SISTEMAS OPERATIVOS DISTRIBUIDOS

JOSE FERNANDO BASTIDAS C.
HERBERT CARRILLO G.
DAVID EDUARDO CIFUENTES C.
MARIA FERNANDA SERRANO B.
ANDRES VARGAS CABAS

Alumnos del curso de Investigación de VIII Semestre de Ingeniería de Sistemas del ICESI

OBJETIVOS

- Dar un concepto claro de lo que es un Sistema Distribuido, tocando algunos temas técnicos importantes, pero explicándolos de manera que lleguen fácilmente a las personas.
- Dar a conocer la importancia de apoyar la investigación en este campo, que apenas está surgiendo, pero que promete mucho para el desarrollo tecnológico mundial.
- Conocer la manera como cooperan y se sincronizan los procesos entre sí, en un Sistema Distribuido, así como la forma de implantar regiones críticas o asignar recursos.
- Conocer las características y componentes de un Sistema Distribuido de archivos.
- Exponer los puntos fundamentales en la implantación de un Sistema Distribuido de archivos.
- Lograr captar el interés de todas aquellas personas interesadas en consultar este libro, específicamente de quienes se encuentran de una u otra forma vinculados con el análisis y desarrollo del software, para que empiecen a interesarse en la creación de aplicaciones enfocadas hacia la distribución, las cuales deben conducir a que, en un futuro no muy lejano, podamos hablar de comunicaciones a nivel mundial, realmente confiables, transparentes y eficientes.

INTRODUCCION

Desde hace algunos años el uso de los computadores ha cambiado enormemente, debido a la transición, desde los sistemas centralizados que constan de un único CPU, sus periféricos de memoria y algunas terminales, hacia sistemas de cómputo compuestos por un gran número de CPUs, conectados mediante una red de alta velocidad, conocidos como *Sistemas Distribuidos*. Aunque los sistemas operativos necesarios

para estos *Sistemas Distribuidos* están apenas en etapa de surgimiento, ya es bastante lo que conocemos acerca de ellos.

Nuestro objetivo es presentar a ustedes, de manera clara y sencilla, los conceptos básicos acerca de los *Sistemas Distribuidos*, sus principales características, las ventajas y desventajas, el manejo de la comunicación y la sincronización dentro de estos sistemas, los sistemas distribuidos de archivos y un estudio de los procesos y procesadores de los *Sistemas Distribuidos*.

Esperamos que el lector comprenda la filosofía de los *Sistemas Distribuidos*, después de haber leído nuestra obra, y pueda llenar, si no todas, muchas de las expectativas que tenga acerca de este novedoso campo de los sistemas de computación.

1. INTRODUCCION A LOS SISTEMAS DISTRIBUIDOS

1.1. Historia

El uso de los computadores está a punto de sufrir una revolución. Desde 1945, cuando comenzó la era del computador moderno, hasta 1985, los computadores eran grandes y caros, incluyendo los minicomputadores. Como resultado, las organizaciones tenían un grupo pequeño de computadores que operaban independientemente, por carecer de una forma para conectarlos.

A partir de 1985 comenzó a cambiar esta situación. Primero, con el desarrollo de poderosos microprocesadores, se disponía de máquinas de 8 bits, pero pronto se volvieron comunes las CPU de 16, 32 e incluso 64 bits, muchas de las cuales tenían el poder de cómputo de un "mainframe" respetable; y segundo, con la invención de redes locales de alta velocidad (LAN), las cuales permitieron conectar docenas e incluso cientos de máquinas, de tal forma que se pudiesen transferir pequeñas cantidades de información, entre ellas, durante un milisegundo o un tiempo parecido. La mayor cantidad de datos se puede desplazar, entre las máquinas, a razón de 10 millones de bits/segundo y aun más.

Hoy en día es fácil reunir sistemas de cómputo, compuestos por un gran número de CPUs, conectados mediante una red de alta velocidad, los cuales reciben el nombre genérico de *Sistemas Distribuidos*, en contraste con los Sistemas Centralizados que constan de un único CPU, sus periféricos de memoria y algunas terminales.

Los Sistemas Distribuidos necesitan un software radicalmente distinto del de los sistemas centralizados. En particular, aunque los sistemas operativos, necesarios para estos Sistemas Distribuidos, están apenas en una etapa de surgimiento, ya se sabe bastante acerca de ellos.

1.2. Ventajas y desventajas de los Sistemas Distribuidos

1.2.1. *Ventajas de los Sistemas Distribuidos respecto de los Centralizados*

La fuerza real, detrás de la tendencia hacia la descentralización, es la economía. Por unos cuantos cientos de dólares es posible comprar un CHIP de CPU, que puede ejecutar más instrucciones, por segundo, de las que realizaba uno de los más grandes "mainframes" de los años ochenta. Si uno está dispuesto a pagar el doble, se obtiene el mismo CPU, sólo que con una velocidad un poco mayor. Como resultado, la solución más eficaz, en cuanto a costos, es limitarse a un gran número de CPUs baratos, reunidos en un mismo sistema. Así, la razón número uno de la tendencia hacia los Sistemas Distribuidos es que éstos tienen en potencia una proporción precio/desempeño mucho mejor que la de un único sistema centralizado.

