Capítulo 5. Sincronización

Objetivo: Que el alumno analice y comprenda los diferentes algoritmos relacionados con sincronización, exclusión mutua, elección y consenso distribuidos, así como la importancia de estos en el diseño de sistemas distribuidos.

5.1 Introducción

La sincronización en sistemas distribuidos es más complicada que en un sistema centralizado, ya que se debe de considerar algunos de los siguientes puntos:

- Que la información se distribuye en varias máquinas.
- Los procesos toman decisiones con base en información local.
- Se debe evitar un punto único de falla.
- No existe un reloj común, como tampoco otra fuente de tiempo global.

5.2 Sincronización de relojes

Para la sincronización de relojes existen las siguientes alternativas:

Relojes lógicos

- Según Lamport, la sincronización de relojes no debe ser absoluta, debido a que si dos procesos no interactúan entre sí, no requieren que sus relojes estén sincronizados.
- La distorsión de reloj es la diferencia entre los valores de tiempo de los diferentes relojes locales.
- Aquí importa el orden de ocurrencia de los eventos, no la hora exacta.

Relojes físicos

- Usan el tiempo atómico internacional (TAI) y el tiempo coordenado universal (UTC).

- Se pueden sincronizar por medio de radios de onda corta.
- También se puede usar satélite para sincronizar.

5.3 ALGORITMOS PARA LA SINCRONIZACIÓN DE RELOJES

5.3.1 El Algoritmo de Lamport

Lamport define una relación temporal llamada: *ocurre antes de*. Por ejemplo, (a → b) se interpreta como "a ocurre antes de b". Para ilustrar esto, consideremos la siguiente situación:

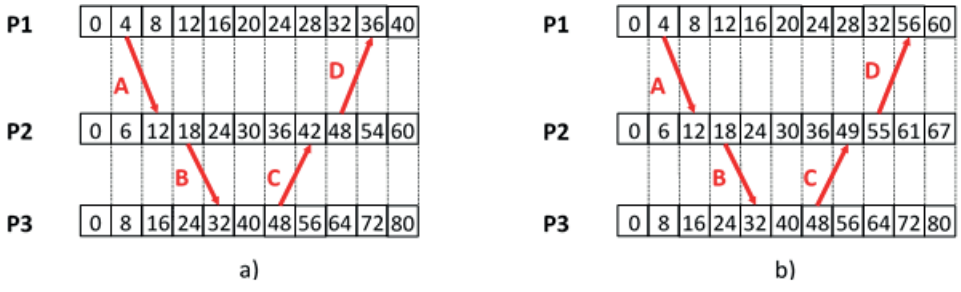
1. Si *a* y *b* son eventos del mismo proceso y *a* ocurre antes que *b*, entonces $a \rightarrow b$ es verdadero.
2. Si *a* es un envío de un mensaje por un proceso y *b* es la recepción del mensaje por otro proceso, entonces $a \rightarrow b$ es verdadero.
3. Eventos concurrentes: si $a \rightarrow b$ no es verdadero ni $b \rightarrow a$ tampoco.

Se requieren valores de tiempo para medir el tiempo:

- Si $a \rightarrow b$, entonces $C(a) < C(b)$.
- C siempre es creciente (se pueden hacer correcciones hacia adelante pero no hacia atrás).
- Para todos los eventos *a* y *b*, tenemos que $C(a) \neq C(b)$.

Un ejemplo de tres procesos (P1, P2 y P3), donde cada uno tiene su propio reloj a diferentes velocidades se muestra en la figura 5.1a, mientras que en la figura 5.1b se muestra la corrección de los relojes usando el algoritmo de Lamport.

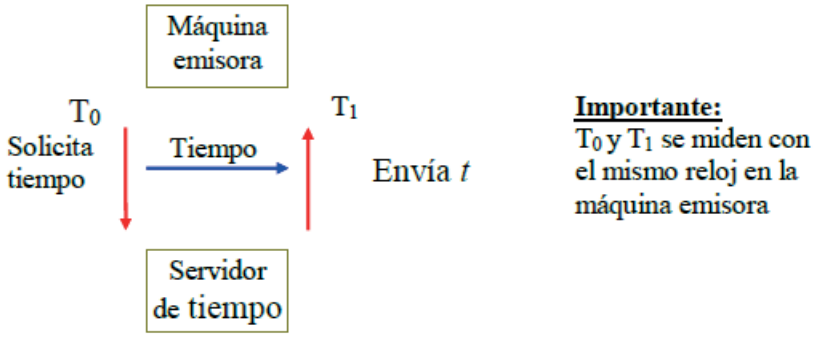
Figura 5.1. Un ejemplo del algoritmo de Lamport en tres procesos



5.3.2 El algoritmo de Christian

En este algoritmo un servidor central provee el tiempo por petición del usuario, este servidor es conocido como servidor del tiempo (ver figura 5.2).

Figura 5.2. Actualizando la hora a través de un servidor de tiempo



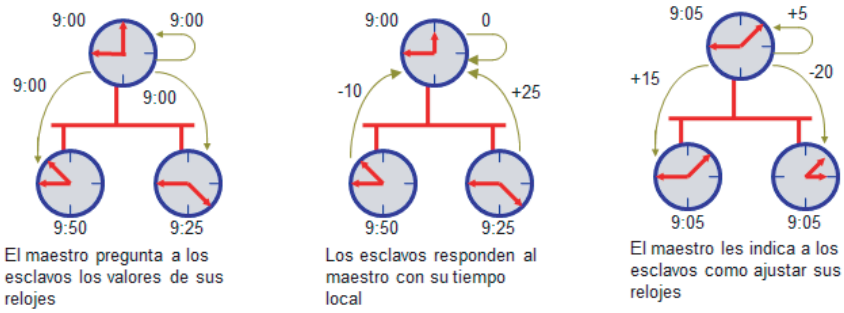
La operación de este algoritmo se resume a continuación:

- Periódicamente cada máquina envía un mensaje para solicitar el tiempo actual a este servidor.
- El servidor de tiempos responde con un mensaje que contiene el tiempo actual *t*.
- La máquina emisora puede poner su tiempo en $t + T_{trans}$.
- Un problema grave es que, como el tiempo nunca corre hacia atrás, el valor actual del tiempo del emisor podría ser mayor al valor de $t+T_{trans}$, lo que perjudicaría a los archivos compilados.
- Otro problema es el tiempo de propagación (T_{trans}), el cuál es distinto de 0 y varía según la carga en la red.
- Una estimación del tiempo de propagación sería realizado por el reloj de la máquina emisora calculando $(T_1 - T_0)/2$ e incrementándolo al llegar el valor *t*.
- Si se conoce el tiempo que tarda el servidor en manejar la interrupción (t_l), la estimación de tiempo puede mejorar con base en la siguiente expresión: $(T_1 - T_0 - t_l) / 2$.
- Christian sugiere hacer varias mediciones para mejorar la precisión y descartar los valores límites de (T_1-T_0), ya que estos están en función de la operación de la red.

5.3.3 El algoritmo de Berkeley

En este algoritmo una computadora maestra pide periódicamente a las computadoras esclavas sus relojes y efectúa la sincronización (ver figura 5.3). Se calcula un tiempo promedio, el cual es tomado como base si este no se diferencia más de un valor base. La computadora maestra envía los desfases de tiempo a los esclavos para que avancen su reloj o disminuyan la velocidad del mismo hasta lograr una reducción específica.

Figura 5.3. Representación del algoritmo de Berkeley [Tanenbaum & Van Steen, 2008]



5.4 EXCLUSIÓN MUTUA

En los sistemas distribuidos existen situaciones en que hay recursos compartidos los cuales no pueden ser utilizados por más de un proceso al mismo tiempo. En sistemas que comparten memoria se pueden utilizar semáforos o monitores para garantizar el uso de la región crítica de manera exclusiva. Sin embargo, en los sistemas distribuidos, los procesos ya no comparten la memoria física, por ello se deben de idear otros algoritmos que garanticen la exclusión mutua. Los requisitos básicos que debe cumplir un algoritmo de exclusión mutua son:

- Inanición.
- Seguridad.
- Orden.

La inanición cuida que un proceso que desea entrar en una región crítica pueda entrar en algún tiempo, lo cual implica que no se deben producir interbloqueos ni inanición. La seguridad se refiere a que la exclusión mutua debe de garantizar que en la región crítica, cuando mucho, solo se ejecute

un proceso en un momento dado. El orden implica que el acceso a la región crítica debe de realizarse siguiendo el orden casual propuesto por Lamport "sucedió antes". Para garantizar que ningún proceso entre en una región crítica mientras esta sea ocupada por otro proceso en sistema distribuido, se usan los siguientes algoritmos:

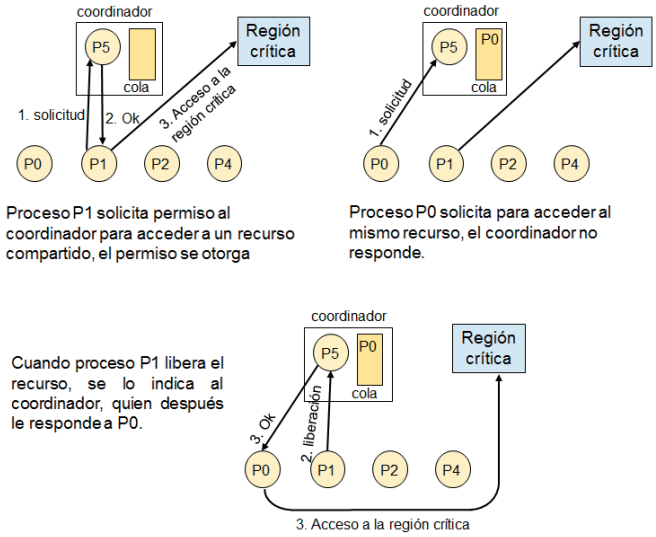
- Algoritmo centralizado.
- Algoritmo distribuido (Ricart y Agrawala).
- Algoritmo de anillo de token.

5.4.1 Algoritmo de servidor centralizado

Este algoritmo simula un solo procesador para realizar la exclusión. El algoritmo nombra a un proceso coordinador. En este algoritmo el coordinador da acceso a la región crítica al que posea un token (lo devuelve al coordinador). Cuando un proceso sale de la región crítica libera el token. El coordinador permite la entrada enviando el token o no (para lo cual no envía algo o puede enviar un permiso denegado).

Los problemas de este algoritmo se presentan cuando el coordinador falla o cuando falla el poseedor del token. Un solo coordinador en un gran sistema puede ocasionar un cuello de botella. La figura 5.4 muestra un ejemplo de cómo opera este algoritmo.

Figura 5.4. Algoritmo de servidor centralizado [Tanenbaum & Van Steen, 2008]

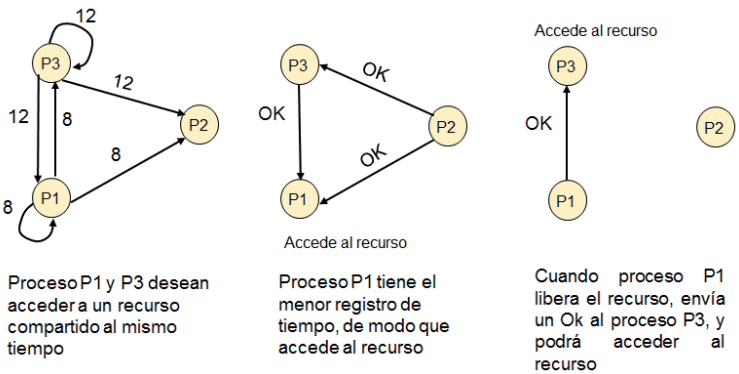


5.4.2 Algoritmo de Ricart y Agrawala

Este algoritmo está basado en un consenso distribuido y requiere de un orden para todos los eventos en el sistema [Tanenbaum & Van Steen, 2008]. Cuando un proceso desea entrar a la región crítica, construye un mensaje formado por su nombre, su número de proceso y la hora actual. Después, este mensaje es mandado a todos los procesos, incluyéndose el mismo. Así, se considera que la comunicación es confiable. La figura 5.5 muestra un escenario de este algoritmo.

- Cuando un proceso recibe el mensaje se pueden presentar tres alternativas:
  a) Si el proceso receptor no está en la región crítica y tampoco desea entrar, envía un mensaje de OK al proceso emisor.
  b) Si el receptor se encuentra en la región crítica, no responderá y solo colocará la solicitud en una cola.
  c) Si el receptor no está en la región crítica pero desea entrar, compara la etiqueta de tiempo del mensaje recibido con la suya, la menor gana. Si el mensaje recibido es menor al suyo, ejecuta el paso a), pero si es mayor al suyo ejecuta el paso b).
- El proceso solicitante espera el permiso de los otros, entonces entra a la región crítica (recurso compartido), si no ocurre así se bloquea.
- Cuando sale de la región crítica envía un OK y saca a todos los procesos en su cola.
- Los problemas que presenta este algoritmo son:
  - Que requiere mucho más mensajes que el algoritmo de servidor centralizado.
  - Que no existe la tolerancia a fallas para ningún proceso.

Figura 5.5. Ejemplo de Algoritmo de Ricart y Agrawala



5.4.3 Algoritmo de anillo de token

Este algoritmo distribuido fue propuesto por Lann [1977] para alcanzar exclusión mutua en un sistema distribuido. En este algoritmo los procesos pueden estar desordenados (ver figura 5.6) y opera como sigue:

Premisas:

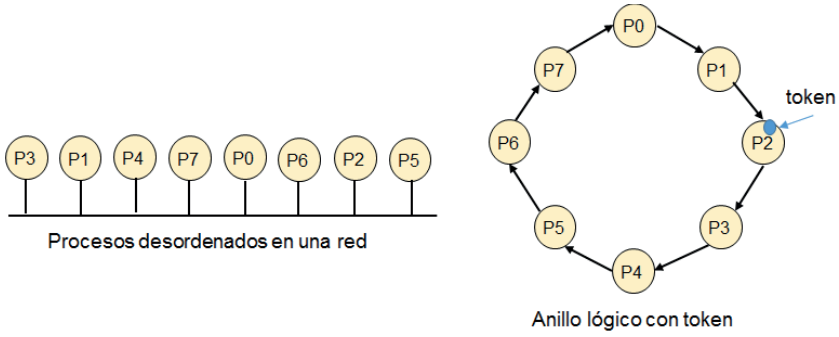
- Todos los equipos conectados forman un anillo lógico.
- Cada proceso es un nodo del anillo y se le asigna una posición en el anillo.
- Cada nodo del anillo debe de saber cuál es la dirección de su vecino.

