdadero* en algún punto de la ejecución y los casos en los que posiblemente era *verdadero*. La noción *posiblemente* surge como un proceso natural porque podemos extraer un estado global consistente *S* de un sistema en ejecución y encontrar que f(*S)* es *verdadero*. Una observación única de un estado global consistente no permite concluir si un predicado no estable será evaluado siempre a *verdadero* en la ejecución actual.

La noción *sin duda alguna* se aplica a la ejecución actual y no a una ejecución que hayamos extrapolado de ella. Puede parecer paradójico considerar que sucedió en la ejecución actual. Sin embargo, es posible evaluar si f fue sin duda *verdadero* considerando todas la linealizaciones de los sucesos observados. Se definen ahora las siguientes nociones para un predicado f en función de la linealizaciones de *H*, la historia de la ejecución del sistema.

Posiblemente f : esta afirmación significa que hay un estado consistente *S* a través del cual pasa una linealización *H* tal que *f(S) es verdadero.*

Sin duda alguna f: esta afirmación significa que para todas las linealizaciones *L* de *H*, hay un estado consistente *S* a través del cual pasa *L* tal que *f(S) es verdadero.*

Recolectado en estado. Los procesos observados *pi (i = 1, 2,..., N)* envían su estado inicial al proceso monitor inicialmente, y después de tiempo en tiempo, en *mensajes de estado*. El proceso monitor registra los mensajes de estado de los procesos *pi* en una cola separada *Qi* , para cada *i* = 1,2,....N.

La actividad de preparar y enviar mensajes de estado puede retrazar la ejecución normal de los procesos, pero aparte de eso no interfiere con ellos. No hay necesidad de enviar el estado inicialmente o cuando cambia. Hay dos optimizaciones para reducir el tráfico de mensajes de estado hacia el monitor. Primero, el predicado de estado global puede depender solo de ciertas partes de los estados de procesos. Segundo, solo necesitan enviar el estado en los instantes en que el predicado f puede llegar a ser *verdadero* o dejar de serlo. No tiene sentido enviar los cambios de estado que no afectan al valor del predicado

Fuente: http://exa.unne.edu.ar/depar/areas/informatica/SistemasOperativos/MonogSO/TIEMEG02.htm