Operación:

- Al arrancar el sistema, al proceso de posición 1 se le da un token, el cual irá circulando por el anillo.
- Cuando el proceso *k* tenga el token, debe transferirlo mediante un mensaje al proceso *k+1*.
- Así, el token irá pasando por todos los nodos del anillo.
- Cuando un proceso recibe el token, si quiere entrar a la región crítica, retiene el token y entra en la región crítica.
- Cuando el proceso sale de la región crítica le pasa el token al siguiente nodo del anillo.

Este algoritmo no respeta el orden y puede requerir de 1 mensaje (en el mejor de los casos) a *n-1* mensajes (en el peor de los casos) para conseguir entrar a la región crítica. En caso de perderse un mensaje con el token sería difícil detectarlo, porque se puede suponer que algún proceso está usando el token. Si falla cualquier proceso del anillo estando en la región crítica también es un problema en este algoritmo, ya que es difícil de detectarlo.

Figura 5.6. Algoritmo de anillo de token [Tanenbaum & Van Steen, 2008]



5.5 ALGORITMOS DE ELECCIÓN

Una elección es un procedimiento que se lleva a cabo por los procesos de cierto grupo para escoger a alguno para una tarea específica (ejemplo: coordinación). El objetivo del algoritmo de elección es asegurar que cuando se comienza una tarea, se concluya que todos los procesos están de acuerdo en quién es el nuevo coordinador. Un algoritmo de elección para realizar este procedimiento emplea tres tipos de mensaje:

- Anunciación del evento de elección.
- Votación.
- Anunciación del resultado de la elección

5.5.1 El algoritmo del Grandulón

El algoritmo del "Grandulón" es un algoritmo para elegir un proceso coordinador y fue propuesto por García Molina, en 1982 [Coulouris *et al.*, 2012], [Tanenbaum, 1996]. En general, el método a seguir es el siguiente:

Premisas:
- El sistema es síncrono y utiliza tiempo de espera para la identificación de fallas en los procesos.
- Se permite que los procesos se bloqueen durante la ejecución del algoritmo.
- La entrega de mensajes entre procesos se supone fiable y dentro de un periodo máximo.
- Los procesos están ordenados, tienen un único identificador (IDs) conocido y se sabe cuántos procesos existen.

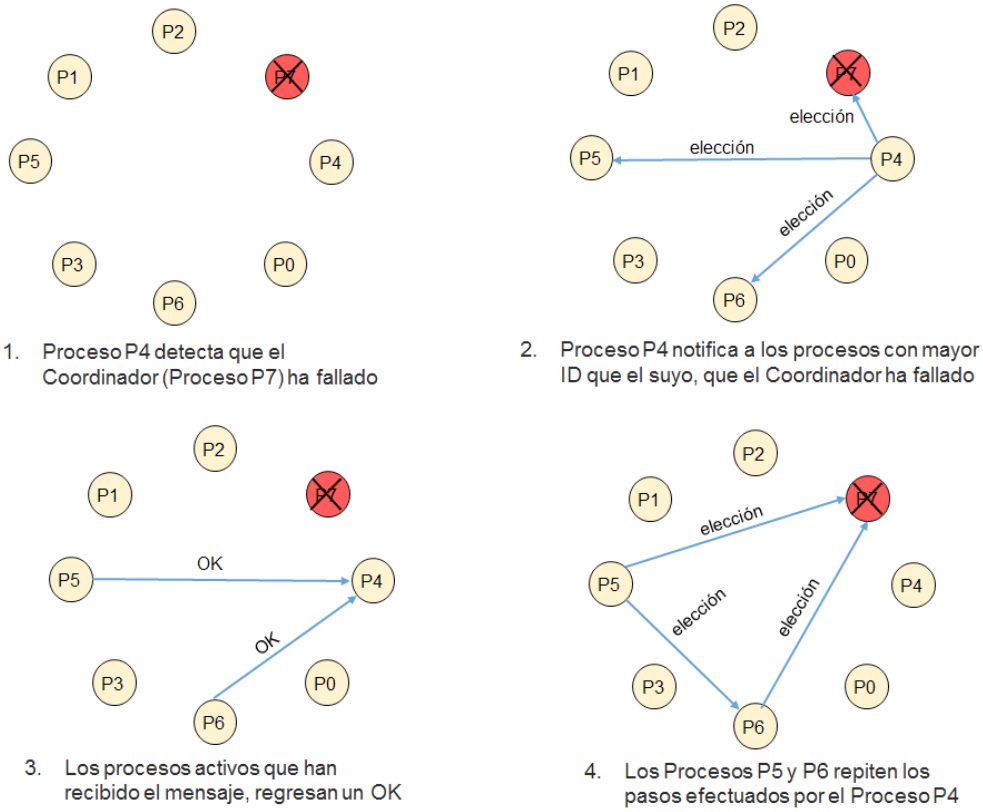
Tipo de mensajes:
- Mensaje de Elección: comunica que se seleccionará un nuevo coordinador.
- Mensaje de Respuesta: es una respuesta al mensaje de elección.
- Mensaje de Coordinador: comunica el ID del proceso seleccionado como coordinador.

Operación:
1. Un proceso x manda un mensaje de elección a todos aquellos procesos que tengan un identificador más grande cuando detecta que el coordinador ha fallado.

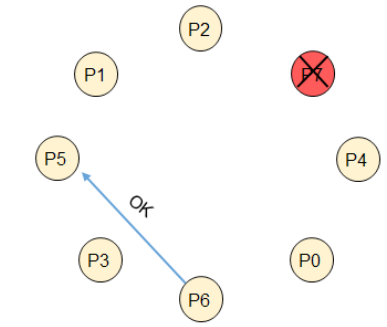
2. El proceso x espera los votos y, si ningún voto (ok) llega después de un cierto tiempo, el proceso x se declara coordinador, y envía el mensaje de coordinador a los procesos con identificador más pequeño que el suyo.
3. Si un voto llega (aunque pueden llegar varios votos), puede ser que otro coordinador sea declarado ganador.
4. Si un proceso recibe un mensaje de elección, envía un voto y otra elección empieza (paso 4 de la figura 5.7).
5. Cuando un coordinador que había fallado regresa, empieza una elección y puede volver a readquirir el control, aunque exista un coordinador actual.

Un ejemplo de cómo trabaja el algoritmo del "Grandulón" se muestra en la figura 5.5.

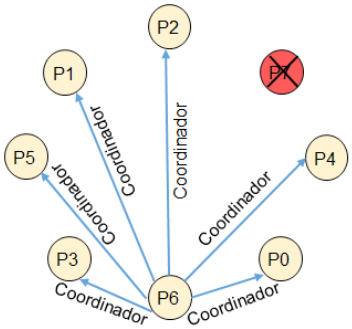
Figura 5.7. Ejemplo del algoritmo del "Grandulón" para elegir a un nuevo coordinador



1. Proceso P4 detecta que el Coordinador (Proceso P7) ha fallado
2. Proceso P4 notifica a los procesos con mayor ID que el suyo, que el Coordinador ha fallado
3. Los procesos activos que han recibido el mensaje, regresan un OK
4. Los Procesos P5 y P6 repiten los pasos efectuados por el Proceso P4



5. El Proceso P6 envía OK al proceso P5 y detecta que es el proceso con mayor ID



6. Proceso P6 tiene el mayor ID, se proclama Coordinador y lo comunica a los otros

5.5.2 Algoritmo de anillo

Otros algoritmos de elección se basan en topologías de anillo, ya sea lógico o físico. Existen varios algoritmos de elección basados en anillo. Sin embargo, aquí se describe el algoritmo de anillo de Chang & Roberts [1974] basado en el principio de extinción selectiva. Este algoritmo se usa cuando:

- Los procesos están física o lógicamente ordenados en anillo.
- No se conoce el número total de procesos (n).
- Cada proceso se comunica con su vecino (izquierda o derecha).

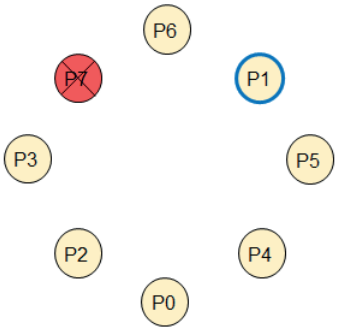
Operación (la figura 5.8 ilustra la operación de este algoritmo):

1. Inicialmente todos los procesos son "no participantes".
2. Cualquier proceso *P* decide arrancar una elección en cualquier momento.
3. Este proceso *P* se pone en estado "participante" y envía un mensaje de elección *M* a su vecino.
4. El mensaje *M* enviado contiene el *ID* (identificador) del proceso que ha iniciado la elección.
5. Cuando el vecino recibe el mensaje de elección *M*, establece su estado como participante y comprueba el *ID* del mensaje.
6. Si es mayor que su propio *ID*, entonces se lo envía directamente a su vecino.
7. Si su *ID* es mayor al *ID* recibido, entonces lo coloca en el mensaje *M* y lo envía a su vecino.
8. Así se circula el mensaje *M* sucesivamente hasta que llega a un proceso P_n que comprueba que el *ID* recibido es el propio. Eso indica que solo ha sobrevivido el mayor *ID*, que es la del proceso P_n.

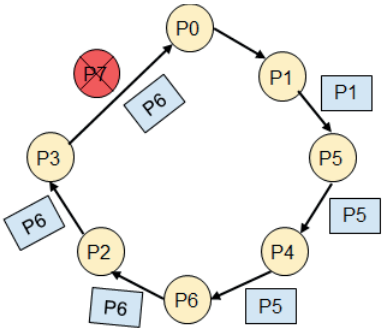
9. Entonces, este proceso P_n es el coordinador y lo notifica a su vecino. Cuando un proceso recibe un mensaje de coordinador debe de poner su estado como "no participante" y enviar el mensaje a su vecino.
10. Cuando el mensaje de coordinador retorna al proceso que lo emitió (el coordinador), entonces todos los procesos saben quién es el coordinador y todos quedan en estado "no participantes".

En caso de que dos procesos inicien al mismo tiempo una elección y se envíen mensajes de elección, un proceso en estado de "participante" debe de verificar el ID del proceso que envía el mensaje de elección. Si es menor al propio, el mensaje se descarta. Así, todos los mensajes de elección se extinguirán, excepto el que lleva el ID más alto.

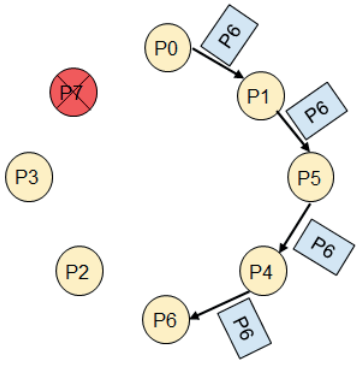
Figura 5.8. Algoritmo de elección que utiliza un anillo



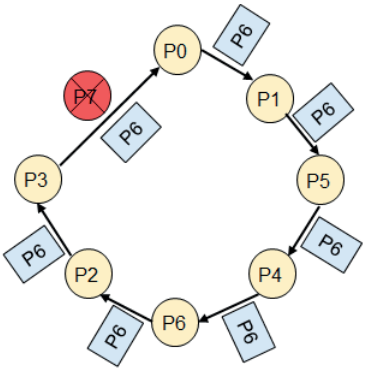
Todos los nodos en estado "no participante". Nodo P7 falla y lo detecta P1



Nodo 2 envía un "mensaje de elección". "Participantes": P1, P5, P4, P6, P2, P3 y P0.



La circulación del mensaje se detiene al llegar al nodo P6



P6 difunde "mensaje-coordinador. Todos los nodos en estado "no participante"

5.6 ALGORITMOS DE CONSENSO

La gran mayoría de los investigadores del área considera que el problema fundamental del cómputo distribuido es el consenso distribuido. Es decir, cómo lograr que un conjunto de procesos, ejecutándose en distintos nodos, se pongan de acuerdo con respecto al valor de un dato. Para que cualquier protocolo de coordinación funcione correctamente, se basan en que todos los procesadores correctos se pongan de acuerdo con respecto al valor del dato que intercambian. El problema del consenso consiste en conseguir que, aun en presencia de procesadores erróneos, los procesadores correctos sean capaces de ponerse de acuerdo con respecto a un valor, sin importar que este valor no sea el óptimo.

5.6.1 Problema de los generales bizantinos

El *problema de los generales bizantinos* fue planteado originalmente por Lamport, Shostak y Peace [1982]. Este problema plantea que un grupo de generales sitia una ciudad y deben de ponerse de acuerdo a través de un plan de ataque precisamente para atacar o retirarse, independientemente de que existan generales traidores. Los generales solo se comunican a través de mensajes a los otros generales. Uno de ellos, el general comandante, da las órdenes. Los otros, tenientes generales, deben de decidir si atacar o retirarse. Sin embargo, uno o más de los generales puede ser un traidor o pueden fallar. Esta traición puede verse de dos formas:

- Los mensajes pueden no llegar o, dicho de otra manera, las comunicaciones no son confiables.
- Un general traidor puede mentir, es decir, un nodo puede fallar de manera impredecible.

A continuación se revisa las circunstancias bajo las que se puede lograr un acuerdo entre los generales leales y cuando este acuerdo es imposible.

Caso 1: Tres generales

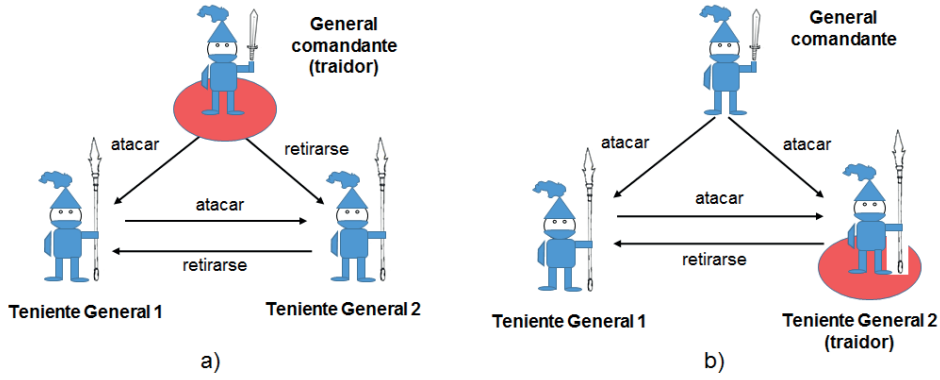
Este caso ilustra el escenario para cuando hay un general comandante y dos tenientes generales, y uno de ellos es traidor. ¿Pueden los generales leales llegar a un consenso?, es decir, ¿acordar "atacar" o "retirarse"?

Asume que el general comandante es el traidor (como se ilustra en la figura 5.9a). Entonces, el general comandante indicará al teniente general 1 "atacar" y al teniente general 2 "retirarse". Ambos tenientes tratarán de verificar la orden recibida comunicándose entre ellos. El teniente general 1 indicará al teniente general 2 que recibió la orden de "atacar", mientras que el teniente general 2 indicará al teniente general 1 que recibió la orden de "retirarse". Ambos tenientes generales podrán solo deducir que el general comandante es traidor pero no podrán tomar una decisión consensada.

Si se consideran los mismos participantes, ahora asume que el teniente general 2 es el traidor, el general comandante es leal, al igual que el teniente general 1. Este escenario se ilustra en la figura 5.9b, donde el general comandante da la orden de "atacar" a ambos tenientes generales, quienes intercambian la orden recibida. El teniente general 1 indica al teniente general 2 "atacar". Sin embargo, el teniente general 2 es traidor, por ello cambia la orden recibida y le indica al teniente general 1 "retirarse".

Para ambos escenarios, el teniente general escucha dos órdenes distintas, tanto del general comandante como del teniente general 2, sin saber cómo actuar, por lo que en ambos escenarios no existe consenso.

Figura.5.9. Caso de tres generales donde uno es traidor. a) El general comandante es el traidor, b) El teniente general es el traidor. No se llega a un consenso



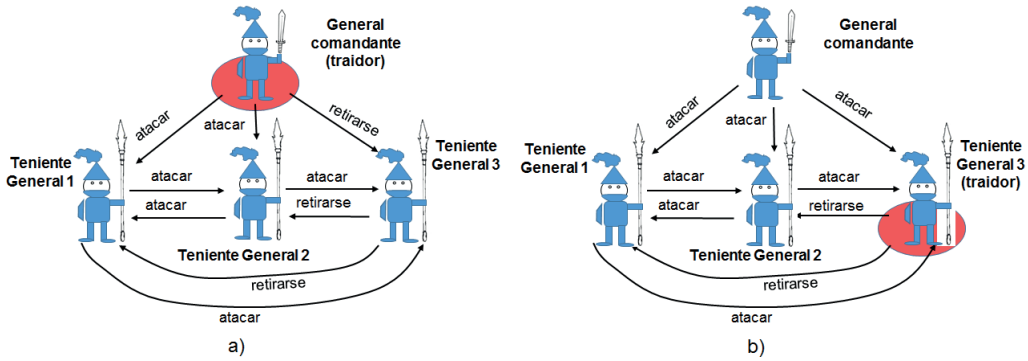
Caso 2: Cuatro generales

Ahora se muestra un caso similar al anterior pero con cuatro generales. Hay un general comandante y tres tenientes generales, y uno de ellos es traidor. ¿Pueden los generales leales llegar a un consenso?, es decir, ¿pueden acordar "atacar" o "retirarse"?

Asume que el general comandante es el traidor y los tres tenientes generales son leales. Independientemente de las órdenes que reciban los tenientes generales del general comandante, ellos las intercambian entre sí, de tal manera que los tres recibirán tres mensajes. Al decidir por la mayoría de las órdenes recibidas, los tres tenientes generales tomarán la misma decisión. En este caso, "atacar". Este escenario se muestra en la figura 5.10a.

Si se asume que el comandante general, y los tenientes generales 1 y 2 son leales, excepto el teniente general 3, quien es traidor, se tendría un escenario como el plateado en la figura 5.10b. El comandante general envía a todos los tenientes generales la orden de "atacar". Esta orden es retransmitida por los tenientes generales 1 y 2 a los demás, sin embargo, el teniente general, como es un traidor, cambia la orden recibida y envía a los tenientes generales 1 y 2 la orden de "retirarse". Esto no cambia la decisión consensada de los tenientes generales 1 y 2, ya que por mayoría deciden "atacar". También se puede notar que el general traidor pudo haber enviado cualquier mensaje sin que esto afectara el consenso del resultado.

Figura 5.10. Caso de cuatro generales donde uno es traidor.
a) El general comandante es el traidor, b) Un teniente general es el traidor.
En ambos casos, el consenso es "atacar"



El problema de los generales bizantinos es frecuentemente referido en la tolerancia a fallas de los sistemas distribuidos, también se conoce como problema de acuerdo bizantino. Lamport *et al.* [1982] demostraron que en un sistema con *k* procesos defectuosos, el consenso se puede lograr solo si están presentes *2k+1* procesos que funcionen correctamente, para un total de *3k+1* procesos.

Ejercicios

1. ¿Por qué es conveniente el uso de relojes lógicos en lugar de los relojes físicos?

2. Cita la diferencia entre el algoritmo de Christian y el algoritmo de Berkeley.

3. Dado tres procesos P1, P2 y P3, representa la planificación de los procesos siguiendo el algoritmo de "ocurre antes de".

4. ¿Cuál es la complejidad del algoritmo de anillo de Chang y Roberts?

5. ¿Cuáles son las desventajas del algoritmo de Ricart y Agrawala?

6. ¿Cuál es la principal limitante del algoritmo centralizado? Sugiere una variante.

7. En el algoritmo del Grandulón, ¿qué pasa si dos procesos detectan la caída del coordinador de forma simultánea y ambos deciden hacer una elección?

8. ¿Cómo se podría tolerar la falla del coordinador en un sistema centralizado?

9. ¿Cómo son afectadas las memorias cachés cuando se sincronizan los relojes internos de los procesadores de un sistema distribuido?

10. En el algoritmo de Ricart y Agrawala, ¿cómo se puede interpretar cuando un proceso falla y no responde a la solicitud de otro proceso para entrar a una región crítica?

Actividad integradora

1. De la aplicación de los contenidos, así como de las habilidades y actitudes desarrolladas.
   Se recomienda exposiciones por parte del profesor, así como una explicación detallada de cada algoritmo por medio de talleres prácticos donde los estudiantes demuestren diferentes casos y posibles escenarios de estos algoritmos. Se recomienda encargar como trabajo extraclase y en equipo el

desarrollar demos de algunos de estos algoritmos, además de exponer en clase para comprobar la apropiación del conocimiento de este capítulo. Debido a que este tema es fundamental en un curso de sistemas distribuidos, se recomienda desarrollar el contenido en un promedio de seis sesiones que incluyan trabajo intenso de ejemplos demostrativos de la manera en que operan estos algoritmos, así como para explicar su complejidad.

El alumno debe de mostrar un gran interés por abstraer y modelar problemas que pueden resultar cotidianos pero que tienen gran aplicabilidad en los sistemas de cómputo distribuido, también de demostrar habilidades para programar algún algoritmo de sincronización en algún lenguaje de programación de alto nivel. Se recomienda Java.

2. Del logro de los objetivos establecidos en el capítulo o apartado del material.

   Se espera alcanzar los objetivos de este capítulo con el apoyo en la solución de los ejercicios, así como en la programación de diferentes algoritmos de sincronización que simulen su operación.

Capítulo 6. Transacciones distribuidas

Objetivo: Que el alumno analice y comprenda las particularidades de las transacciones en un entorno distribuido, así como sus modelos y aplicaciones.

6.1 Introducción

En general, en los sistemas distribuidos se realizan dos acciones que se pueden considerar como contrastantes. La primera acción consiste en que los clientes no deben intervenir en las acciones que realiza otro cliente, mientras que la segunda acción consiste en que el servidor debe ser usado por los clientes para compartir e intercambiar información entre ellos.

Para el manejo de este tipo de situaciones en los datos compartidos existe una herramienta conceptual muy útil conocida como "transacción", la cual es una unidad indivisible de manipulación de información. Las transacciones en los sistemas distribuidos reciben el nombre de transacciones atómicas, siendo estas un mecanismo de alto nivel que oculta los aspectos técnicos de sincronización, como la exclusión mutua, interbloqueos y recuperación de fallas. Algunas características de las transacciones son:

- Que agrupa operaciones en una transacción en las que todas terminan o ninguna es realizada, bajo esta última situación se regresa al estado inicial.
- Que para implementarlas, el servidor debe de encapsular (aislar) los recursos que maneja y administra.
- Que las transacciones intentan eliminar algunos problemas debidos a las concurrencias. Por ejemplo, si dos usuarios ejecuten al mismo tiempo Rename ("x","y") y Rename ("x", "z") en un servicio de directorio.

6.2 MODELO DE TRANSACCIONES

Por medio de este se puede modelar con mayor precisión las transacciones a través del uso de procesos independientes. Entre los puntos a considerar están:

- *Almacenamiento estable*: Permite que la información perdure a todo, con excepción de las catástrofes, para lo cual se realiza un disco espejo.
- *Primitivas de transacción*: Son proporcionadas por el sistema operativo o compilador, como *begin - transaction, end - transaction, abort- transaction, read, write*.
- *Propiedades de las transacciones*: Son las propiedades fundamentales de las transacciones y se indican como ACID, si se cumple con lo siguiente:
  - *Atomicidad*: Las transacciones se ejecutan completamente o se abortan.
  - *Consistencia*: El resultado de una transacción no debe de depender del número de clientes concurrentes.
  - *Aislamiento*: Los efectos intermedios de las transacciones no deben de ser visibles a otras.
  - *Durabilidad*: La información debe de guardarse en disco una vez que la transacción se ejecuta exitosamente.
- Si una transacción se aborta, es necesario un procedimiento de recuperación para eliminar las operaciones ya realizadas.

6.3 PROBLEMAS DEBIDO A LA CONCURRENCIA DE TRANSACCIONES

Entre los principales objetivos del servidor de transacciones se encuentran maximizar la concurrencia y usar la serie equivalente. Las operaciones atómicas tienen que ser capaces de bloquear el acceso de varios usuarios a una región crítica. El problema de modificaciones y pérdidas sucede cuando una transacción aborta. El problema de reportes inconsistentes ocurre cuando una transacción lee un dato antes que otra transacción actualice. Una solución sería usar la serie equivalente de transacciones por las entrelazadas de instrucciones. La equivalencia en serie es usada como criterio para la derivación de diferentes mecanismos de control de concurrencia. El servicio de transacciones asegura que el resultado será el mismo si se realizan las transacciones de manera secuencial.

6.4 RECUPERACIÓN DE TRANSACCIONES

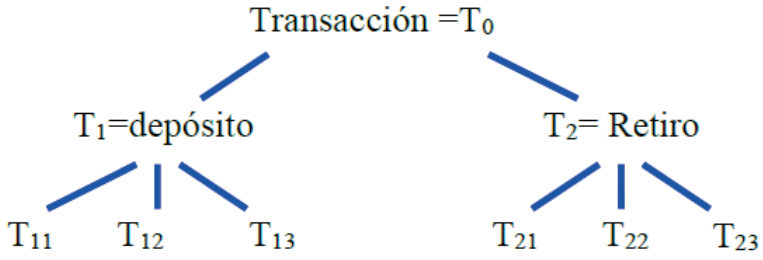
Un servicio de transacciones tiene que ser capaz de abortar transacciones y recuperarse de los efectos ocasionados por ello. Entre algunos de los problemas relacionados al aborto de las transacciones, se pueden citar los siguientes: lecturas sucias, cascada de aborto, y escrituras prematuras. El problema de "lecturas sucias" ocurre cuando una transacción exitosa lee datos que han sido afectados por una transacción abortada. Una solución a este problema es retrasar la decisión de declarar exitosa una transacción hasta que todas las transacciones relacionadas sean exitosas. Para el problema de "cascada de aborto" se propone como solución que se retrasen las lecturas hasta que las transacciones que se encuentran escribiendo al mismo dato, aborten o tengan éxito. El problema de las "escrituras prematuras" se presenta cuando dos transacciones escriben al mismo dato al mismo tiempo. Para esto se propone como solución retrasar las operaciones de escritura. Usando versiones tentativas normalmente se pueden evitar retrasos.

6.5 TRANSACCIONES ANIDADAS Y DISTRIBUIDAS

6.5.1 Transacciones anidadas

Una transacción anidada (TA) está formada por un conjunto de transacciones, cada una de las cuales puede estar a su vez formada por transacciones, y así sucesivamente, como se observa en la figura 6.1.

Figura 6.1. Ejemplo de transacción anidada



Las transacciones anidadas son útiles para:

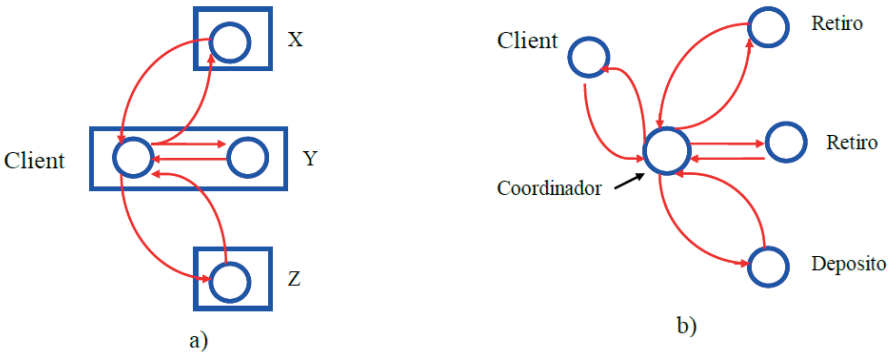
- Dar una jerarquía a las transacciones según su importancia.
- Especificar la concurrencia.
- Mejorar el control de las transacciones. Por ejemplo, una transacción anidada puede abortar sin que su padre tenga que hacerlo.
- Modelar transacciones distribuidas, las cuales son transacciones cuyas partes se realizan en diferentes sitios.

6.5.2 Transacciones distribuidas

En este tipo de transacciones, una transacción implica a muchos servidores (ver figura 6.2a), por lo que existe la posibilidad de datos replicados. Un coordinador (ver figura 6.2b) es el primer servidor que atiende las transacciones y es responsable de [Coulouris *et al.*, 2001]:

- Abortar o declarar exitosa la transacción.
- Utilizar nuevos servidores para la transacción.

Figura 6.2. a) Esquema de la transacción distribuida,
b) Coordinación de transacciones



6.6 IMPLANTACIÓN DE SISTEMAS DISTRIBUIDOS

Para la implantación de sistemas distribuidos se usan comúnmente dos métodos:

- *Espacio de trabajo individual:* Consiste en que cuando un proceso inicia su transacción se le asigna un espacio de trabajo individual, que contiene

los archivos que va a acceder. Uno de los inconvenientes de este método es el costo de copiar todo.
- *Bitácora de escritura anticipada*: Aquí los archivos se modifican, pero antes de hacerlo se deberá de escribir en una bitácora un registro indicando la transacción que realiza el cambio, el archivo y bloque modificado, así como los valores anteriores y nuevos.

6.7 TRANSACCIONES CON REPLICACIÓN

Existen tres modelos para atender peticiones en un sistema de transacciones con replicación:

- *Asíncrono:* Son modelos donde las peticiones son procesadas por servidores de réplicas locales.
- *Síncrono:* En este modelo las peticiones de modificación son procesadas en el mismo orden en todas las réplicas (orden total).
- *Mixto:* En este modelo ciertas réplicas pueden ser contactadas y procesadas por el mismo orden.

El modelo seleccionado depende de la razón entre el número promedio de lecturas y escrituras. El orden en que pueden ser procesadas dos peticiones r_1 y r_2 en los servidores de réplicas son:
- *Total*: Si r_1 es procesada antes que r_2 en todos los servidores de réplicas o viceversa.
- *Casual*: Si $r_1 \rightarrow r_2$, lo que implica que r_1 sea procesado antes que r_2 en todos los servidores de réplicas.

El ejemplo anterior sería síncrono si la ejecución de r_1 implica que todas las peticiones fueron realizadas antes en todos los servidores de réplicas.

Una petición no será procesada por un servidor de réplicas hasta que las restricciones de orden sean cumplidas. La petición es guardada en una cola. Entre las propiedades de la computadora solicitante se pueden mencionar las siguientes:

- *Seguridad*: Se refiere a que ningún mensaje puede ser procesado fuera de orden.

- *Vida limitada*: Ningún mensaje permanecerá esperando en la cola indefinidamente.

Ejemplos de arquitecturas de replicación son [Coulouris *et al.*, 2001]:

- *Gossip*: Esta arquitectura proporciona servicios altamente disponibles por medio del uso de réplicas.
- *Isis*: En esta arquitectura los grupos de procesos se clasifican de acuerdo con el patrón de comunicación seguido, que puede ser del mismo rango, de servidores, de cliente - servidor y de difusión.

6.8 PROBLEMA DEL "DEADLOCK" EN LOS SISTEMAS DISTRIBUIDOS

El problema del deadlock o candado mortal en los sistemas distribuidos es similar al que se presenta en los sistemas de un solo procesador, pero con mayor dificultad, ya que evitarlos, prevenirlos, detectarlos y remediarlos se complica aún más al tener la información dispersa en varias computadoras.

Para afrontar el problema del deadlock en los sistemas distribuidos, se consideran las siguientes estrategias [Coulouris *et al.*, 2001]:

1. *Ignorar el problema*.
2. *Detección:* Esta estrategia permite que el deadlock se presente, lo detecta e intenta recuperarse de ellos. Dos son los tipos de algoritmos a resaltar:
   - Algoritmos centralizados
   - Algoritmos distribuidos
3. *Prevenir*: Esta estrategia no permite que el deadlock se presente estructuralmente, para lo cual asume transacciones atómicas y tiempo real.
4. *Evitarlos*: Se realiza una asignación cuidadosa de recursos pero casi nunca se utiliza.

6.9 SERVICIOS WEB

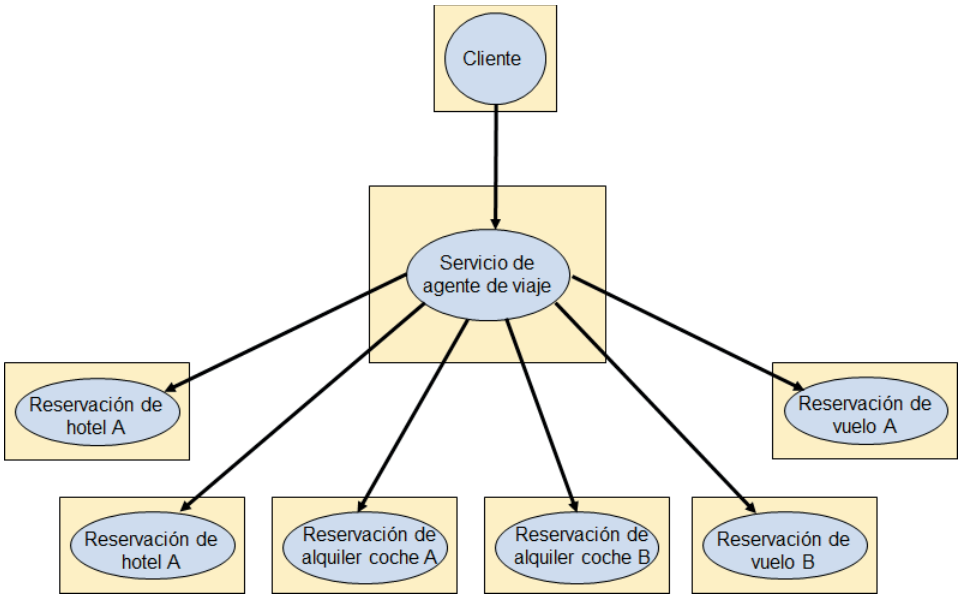
Un servicio web ofrece una interfaz de servicio que permite a los clientes interactuar con los servidores de una manera más general que como los navegadores lo hacen. Los clientes acceden a las operaciones en la interfaz de un servicio web a través de solicitudes y respuestas con formato en XML

(eXtensible Markup Language) y generalmente se transmite a través protocolos HTTP (Hypertext Transfer Protocol) [Coulouris *et al.*, 2012]. La interfaz de servicio web consiste generalmente en una colección de operaciones que puede ser utilizada por un cliente a través de Internet. Las operaciones en un servicio web pueden ser proporcionadas por una variedad de recursos, por ejemplo, los programas, los objetos o las bases de datos. Un servicio web puede ser manejado por un servidor web junto con las páginas web o ser un servicio independiente. La representación de datos externos y de clasificación de mensajes intercambiados entre los clientes y los servicios web se hace en XML.

Los servicios web proporcionan una infraestructura para mantener de una forma más rica y estructurada la interoperabilidad entre clientes y servidores. Asimismo, los servicios web proporcionan una base por la que un programa cliente en una organización puede interactuar con un servidor en otra organización sin supervisión humana. En particular, los servicios web permiten que aplicaciones complejas a ser desarrolladas por prestación de servicios integren otros servicios. Es decir, los servicios web pueden actuar como componentes independientes que se integrar entre sí para formar sistemas distribuidos complejos.

Un escenario típico de un servicio web es la integración de un conjunto de aplicaciones de distintas empresas u organizaciones. Un ejemplo es un agente de reservación de viajes. Para viajar, muchas personas realizan la reservación de vuelos, hoteles y alquiler de coches en línea por medio de diferentes sitios web. Si cada uno de estos sitios web es para proporcionar una interfaz de servicios web estándar, entonces un "servicio de agente de viajes" podría utilizar sus operaciones para proporcionar a un cliente viajero una combinación de estos servicios. Este escenario se ilustra en la figura 6.3.

Figura 6.3. Ejemplo de uso de servicios web [Coulouris *et al.*, 2012]



Es posible que un servicio de agente de viajes use patrones de comunicación alternativos disponibles en el servicio web. En este caso, el primer patrón sería para el soporte a documentos de consulta con el fin de agilizar la reserva. Esto se logra con el intercambio asíncrono de diferentes documentos basados en fechas y destinos. Por ejemplo, diferentes hoteles en determina ciudad, diferentes líneas aéreas a un destino o diferentes compañías para rentar autos. Otro patrón de comunicación podría ser para el control de los datos de la tarjeta de crédito, en este caso las interacciones con el cliente deben de ser apoyadas por un protocolo de petición-respuesta.

Entre las características clave de un servicio web se pueden citar las siguientes [Coulouris *et al.*, 2012]:

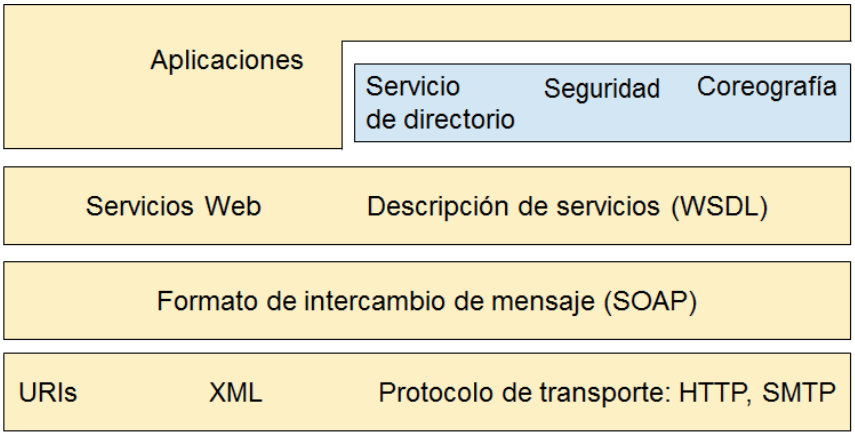
- Interoperabilidad.
- Estándares abiertos.
- Acoplamiento débil.

La interoperabilidad implica que distintas aplicaciones desarrolladas en diferentes lenguajes de programación y desplegadas en cualquier plataforma de cómputo pueden intercambiar datos a través de los servicios web.

Los servicios web soportan interoperabilidad a través de la Internet global, incluyendo el área clave de integración de empresa-a-empresa y también proporcionan el middleware subyacente tanto para el cómputo grid como para el cómputo en la nube.

El éxito de la interoperabilidad en los servicios web se logra gracias a la adopción de protocolos y estándares abiertos. Los responsables de la estandarización y arquitectura de los servicios web son el World Wide Web Consortium [W3C, 2014] y The Organization for the Advancement of Structured Information Standards [OASIS, 2014]. En la figura 6.4 se muestra un conjunto (pila) de servicios y protocolos de los servicios web.

Figura 6.4. Componentes e infraestructura de los servicios web [Coulouris *et al.*, 2012]



Donde:

- *XML (Extensible Markup Language)*: Es el lenguaje estándar para los datos que se intercambian. Es flexible y extensible.
- *SOAP (Simple Object Access Protocol)*: Son los protocolos sobre los que se establece el intercambio entre extremos.
- *WSDL (Web Services Description Language)*: Es el lenguaje de la interfaz pública para los servicios web. Está basada en XML e incluye la información necesaria para suplir la ausencia de un middleware.
- Protocolos *HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol)* o *SMTP (Simple Mail Transfer Protocol)*: Son usados para enviar los datos XML de una aplicación a otra.

- *UDDI (Universal Description, Discovery and Integration)*: es el protocolo para publicar la información de los servicios web.
- *WS-Security (Web Service Security)*: es el protocolo de seguridad para garantizar la autenticación de los actores y la confidencialidad de los mensajes enviados.
- *URI (Uniform Resource Identificator)*: es una secuencia compacta de caracteres que identifca un recurso físico o abstracto. Está definido por el RFC 3986. Ejemplos de URI pueden ser URL (Uniform Resource Locator) o URN (Uniform Resource Name).

Con el propósito de reducir el riesgo de que un cambio en un servicio web tenga un efecto en cadena sobre otros servicios, hay un interés en la articulación flexible o acoplamiento débil en los sistemas distribuidos. En el contexto de los servicios web, la articulación flexible se refiere a minimizar las dependencias entre servicios, con el fin de tener una arquitectura subyacente flexible.

Los servicios web se pueden utilizar sobre cualquier protocolo de comunicación, sin embargo, el más usado es el TCP (Transmission Control Protocol). EL TCP usa el puerto 80, el cual no es bloqueado por el cortafuego (firewalls) de las organizaciones. Este es generalmente el puerto usado por los navegadores de Internet. De esta manera, los servicios web pueden usar este puerto para no ser bloqueados.

EJERCICIOS

1. Explica la propiedad A.C.I.D. en las transacciones.

2. Cita las dos reglas de operación de los candados en las transacciones.

3. ¿Qué pasa con un dato cuando se aborta una transacción?

4. Cita algunos ejemplos del uso de los servicios web.

5. Desarrolla un ejemplo de escritura prematura para una concurrencia.

6. Indica la razón principal por la que los servicios web son soportados por el TCP.

7. Explica el caso de una lectura sucia en una transacción.

8. ¿Por qué el "acoplamiento débil" es una característica deseada en los servicios web?

9. Describe un ejemplo de transacción anidada.

10. Describe un ejemplo de transacción distribuida.

11. ¿Cuál es la función de los candados en una transacción?

Actividad integradora

1. De la aplicación de los contenidos, así como de las habilidades y actitudes desarrolladas.
   Se recomienda exposiciones por parte del profesor, así como una explicación de las características fundamentales de las transacciones distribuidas y su vinculación con la replicación, consistencia, localización de contenido, candados, recuperación y los deadlock. A modo de taller, se recomienda la realización de ejemplos sobre diversos casos a través de los cuales una transacción pueda caer en inconsistencia, tanto en el aula como de modo extraclase. Se recomienda desarrollar el contenido en un promedio de dos a tres sesiones que incluyan trabajo intenso de ejemplos demostrativos sobre la manera en que trabajan las transacciones, así como de los servicios web.
   El alumno debe de mostrar un fuerte interés por abstraer y modelar las transacciones distribuidas que pueden caer en estado de inconsistencia en los sistemas de cómputo distribuido. También tiene que demostrar habilidades para programar algoritmos relacionados al deadlock en las transacciones distribuidas en algún lenguaje de programación de alto nivel.
2. Del logro de los objetivos establecidos en el capítulo o apartado del material.
   Se espera alcanzar los objetivos de este capítulo con el apoyo en la solución de los ejercicios, así como en talleres con diversos casos de transacciones distribuidas que eviten inconsistencia o el deadlock en los sistemas distribuidos.

Capítulo 7. Sistemas operativos distribuidos

Objetivo: Que el alumno comprenda la importancia de los sistemas operativos distribuidos y la funcionalidad de sus componentes para el soporte de aplicaciones distribuidas de cómputo.

7.1 Introducción

Los sistemas operativos distribuidos presentan algunas de las siguientes ventajas [Sánchez, 1995]:

- Facilitan la implementación de sistemas distribuidos.
- Proveen abstracciones de los recursos en un sistema distribuido, por ejemplo, canales de comunicación y procesos en lugar de redes y procesadores.
- En los sistemas abiertos no existe una clara división entre el sistema operativo distribuido y las aplicaciones que se ejecutan en él.

7.2 Núcleo y servidores

Los núcleos y servidores de un sistema operativo distribuido administran recursos los cuales deben proveer las siguientes funciones [Sánchez, 1995]:

- *Encapsulamiento*: Deben de tener una interfaz de servicio útil para que el cliente pueda acceder sus recursos sin necesidad de que conozca los detalles de implementación.
- *Procesamiento concurrente*: implica que los recursos y accesorios del sistema pueden ser compartidos de manera simultánea por los clientes.

- *Protección*: Se deben de proteger los recursos contra accesos no autorizados.
- *Invocación*: se refiere al acceso a un recurso encapsulado y tiene las siguientes tareas relacionadas:
  - *Resolución de nombres*: El servidor que administra un recurso debe de ser localizado por medio del identificador del recurso.
  - *Comunicación*: Los parámetros y resultados se tienen que enviar a los administradores de recursos a través de una red.
- *Itinerario*: Cuando se invoca una operación, su procedimiento debe de ser itinerado dentro del núcleo o servidor.

Características del núcleo

El núcleo de un sistema operativo se caracteriza porque es un programa que se ejecuta con privilegios de acceso a los dispositivos físicos de la computadora huésped, como son los registros de memoria. El núcleo se responsabiliza de separar y proteger los espacios de memoria de todos los procesos, así como de la gestión de los registros. Los recursos del núcleo pueden ser accedidos por los procesos por medio de llamadas al sistema.

Los núcleos pueden ser [Sánchez, 1995]:

- *Monolíticos*: Cuando realizan todas las funciones básicas del sistema operativo, por ejemplo el UNIX. Un problema de estos núcleos es que ocupan mucho espacio (aproximadamente 200 MB).
- *Microkernel:* En este enfoque el núcleo es muy pequeño y sólo contienen las funciones más básicas tales como la comunicación entre procesos, servidores y memoria. El resto de funciones es previsto por servidores cargados dinámicamente (por ejemplo: MACH).

7.3 NOMBRAMIENTO Y PROTECCIÓN DE RECURSOS

7.3.1 Nombrado de recursos

Un servicio que administra recursos debe de dar un nombre a cada uno de estos recursos. Este nombre debe de ser independiente de su localización (memoria, computadora). Por ejemplo, el uso de recursos en Chorus y Amoeba requiere [Coulouris *et al.*, 2001], [Tanenbaum, 1996]:

- Puerto del servidor
- Identificación del recurso
- Conocer el puerto y el identificador del recurso
- Los nombres deben de ser únicos y accesibles para ser localizados
- Usar mensajes a puertos para manejar recursos

La ventaja del nombramiento de recursos es que existe una transparencia de red que es la que busca un sistema operativo distribuido, porque el servidor puede estar en otra máquina o en la máquina local y sin embargo debe ser transparente.

7.3.2 Protección de recursos

La protección de recursos se refiere a que los clientes solo podrán acceder a aquellos recursos para los cuales tengan permisos de acceso. La protección de recursos se logra implementando un dominio de protección, el cual es un conjunto de reglas de acceso de recursos compartidos por un conjunto de procesos. Existen dos formas de realizar la implementación:

1. Los procesos contienen un conjunto de capacidades o identificadores de servicios (con sus respectivos derechos) que pueden usar, y cada proceso conoce qué recurso usar.
2. Cada recurso tiene una lista de control de acceso, la cual es concentrada en el servidor como una lista de las personas que pueden tener acceso al sistema.

7.4 Casos de sistemas operativos distribuidos

Diferentes sistemas operativos distribuidos han sido desarrollados por consorcios industriales, centros de investigación y universidades. Algunos ejemplos de sistemas distribuidos son:

- *Mach*: Sistema de punta con un diseño flexible de memoria.
- *Chorus*: Sistema de punta con diseño de microkernel.
- *DCE*: Sistema comercial con ambiente coherente para aplicaciones distribuidas.

A continuación se revisa brevemente las principales características de estos sistemas.

7.4.1 Mach

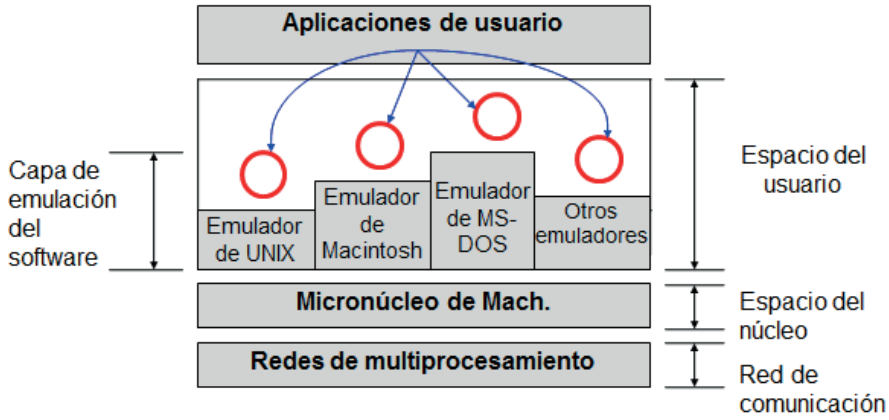
7.4.1.1 Generalidades

El sistema operativo Mach es un proyecto de la Universidad Carnegie Mellon (CMU), desarrollado en 1986. Entre los objetivos de este sistema se encuentran:

- Crear una base para emular otros sistemas.
- Proveerlo de facilidades avanzadas en el núcleo.
- Permitir la transparencia en el acceso a recursos en red.
- Explorar el paralelismo.

La versión 2.5 de Mach fue usada por el sistema OSF/1 como base. La versión 3 eliminó la emulación de UNIX del núcleo. Mach puede ejecutarse en máquinas con procesadores 386, 486, estaciones DEC y SPARC, además puede funcionar en máquinas con un solo procesador o con múltiples procesadores. El acceso al núcleo puede ser un problema en Mach si este ha sido mal diseñado. Mach permite manejar memoria virtual como si fuera memoria compartida. Los recursos son manejados a través de mensajes de puertos en los servidores a nivel usuario. Mach permite la portabilidad, ya que el código dependiente de máquina puede ser asilado. En la figura 7.1 se muestra una arquitectura de Mach.

Figura 7.1. Arquitectura de Mach [Tanenbaum, 1996]



7.4.1.2 El micronúcleo y los puertos en Mach

El micronúcleo en Mach controla las siguientes abstracciones:

- *Puertos:* Canal de comunicación unidireccional que tiene asociado un buffer. A través de estos puertos se puede enviar o recibir mensajes.
- *Proceso:* Ambiente donde se lleva a cabo la ejecución, contiene un espacio de direcciones protegidas y un conjunto de capacidades para acceder a puertos.
- *Hilos:* Los procesos están constituidos de diferentes hilos que se pueden ejecutar en paralelo a diferentes procesadores de una máquina con memoria compartida.
- *Mensajes:* Es un conjunto de datos o peticiones para acceder a recursos.
- *Objetos de memoria*: Es una instancia de un tipo de dato abstracto, por cada objeto de memoria existe un objeto del núcleo que contiene un caché de las páginas residentes en memoria.

Características de los puertos en Mach

Los puertos sirven para acceder a recursos. Existen tres diferentes tipos de derechos para acceder a un puerto:

- De envío.
- De envío una sola vez.
- De recepción.

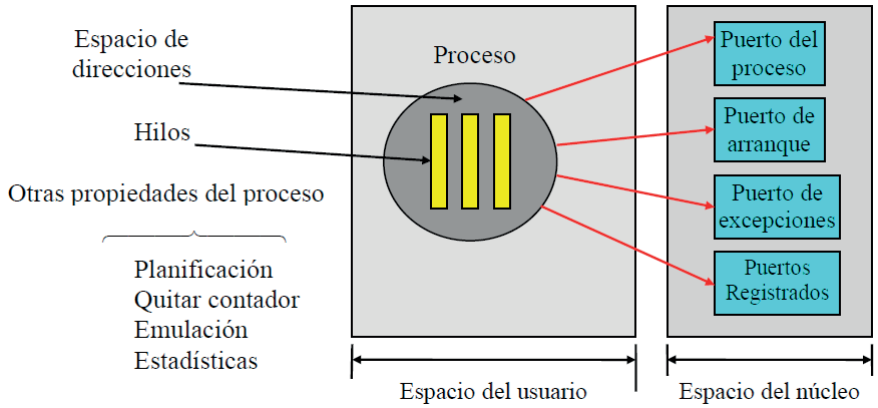
En un puerto varias tareas pueden tener derechos de envío pero solo una tarea tendrá los de recepción.

7.4.1.3 Administración de procesos en Mach

Procesos en Mach

Los procesos son solo medios de ejecución, ya que solo los hilos que se encuentran en él pueden ejecutar operaciones. En la figura 7.2 se muestra la operación de un proceso.

Figura.7.2. Un proceso en el sistema Mach [Tanenbaum, 1996]



Los puertos que se muestran en la figura 7.2 tienen las siguientes funciones:

- *Puerto de arranque*: Usado en la inicialización de todos los parámetros al comenzar el proceso.
- *Puerto de excepciones*: Se utiliza para informar de las excepciones ocasionadas por el proceso.
- *Puerto de proceso*: Permite la comunicación con el núcleo.
- *Puerto de registrados*: Permite al proceso una forma de comunicación con otros servidores estándar del sistema.

Hilos en Mach

Los hilos son entidades activas en Mach que ejecutan instrucciones, controlan sus registros y espacios de direcciones. Cada hilo pertenece exactamente a un proceso específico, el cual no podrá realizar algo si no tiene por lo menos un hilo. La creación de hilos en Mach sigue el modelo de UNIX y usa la llamada *fork*, mientras que con la llamada *exit* termina un hilo su trabajo. Cuando un hilo es creado se le asigna un puerto para que pueda acceder al núcleo.

7.4.1.4 Modelo de comunicación en Mach

La comunicación en Mach está basada en los puertos, los cuales son objetos del núcleo que contienen mensajes. Un mensaje está constituido de un encabezado de tamaño fijo, seguido por una lista variable de datos. El encabezado de un mensaje está compuesto de los siguientes elementos:

- Puerto destino
- Puerto respuesta
- Identificador de operación
- Tamaño de la lista de datos

Cada dato en la lista puede ser un apuntador a datos, un dato tipo o tipos de derechos sobre puertos. Se usa la llamada *mach_msg* para el envío y recepción de mensajes. Los puertos tienen un buffer cuyo tamaño puede ser ajustado. Si se intenta enviar un mensaje por un puerto que tenga su buffer lleno, el hilo se bloqueará. Los mensajes pueden ser recibidos en un conjunto de puertos. La confiabilidad en la comunicación del sistema existe y, además, permite la entrega ordenada de los mensajes. Servidores de mensaje son usados para realizar la comunicación a través de la red. Estos servidores están constituidos por varios hilos, los cuales llevan a cabo distintas funciones. La comunicación externa se realiza por los puertos de red, los cuales soportan el protocolo TCP/IP.

7.4.1.5 Manejo de memoria en Mach

Mach cuenta con un sistema de memoria flexible basado en la paginación, el cual separa las partes dependientes de la máquina de las partes independientes, lo que le permite una portabilidad. Las páginas solo se copian cuando han sufrido modificación y para esto se usa la llamada *copy*. Debido a que Mach no soporta archivo, debe usar el paginador externo, el cual tiene, entre sus funciones:

1. El manejo del respaldo de datos eliminados por el núcleo de su caché.
2. El mantener un registro de las páginas virtuales en uso.
3. El definir las restricciones necesarias que permitan mantener la consistencia de los datos.

Mach maneja una memoria virtual grande y lineal. Cuando se crea una tarea, se hereda memoria, la cual podrá ser compartida o copiada.

7.4.2 Chorus

7.4.2.1 Generalidades

Este sistema es un proyecto del INRIA, desarrollado en 1980 [Tanenbaum, 1996]. En 1997 el Chorus fue adquirido por Sun Microsystems. Algunos de los objetivos de Chorus son:

- Permitir una emulación de UNIX de alto rendimiento.
- Para usar en sistemas distribuidos.
- Aplicaciones en tiempo real.
- Integrar programación orientada a objetos.
- Permite servidores de grupo y ser reconfigurable.

Algunos aspectos que guardan en común Chorus y Mach son:

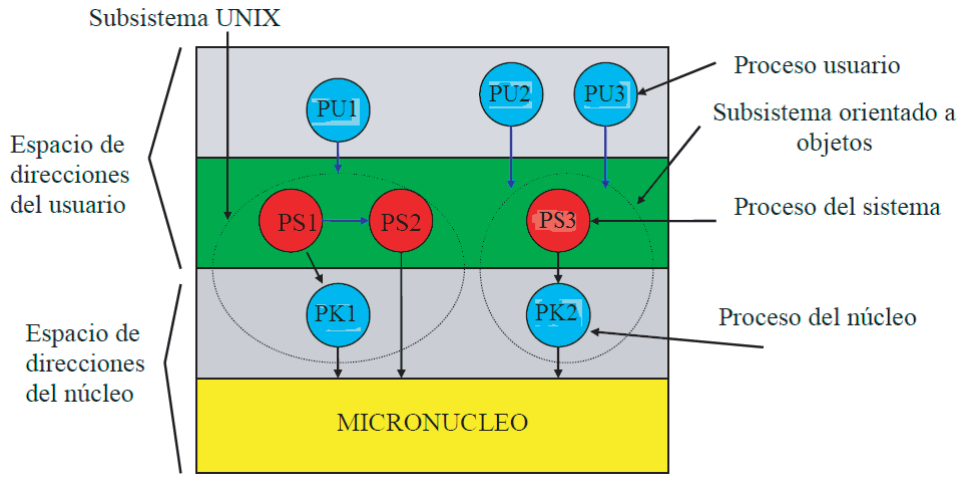
- Operación paralela con memoria compartida.
- Transparencia a nivel de red.
- Emulación de UNIX y otros sistemas operativos.
- Flexibilidad al implementar la memoria virtual.
- Portabilidad.
- Tienen servidores de grupo reconfigurable.

Estructura

Chorus se estructura en capas, con el *micronúcleo* en la parte inferior, que lleva una mínima administración de los nombres, procesos, hilos, memoria y mecanismos de comunicación. Los *procesos del núcleo* se sitúan por encima del microkernel y se cargan de manera dinámica para cada nuevo programa permitiendo adicionar funcionalidad al microkernel sin aumentar su tamaño y complejidad. Los *procesos del sistema* se encuentran en la capa superior siguiente, se ejecutan en modo usuario y envían mensajes a los procesos del núcleo. Finalmente, en la capa superior se encuentran los procesos de usuario, los cuales no podrán llamar directamente al microkernel, excepto los definidos para tal fin. La figura 7.3 ilustra la estructuración de estas capas en el sistema Chorus. El micronúcleo conoce qué subsistema ocupa cada proceso

usuario, por lo que las llamadas de este proceso se restringirán únicamente a las ofrecidas por este.

Figura 7.3. Estructura de capas en Chorus [Tanenbaum,1996]



7.4.2.2 Micronúcleo del sistema Chorus

El Micronúcleo en el sistema Chorus controla las siguientes abstracciones:

- *Actores*: Son en esencia equivalentes a los procesos en otros sistemas operativos y son un medio que encapsula a los hilos.
- *Hilos*: Son similares a un proceso, ya que tienen una pila, apuntador a la pila, contador de programas y registros, pero comparten el mismo espacio de direcciones.
- *Puertos:* Es el canal de comunicación que tiene asociado un buffer para recepción de mensajes. Los puertos se pueden agrupar para que una tarea pueda mandar o recibir datos.
- *Mensajes*: Datos o peticiones para poder acceder a un recurso, los cuales tienen una parte fija y otra de tamaño variable. El cuerpo del mensaje contiene la información enviada por el emisor.
- *Regiones:* Es un rango consecutivo de direcciones que se asocian con alguna pieza de datos, ya sea programa o archivo.
- *Segmento*: Es el conjunto lineal de bits en la cual podemos asociar a regiones y que son guardadas en el núcleo en un caché local.

7.4.2.3 Administración de procesos en Chorus

Actores en Chorus

Un actor (o proceso) en Chorus es un conjunto de elementos activos y pasivos que se ejecutan en grupo para realizar cierto trabajo. En el sistema Chorus, los actores y los hilos pueden ser cargados de manera dinámica en el espacio del núcleo. Del mismo modo, los servidores se pueden cargar activamente, ya sea en el espacio del núcleo o del usuario. Existen tres tipos de privilegios para los actores en Chorus:

- *De usuario*: Solo algunas llamadas a sistema.
- *De sistema*: Permite las llamadas al sistema.
- *De núcleo o supervisor*: Tiene acceso completo.

Cada actor en Chorus tiene asociado un *identificador de protección*, el cual les proporciona un mecanismo de autentificación.

Hilos

En Chorus, los hilos son los que ejecutan el código en un actor y tienen su propio contexto privado. Siempre permanecen unidos al actor en el que fueron creados. Es función del núcleo conocer y planificar cada hilo. Los hilos se comunican entre sí, ya sea enviando o recibiendo mensajes, sin importar que los hilos receptores se encuentren en otros actores. Un hilo distingue los siguientes estados no excluyentes (un hilo puede estar en más de un estado al mismo tiempo):

- *Activo*: El hilo se puede ejecutar.
- *Suspendido*: El hilo ha sido intencionalmente suspendido.
- *Detenido*: El actor donde se encuentra el hilo se ha suspendido.
- *Espera*: El hilo está en espera a que un evento se realice.

Chorus usa dos registros de software para solucionar el problema de administrar los datos particulares y su pila en un hilo. En cada registro está un apuntador a los datos particulares cuando opera en modo usuario y otro apuntador cuando el hilo envía una señal al núcleo.

7.4.2.4 Manejo de memoria en Chorus

Chorus guarda varias semejanzas con respecto al manejo de memoria con Mach. Chorus usa los siguientes conceptos [Tanenbaum, 1996]:

- *Regiones*: Que corresponde a un rango en serie de direcciones virtuales.
- *Segmento:* Es una colección adyacente de bytes que recibe el nombre y protección de una posibilidad. Ejemplos de segmentos son los archivos y las áreas de intercambio.
- *Asociadores*: Son usados para controlar uno o más segmentos a asociar con regiones.

Chorus soporta la memoria compartida distribuida clásica usando un algoritmo descentralizado dinámico, donde el segmento es la unidad a ser compartida. Las llamadas más importantes soportadas por el administrador de memoria se relacionan con *administración de regiones*, *administración de segmentos*, *asociadores*.

7.4.2.5 Comunicación en Chorus

La comunicación en Chorus está basada en la transferencia de mensajes. Un mensaje está constituido por un encabezado que contiene la fuente y destino, una parte fija opcional y un cuerpo opcional, estas dos últimas partes están bajo el control del usuario. Existen tres tipos de mensajes a grupos:

- *Por difusión*: Llega a todos los miembros del grupo.
- *Funcional*: Dirigido a algún miembro del grupo.
- *Selectivo:* A algún miembro del grupo donde se encuentra un recurso específico.

Chorus permite el agrupamiento de puertos para diferentes servidores y proporciona dos formas de comunicación: asíncrono y RPC.

7.4.2.6 Emulación de UNIX en Chorus

Aunque Chorus no tuvo como objetivo inicial la emulación con UNIX, se creó un subsistema (MiX) que permite la compatibilidad binaria. Los procesos, tanto en Chorus como en MiX, comparten el mismo espacio de memoria. El Modulo UNIX en Chorus permite una simulación de llamadas al sistema y

usa una protección de datos similar a la usada en UNIX. En este sistema las señales son controladas por el propio proceso a través de un hilo de control y permite la creación de procesos en computadoras remotas.

MiX está construido con base en los siguientes componentes:

- Administrador de procesos.
- Administrador de objetos.
- Administrador de control de flujo.
- Administrador de comunicación.

7.4.3 DCE

7.4.3.1 Generalidades y objetivos

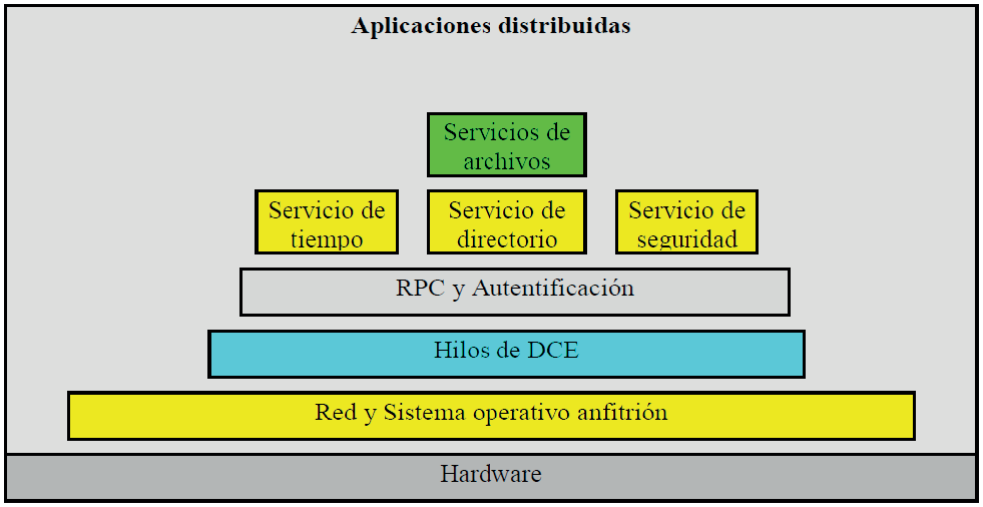
DCE (Distributed Compuyting Enviroment) fue desarrollado por OSF (Open Systems Foundation) que agrupa a compañías como IBM, Digital y Hewlett-Packard, en 1980 [Tanenbaum, 1996]. El objetivo de DCE es ofrecer un ambiente coherente como plataforma para el desarrollo de aplicaciones distribuidas. La base de DCE fueron los sistemas operativos existentes, en principio UNIX para después considerar VMS, WINDOWS y OS/2.

DCE tiene diversas herramientas y servicios, así como la infraestructura para que estos trabajen de manera integrada con el fin de facilitar aplicaciones distribuidas. DCE permite crear un mercado potencial al ejecutarse en distintas arquitecturas, sistemas operativos y redes, además de ofrecer transparencia al usuario. Mecanismos de autentificación y protección integrados en DCE permiten que este sistema brinde seguridad en red a un grupo de trabajo. DCE utiliza los protocolos TCP/IP u OSI para realizar la comunicación en red.

Estructura de DCE

El cliente - servidor es el modelo de DCE, ya que los procesos usuarios operan como clientes para obtener un servicio de un proceso servidor. Los servicios distribuidos proporcionados por DCE son, principalmente, servicio de tiempo, directorio, seguridad y sistema de archivos. Entre otras prestaciones de DCE, se encuentran los hilos y las RPC (llamadas a procedimientos remotos). La figura 7.4 indica el modelo DCE y cómo se relacionan las partes que la componen.

Figura 7.4. Modelo del DCE [Tanenbaum, 1996]



Las celdas en DCE son agrupamientos de usuarios, máquinas y otros recursos sobre la cual están basados aspectos como asignación de nombres, seguridad y administración. Para agrupar las celdas, se consideran las siguientes restricciones:

- *Finalidad*: Los agrupados en una celda deben trabajar con un objetivo común.
- *Seguridad:* Agrupar a los usuarios que mejor confianza tengan entre sí.
- *Costo:* Agrupar a los usuarios que por su localización representen menos costos.
- *Administración:* El nombrar a un administrador de la celda se facilita cuando las restricciones anteriores son previstas.

Con el fin de minimizar el número de operaciones entre celdas en DCE y considerando las restricciones indicadas, es recomendable tener una pequeña cantidad de celdas.

7.4.3.2 Los hilos de DCE

Los hilos constituyen una parte fundamental en DCE, entre los puntos a considerar sobre ellos están la planificación, la sincronización y las llamadas. Están basados en POSIX P1003.4a. Un hilo maneja cuatro estados:

- *Ejecución*: El hilo está en actividad con el CPU.
- *Listo*: El hilo se encuentra en espera de entrar en actividad.
- *En espera*: El hilo se bloquea mientras espera que un evento ocurra.
- *Terminado*: El hilo ha finalizado su actividad pero sigue existiendo.

La planificación en los hilos es lo que determina el tiempo y orden de ejecución de cada hilo. Los algoritmos de planificación a soportar por DCE son:

- *FIFO*: Ejecuta el primer hilo de la cola de máxima prioridad de un grupo de colas.
- *Round-robin*: Ejecuta cada hilo durante un *quantum* fijo para la cola más poblada.
- *Por omisión*: Usa round-robin pero mientras la prioridad sea mayor el hilo tendrá un *quantum* mayor.

Para realizar la sincronización de hilos en DCE se usan:

- *Mutex*: Existen los rápidos, recursivos y no recursivos.
- *Variable de condición*: Son usados con los mutex como otro mecanismo de sincronización.

7.4.3.3 RPC en DCE

Entre los objetivos del uso de RPC en DCE se encuentran:

- Permitir que un cliente acceda a un servicio remoto a través de un procedimiento local.
- Facilitar la escritura de programas cliente y su transparencia.
- Tener pocas modificaciones en el código al ejecutarse.

El RPC de DCE consta de distintos componentes, incluyendo lenguajes, bibliotecas, demonios y programas de utilerías, en la cual la interfaz se escribe en IDL (Interface Definition Language). La salida del compilador IDL en su salida está constituido por los archivos:

- *Archivo de encabezado*: Con el identificador, tipos, constantes y los prototipos de función.
- *Resguardo del cliente*: Se encarga de almacenar los procedimientos reales.

- *Resguardo del servidor:* Aloja los procedimientos llamados por el sistema cuando un mensaje está arriba.

Para conectarse al servidor, el cliente primero localiza cuál es la máquina servidora y después localiza el proceso correcto en el servidor usando una dirección que se asigna dinámicamente.

7.4.3.4 Los servicios en RPC

Servicio de tiempo

DCE tiene un servicio distribuido de tiempo (DTS) que sincroniza los relojes de las máquinas distantes, este registro de tiempos se realiza por intervalos. Las funciones de DTS se centran en mantener:

- La consistencia mutua de los relojes: El mismo valor de tiempo en todos.
- Manejar tiempo real: Implica que el valor coincida con el del mundo real.

Cuando un programa solicita a DTS una comparación de tiempos, las respuestas pueden ser las siguientes:

- Primer tiempo es menor.
- Segundo tiempo es menor.
- No se sabe cuál es menor.

El empleado del tiempo es un demonio de DTS que se ejecuta en el cliente manteniendo el tiempo sincronizado con los relojes remotos.

Servicio de directorios

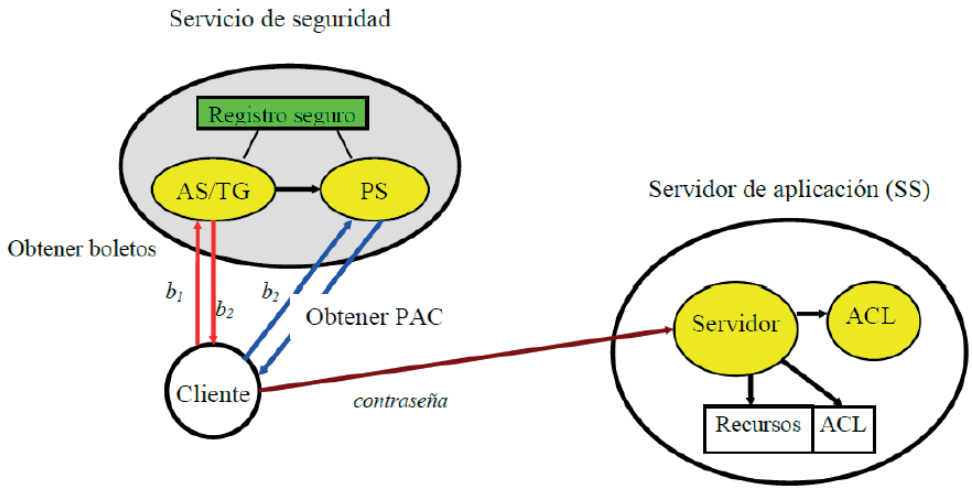
En DCE el servicio de directorio que nombra y registra todos los recursos está constituido por celdas, cada una con un servicio de directorio de celdas (CDS). Cada recurso tiene un nombre constituido por el nombre de la celda y el suyo en ella. Los mecanismos GDS (Global Directory Service) y DNS (Domain Name System) son usados por DCE para localizar celdas. CDS ordena los nombres de las celdas por jerarquía, aunque también permite los enlaces simbólicos. El GDS es un servicio secundario de DCE que le permite localizar celdas remotas, así como información arbitraria relativa a los directorios.

Servicio de seguridad

La seguridad en los sistemas distribuidos es esencial y DCE la proporciona en forma eficaz [Hu, 1995], [Tanenbaum, 1996]. Algunos conceptos de seguridad que involucra DCE son:

- *Principal*: Es un proceso que requiere seguridad en la comunicación. Tiene asociado un número binario como *identificador único de usuario*.
- *Autentificación*: Rutina para determinar si un *principal* es realmente quien dice ser, para lo cual DCE se basa en el sistema Kerberos del MIT (Massachusetts Institute of Technology).
- *Autorización*: Es usado para determinar a cuáles recursos tiene acceso el *principal* y se utiliza una *lista de control de acceso* (ACL).
- *Criptografía:* Transforma datos de tal manera que sea imposible recuperar los datos originales si no se posee una llave.
- El sistema de seguridad está constituido principalmente por los componentes que se indican en la figura 7.5.

Figura 7.5. Componentes elementales de la seguridad en DCE [Tanenbaum, 1996]



El *servidor de autentificación (AS)* es usado al iniciar el sistema o cuando un cliente entra en él. El *servidor de emisor de boletos (TGS)* se encarga de entregar el boleto al cliente o principal. El *servidor de privilegios (PS)* emite los certificados PAC. Cada usuario tiene una clave secreta solo conocida por él y el registro en el servicio de seguridad. Un *boleto* es un conjunto de

datos cifrados, que básicamente tiene las siguientes opciones: servidor pretendido, clave de sesión, cliente y tiempo de expiración, estos tres últimos cifrados mediante una clave temporal K_n.

Los pasos para obtener una RPC autentificada son [Tanenbaum, 1996]:

1. El cliente se autentifica con el Servidor de Autentificación (AS) usando una contraseña, el AS le entrega su llave secreta Kc y la clave C_1 cifradas en un boleto b_1 a usar con TGs.
2. El cliente envía el boleto b_1, cifrado con la clave C_1 al TGS, solicitando un boleto b_2 para comunicarse con el servidor de privilegios (PS). TGS verifica el boleto b_1, la clave C_1 y la marca de tiempo y, si esto es correcto, TGS envía al cliente un boleto b_2 cifrado que solo el cliente que cuenta con la clave K_c podrá abrir.
3. El cliente solicita con el boleto b_2 cifrado con C_2 al PS un certificado de atributos de privilegios (PAC) que contiene la identidad del usuario y grupo al que pertenece. El PAC es enviado al cliente en forma cifrada con C_2.
4. El PAC es enviado por el cliente al servidor emisor de boletos (usando C_3), quien descifra el PCA para corroborar su autenticidad y lo vuelve a cifrar con una nueva clave C_4 para enviarlo al cliente.
5. Finalmente el PAC cifrado con C_4 es enviado al servidor de aplicaciones (SS), quien lo descifrará exhibiendo la clave C_4. Entonces, el servidor responde con la clave C_5 que solo es conocida por el cliente y el servidor para permitir una comunicación segura.

Los recursos en DCE pueden estar protegidos por una lista de control de acceso (ACL) para indicar quién puede acceder y los derechos de acceso. Los derechos estándares soportados son: leer, escribir, ejecutar, modificar ACL, insertar recipiente, eliminar recipiente y probar.

Sistema distribuido de archivos (DFS)

El DFS en DCE permite que los procesos puedan acceder a archivos autorizados, aunque estos se localicen en celdas distantes. El DFS está constituido principalmente por la parte local y la parte de área amplia. El DFS usa hilos para el servicio simultáneo a varias solicitudes de acceso a un archivo, considerando a cada nodo como un cliente o servidor de archivo; guarda similitud con respecto a la interfaz con UNIX para abrir, leer y escribir archivos, así como en el montaje de archivos remotos. El DFS soporta archivos individuales, directorios, conjunto de archivos y particiones de disco (agregados en DCE).

EJERCICIOS

1. ¿Cuál es la función de los hilos en Mach?
2. ¿Cuál es la principal aportación de Mach a los sistemas distribuidos?
3. ¿Cómo trabaja el algoritmo LRU?
4. Cita tres aspectos que guardan en común Mach y Chrorus.
5. ¿Cómo se realiza la implementación de UNIX en Chorus?
6. Explica el concepto de micronúcleo.
7. ¿Para qué se extendieron la semántica de las señales de UNIX en Chorus?
8. ¿Cuáles son las principales diferencias del sistema operativo distribuido DCE con respecto a Chrorus y Mach?
9. ¿Cómo estructura su modelo de seguridad DCE?
10. ¿Cuál es la función del servicio de distribución de tiempo en el sistema operativo distribuido DCE? y ¿cómo opera?

ACTIVIDAD INTEGRADORA

1. De la aplicación de los contenidos, así como de las habilidades y actitudes desarrolladas.
   Se recomienda realizar una exposición general del tema para que el alumno comprenda la importancia de los sistemas operativos distribuidos, como el soporte para aplicaciones distribuidas, sin profundizar en detalles que extiendan el tema. Dado las generalidades del tema por el profesor, se puede desarrollar cada caso específico de sistema operativo distribuido en equipos formados por tres o cuatro estudiantes, con el fin de realizar una exposición de los casos en una sesión. Se recomienda desarrollar el contenido en dos sesiones que incluyan trabajo intenso con ejemplos demostrativos de la manera en que trabajan las transacciones, así como de los servicios web.

El alumno debe de mostrar un gran interés por entender los componentes básicos de los sistemas operativos distribuidos, así como profundizar la investigación de estos sistemas en la literatura científica existente.

2. Del logro de los objetivos establecidos en el capítulo o apartado del material.

   Se espera alcanzar los objetivos de este capítulo con el apoyo en la solución de los ejercicios, en las exposiciones realizadas por los alumnos, así como en la lectura de artículos de investigación por parte de los alumnos.

Capítulo 8. Sistemas de archivos distribuidos

Objetivo: Que el alumno comprenda la importancia de los sistemas de archivos distribuidos, sus componentes, estructura y las operaciones sobre ellos, así como algunos casos de sistemas de archivos distribuidos.

8.1. Introducción

Un archivo es una colección no contigua de bloques o conjunto de apuntadores a bloques del archivo en el índice de bloques que guardan una interfaz con el módulo cliente. Un archivo es la abstracción de datos guardados en almacenamiento secundario. Un directorio es un archivo especial que permite organizar los archivos. El sistema de archivos en un sistema operativo distribuido se encarga de la organización, almacenamiento, nombrado, repartición y protección de archivos. Importa notar que un sistema de archivos no es igual a una base de datos. Las transparencias requeridas en un sistema de archivos distribuidos son [Coulouris *et al.*, 2001]:

- Acceso.
- Localización.
- Concurrencia.
- Falla.
- Desempeño.

8.2 Servicios de archivos

Otros requerimientos para los sistemas de archivos distribuidos son heterogeneidad de replicación y de migración. Los módulos que constituyen el sistema de archivos son:

- *Servicio de archivos planos:* Está compuesto de módulos de archivos y de acceso a archivos. Se encarga de modificar el contenido de los archivos.
- *Servicio de directorios:* Está constituido por módulos de directorios y de control de acceso. Se encarga de definir un alfabeto y una sintaxis para formar los nombres de archivos.
- *Módulo cliente:* Está compuesto de módulo de bloques y dispositivos. Se encarga de proveer al usuario de una interfaz de programación. También se ocupa de las rutinas de bajo nivel para acceder a bloques de memoria y dispositivos.

8.3. SERVICIO DE ARCHIVOS PLANOS

Un servicio de archivos plano es un conjunto de operaciones simple y de propósito general. Los archivos contienen datos y atributos. La tolerancia a fallas se debe considerar para:

- Operaciones repetibles.
- Servidores sin estado.
- La atomicidad de operaciones como*:*
  - Read ( ).
  - Write ( ).
  - Create ( ).
  - Truncate ( ).
  - Delete ( ).
  - GetAttributes ( ).
  - SetAttributes ( ).

8.4 SERVICIO DE DIRECTORIO

El servicio de directorio proporciona una transformación entre nombres de texto para los archivos y sus UFID (Unique File Identifiers). Una UFID tiene el formato indicado en la figura 8.1. El UFID puede ser obtenido por los clientes de un archivo indicando su nombre de texto al servicio de directorio. Un UFID es único entre todos los archivos de todas las computadoras y es difícil de duplicar por personas externas. El UFID no es reusable y contiene información sobre la dirección física. Permite que los archivos sean divididos en grupos. Los archivos contienen campos de permisos encriptados.

Figura 8.1. Formato de la UFID

Identificador de grupo	Número de archivo	Número aleatorio	Permiso

Funciones delservicio de directorio

- Mapeo de nombres a UFIDs.
- Organización de los archivos planos en un árbol de directorios.
- Control del acceso a archivos.

8.5 MÓDULO CLIENTE

En cada computadora cliente se ejecuta un módulo conocido como módulo cliente. Este módulo cliente integra y extiende las operaciones del servicio plano de archivos con una interfaz de programación de aplicación disponible para los programas en las computadoras cliente. Un módulo cliente tiene las siguientes funciones:

- Se encarga del acceso a bloques de disco.
- Normalmente se tiene un sistema de caché para reducir tiempos de acceso.
- Usa el algoritmo LRU (Least Recently Used) para escoger un bloque que se regresa a disco.
- El cliente puede reducir retrasos de red usando un caché local.
- Uno de los inconvenientes es con la coherencia del caché.

8.6 SISTEMAS NFS Y AFS

Dos de los sistemas de archivos distribuidos más difundidos son los sistemas AFS y NFS [Coulouris *et al.*, 2001]. Las principales características de estos sistemas son descritas a continuación.

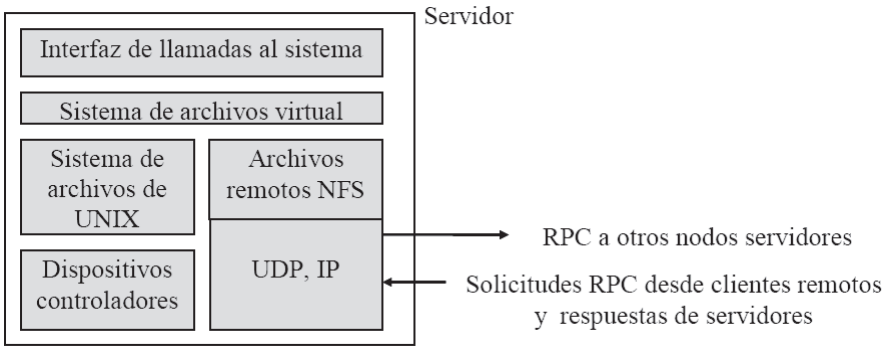
8.6.1 Sistema NFS (Network File Systems)

El sistema de archivos NFS fue diseñado por la compañía Sun Microsystems en 1985, con el propósito de ejecutarse en una red local (LAN). Entre las características de un sistema NFS se puede destacar lo siguiente:

- Es el estándar para tener acceso a archivos distribuidos en UNIX.
- Utiliza el modelo cliente - servidor y ambos se encuentran en el núcleo del sistema operativo.
- Permite la heterogeneidad de arquitecturas.
- Permite las transparencias de acceso, localización, fallas y desempeño.
  - Alcanza transparencia de localización.
  - La transparencia de migración es implementada parcialmente.
  - Las transparencias de replicación, concurrencia y escalabilidad no son implementadas.
  - NFS utiliza RPC. Es un servidor sin estados, por lo que no mantiene abiertos los archivos ni guarda información sobre ellos.

Para su implementación en UNIX, NFS define nuevas capas en el sistema de archivos de UNIX, como se muestran en la figura 8.2.

Figura 8.2. Estructuras de capas de NFS embebidas en el núcleo de UNIX



NFS utiliza un sistema de archivos virtuales VFS que puede distinguir entre archivos locales y archivos remotos. Usa un *v-node* en lugar de un *i-node*. Al *v-node* se le añade un identificador de sistema. NFS emula la semántica del sistema de archivos de UNIX integrándolo en el núcleo, por lo que no se requiere recompilar librerías. El caché de clientes tiene el problema de las diferentes versiones de archivos, ya que la escritura de un cliente no se traduce en la inmediata modificación de copias del caché en los otros clientes. Las estampillas de tiempo son usadas por el NFS para validar bloques de caché.

Inconvenientes del NFS

Entre los inconvenientes que presenta NFS se pueden mencionar las siguientes:

- No es eficiente para sistemas distribuidos grandes.
- Las implementaciones caché inhiben a tener un pobre desempeño en las operaciones de escritura.

8.6.2 Sistema AFS (Andrew File Systems)

Este sistema de archivos distribuido fue desarrollado en Carnegie Mellon University (CMU) en 1986. El objetivo de AFS es compartir información a gran escala (10,000 estaciones de trabajo). Al igual que en Network File Systems (NFS), en AFS no hay que modificar programas de UNIX. La estrategia clave para alcanzar escalabilidad en AFS es a través del manejo de caché de archivos completos en los nodos clientes.

El mecanismo de AFS considera lo siguiente [Sánchez, 1995]:

1. Un proceso cliente realiza un operación "open" en un espacio compartido, entonces el servidor envía una copia del archivo al cliente.
2. El cliente realiza todas las operaciones en la copia local.
3. Una vez que el archivo es cerrado, se envía la versión modificada al servidor, mientras que el cliente mantiene la copia local.

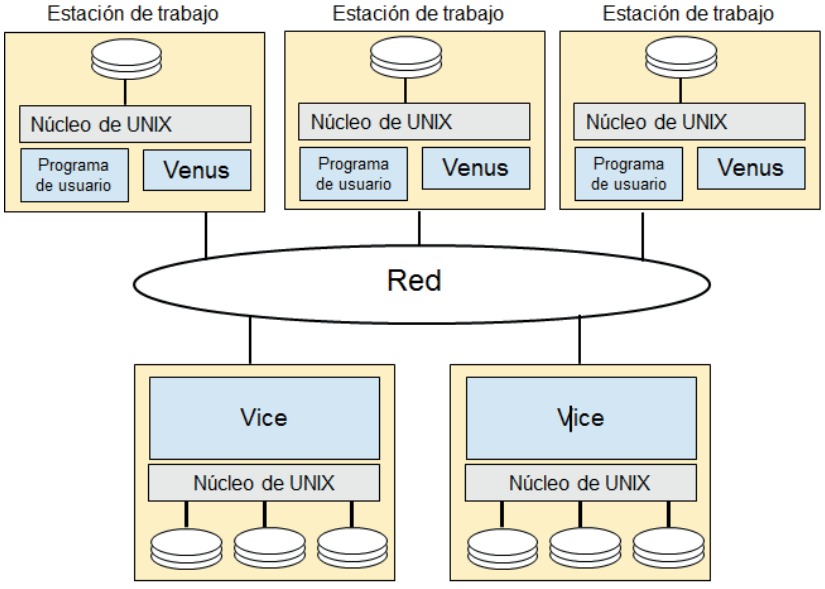
La estrategia de diseño de AFS está basado en UNIX, y considera las siguientes observaciones [Coulouris et al., 2001]:

1. Usar archivos pequeños.
2. Las lecturas se realizan con más frecuencia que las escrituras en una relación de seis a una.
3. El acceso a los archivos es secuencial.
4. Se comparten pocos archivos.
5. Generalmente un solo usuario es el responsable de modificar los archivos compartidos.

Entre las principales características del sistema AFS, se pueden indicar las siguientes [Coulouris et al., 2001]:

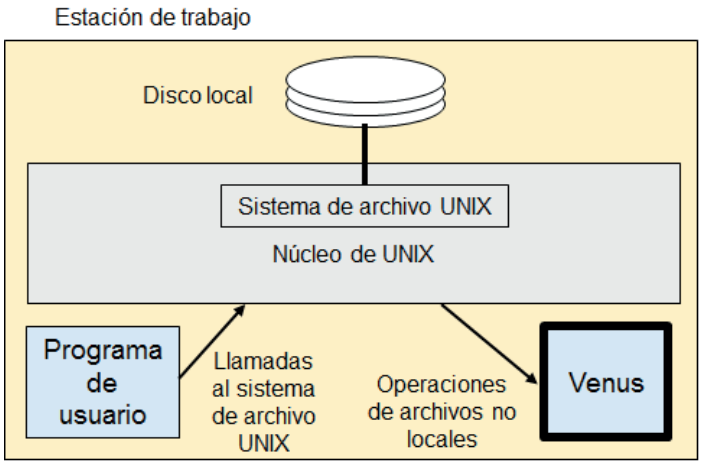
- Los cachés usados son del orden de 100 MB.
- El tamaño de los paquetes para la transferencia de archivos es de 64 KB.
- El uso de bases de datos no está considerado.
- Se usan directorios locales o compartidos para colocar los archivos.
- Está constituido principalmente por dos componentes de software a nivel usuario (ver figura 8.3):
  - Vice, en el servidor.
  - Venus, en el cliente.
- Cuando muchos clientes usan el AFS, se puede comparar favorablemente con respecto al NFS.
- Intercepta las llamadas a operaciones de archivos.
- Permite guardar información de estado sobre las copias de los archivos.
- Emplea réplicas de archivos, por lo que una nueva escritura deberá de ser hecha en todas las copias distribuidas.
- Soporta seguridad, protección y heterogeneidad.
- Los usuarios pueden cambiar de máquinas y usar el mismo nombre. Este proceso deberá ser transparente, excepto por algunas fallas de ejecución iniciales que son necesarias al poner los archivos de usuarios en las máquinas locales.

Figura 8.3. Arquitectura de distribución de procesos en AFS [Coulouris *et al.*, 2012]



AFS intercepta las llamadas al sistema de operaciones de archivos (ver figura 8.4) y los archivos forman grupos llamados volúmenes. Cada archivo y directorio en el sistema es identificado por un FID (File Identifier) único, que es usado para la comunicación entre Vice y Venus.

Figura 8.4. Intercepción de llamadas al sistema en AFS [Coulouris *et al.*, 2012]



8.7 MEMORIA COMPARTIDA DISTRIBUIDA

8.7.1 Generalidades

La *memoria compartida distribuida (DSM)* fue propuesto por Li [1986] como una solución en la cual un conjunto de máquinas conectadas en red comparten un solo espacio de direcciones virtuales con sus páginas. El desempeño es pobre, debido a la latencia y tráfico de la red como resultado de la localización de las páginas en la red.

Ejemplos de tipos de memoria compartida son:

- *Memoria en circuitos*: Varios procesadores comparten una memoria en un circuito.
- *Memoria usando un bus con o sin caché*: Diferentes CPU comparten una memoria que usa o no caché por medio de un bus.

- *Multiprocesadores basados en anillo*: Varios CPU comparten una parte de su espacio de direcciones usando un anillo.
- *Múltiples CPU con conmutador*: Permite incrementar la capacidad de comunicación variando la topología cuando existe una gran cantidad de procesadores.
- NUMA: Este acceso no uniforma a memoria, permite resaltar las diferencias entre un acceso local a uno remoto no ocultándolas con el uso de caché.
- *Basadas en páginas*: Aquí cada CPU tiene una memoria privada y no pueden realizar llamadas directas a memorias remotas.
- *Basadas en objeto*: Los programas deben de recorrer métodos protegidos para poder usar datos compartidos. Todo esto se realiza en software y permite mantener la consistencia.

8.7.2 Consistencia en la DSM

Adve y Hill [1990] definieron al modelo de consistencia como las reglas negociadas entre el software y la memoria compartida distribuida. Ejemplos de modelos de consistencia usados en la DSM son [Tanenbaum, 1996]:

- *Consistencia estricta*: Aquí toda lectura a una localidad de memoria *x* devolverá el valor registrado por la última escritura en *x*.
- *Consistencia secuencial*: Define que el resultado de cualquier ejecución es igual que si las operaciones de todos los procesos fueran realizadas de manera secuencial y las operaciones de cada proceso aparecen en esta secuencia en el orden indicado en el programa.
- *Consistencia casual*: Aquí todas las escrituras potenciales relacionadas en forma casual son vistas en ese orden por todos los procesos. Escrituras concurrentes podrán ser vistas en diferente orden desde diferentes máquinas.
- *Consistencia PRAM*: Define que las escrituras de un proceso se reciben por otros procesos en el orden realizado, sin embargo, las escrituras de procesos diferentes se pueden ver en orden diferentes desde distintas computadoras.
- *Consistencia débil*: El acceso a las variables de sincronización son de consistencia secuencial, no se permite este acceso hasta que se terminen las escrituras anteriores en todos los sitios ni se permite un acceso a datos sin haber terminado todos los accesos anteriores a las variables de sincronización.

- *Consistencia de liberación*: Se debe de terminar con éxito todas las adquisiciones anteriores antes de realizar un acceso normal a una variable compartida, así como terminar lecturas y escrituras anteriores del proceso antes de liberar, ambas deben ser consistentes.
- *Consistencia de entrada*: Define que un proceso que realiza una adquisición no podrá concluir hasta no actualizar todas las variables compartidas protegidas, para lo que entrará en una región crítica en modo exclusivo.

8.7.3 DSM basada en paginación

En este modelo ningún procesador tiene acceso directo a la memoria de otro procesador, por lo que estos sistemas también se dominan sin acceso a memoria remota (NORMA) [Tanenbaum, 1996]. La idea en este modelo se centra en emular el caché de un multiprocesador a través de un administrador de memoria y el sistema operativo para relacionar los pedazos de memoria. La opción de réplica ayuda a mejorar el desempeño, duplicando los pedazos exclusivos para lectura, por lo que un aspecto importante del diseño es el tamaño de estos pedazos, que puede ser una palabra, un bloque, una página o un segmento.

Un problema a considerar en este modelo es que para transferencias grandes de datos la red tiende a bloquearse por más tiempo, pero también se podría presentarse la compartición de manera falsa. Otro problema se presenta con la consistencia secuencial cuando existen réplicas de página de lectura-escritura para la página local y la página remota. Para este problema de consistencia se presentan dos soluciones:

- *Actualización*: Permite la escritura local y actualiza de manera simultánea todos los cachés.
- *Invalidación*: Aquí se transmite por el bus la dirección de la palabra a actualizar, pero no la nueva palabra, por lo que cuando un caché ve que una de sus palabras está siendo actualizada, invalida el bloque que contiene la palabra, eliminándola del caché.

Por lo general, los sistemas DSM basados en paginación adoptan la solución por invalidación. Para la búsqueda de copias en estos DSM se proponen dos posibilidades:

- Transmitir el número de página solicitando su invalidación.
- Usar una lista de conjunto de páginas.

Una DSM puede requerir reemplazar páginas cuando no existan marcos libres, para lo que podrán usar los algoritmos tradicionales, como LRU (*Least Recently Used*, El más reciente usado). Una página duplicada es por lo general la elegida a retirar de la memoria o una página duplicada de un proceso saliente. En caso de que ninguna página duplicada pueda ser elegida, se podrá retirar la página no duplicada de menor uso. La sincronización es otro punto a observar en estos sistemas, para lo cual se prevé la exclusión mutua apoyada con algún controlador de sincronización.

EJERCICIOS

1. Describe tres funciones del módulo cliente en un sistema de archivos.

2. Describe las funciones de los módulos "Venus" y "Vice" en el sistema AFS.

3. Cita cinco operaciones que son posibles realizar en el sistema NFS.

4. ¿Cuál es la diferencia entre un sistema de archivos y una base de datos?

5. ¿Cuál es el principal beneficio de la memoria distribuida?

6. ¿Cuáles son las principales desventajas del sistema NFS?

7. Cita tres características de UNIX que toma AFS en su estrategia.

8. Indica las principales diferencias entre NFS y AFS.

9. Investiga sobre los sistemas de archivos posteriores a NFS y AFS.

10. ¿Por qué es importante la consistencia en la memoria compartida distribuida?

11. Cita tres ejemplos de uso de memoria compartida.

Actividad integradora

1. De la aplicación de los contenidos, así como de las habilidades y actitudes desarrolladas.
   Se recomienda exposición general del tema por parte del profesor, así como la realización de algunas prácticas sobre operaciones en archivos distribuidos por parte de los alumnos en el laboratorio. Se puede usar para tal fin el sistema NFS que tienen los sistemas operativos Linux o Unix. El contenido de este capítulo podría ser cubierto con una sesión teórica y dos sesiones prácticas en el laboratorio, realizando manipulación de archivos en un ambiente distribuido.
   El alumno debe mostrar un gran interés por programar y explorar el uso de comandos preexistentes en un sistema de archivos distribuidos.
2. Del logro de los objetivos establecidos en el capítulo o apartado del material.
   Se espera alcanzar los objetivos de este capítulo con el apoyo en la solución de los ejercicios y la exposición del profesor sobre el tema, complementadas con las prácticas de laboratorio correspondiente.