Una ligera variación, con respeto a este tema, es la observación de que una colección de microprocesadores no sólo facilita una mejor proporción precio/desempeño que un único "mainframe", sino que puede producir un mejor rendimiento del que podría proporcionar cualquier "mainframe" a cualquier precio. Por ejemplo, con la tecnología actual es posible construir un sistema a partir de los 1.000 chips de un computador moderno, cada uno de los cuales tiene una ejecución de 20 MIPS (millones de instrucciones por segundo), para un rendimiento total de 20.000 MIPS. Para que un único procesador (es decir, CPU) logre esto, tendría que ejecutar una instrucción en 0,05 nanosegundos (50 picosegundos). Ninguna máquina existente llega a acercarse a esta cifra, además de que consideraciones teóricas y de ingeniería lo consideran improbable durante algún tiempo. Así, si el objetivo es un rendimiento normal a bajo costo, o un alto rendimiento con un mayor costo, los Sistemas Distribuidos tienen mucho que ofrecer.

Otra razón para la construcción de un Sistema Distribuido es que ciertas aplicaciones son dispuestas en forma inherente. En un sistema de automatización de una fábrica, que controle los robots y las máquinas, a lo largo de una línea de ensamblaje, con frecuencia tiene sentido darle a cada robot o máquina su propio computador para que lo controle. Al conectarse éstos, se tiene un Sistema Distribuido Industrial.

Otra ventaja potencial de un Sistema Distribuido sobre uno Centralizado es una mayor confiabilidad. Al distribuir la carga de trabajo en muchas máquinas, la falla de un chip descompondrá, a lo sumo, una máquina, pero el resto seguirá intacto. Para el caso de aplicaciones críticas, como el control de los reactores nucleares o de los aviones, el uso de un sistema distribuido, para lograr una mayor confiabilidad, puede ser el factor dominante.

Por último, el crecimiento por incrementos también es una ventaja potencial. Con frecuencia ocurre que una compañía compra un "mainframe", con la intención de hacer todo su trabajo en él. Si la compañía prospera y la carga de trabajo aumenta, el "mainframe" no será adecuado en cierto momento. Las ideas de reemplazar el "mainframe" por otro más grande (si existe) o añadir un segundo "mainframe", pueden dar un tremendo castigo a las operaciones de la compañía. Por el contrario, con un sistema distribuido, podrían añadirse simplemente más procesadores al sistema, lo que permitiría un desarrollo gradual, conforme surjan las necesidades.

ECONOMIA	**Los microprocesadores ofrecen una mejor proporción precio-rendimiento que los mainframes.**
VELOCIDAD	**Un sistema distribuido puede tener un mayor poder de cómputo que un mainframe.**
DISTRIBUCION INHERENTE	**Algunas aplicaciones utilizan máquinas que están separadas una cierta distancia.**
CONFIABILIDAD	**Si una máquina se descompone, sobrevive el sistema como un todo.**
CRECIMIENTO POR INCREMENTOS	**Se puede añadir poder de cómputo en pequeños incrementos.**

Ventajas de los Sistemas Distribuidos con respecto de los Centralizados.

1.2.2. *Ventajas de los Sistemas Distribuidos respecto de los PC independientes.*

Muchos usuarios necesitan compartir ciertos datos. Por ejemplo, los empleados de reservaciones, en las líneas aéreas, necesitan tener acceso a la base maestra de datos de los vuelos y reservaciones existentes. Si se le diera a cada empleado una copia particular de toda la base de datos, eso no funcionaría, puesto que nadie conocería los asientos vendidos por los demás empleados. Los datos compartidos son absolutamente esenciales para ésta y otras aplicaciones, de modo que las máquinas deben estar conectadas entre sí. Los datos no son los únicos elementos que se pueden compartir; otros candidatos son también los periféricos caros, como las impresoras láser de color, los equipos de fotocomposición y los dispositivos de almacenamiento masivo (por ejemplo, las cajas ópticas).

Una tercera razón, para la conexión de un grupo de computadores aislados en un sistema distribuido, es lograr una mayor comunicación entre las personas, en donde, según la gente, el correo electrónico tiene numerosos atractivos respecto del correo con cartas, el teléfono o el fax.

Por último, un sistema distribuido tiene una mayor flexibilidad potencial que el hecho de darle a cada usuario un computador personal aislado. No sólo existe el método de conectar un grupo de computadores personales, mediante una LAN, sino también tener una mezcla de computadores personales y compartidos, tal vez con distintos tamaños, y dejar que los trabajos se ejecuten de la forma más adecuada, en vez de ejecutarlos siempre en el computador del propietario. De esta manera, la carga de trabajo se puede difundir entre los computadores, de forma más eficaz, y la pérdida de unas cuantas máquinas se puede compensar, si se permite a las personas que ejecuten sus trabajos en otra parte.

DATOS COMPARTIDOS	**Permite que varios usuarios tengan acceso a una base de datos común.**
DISPOSITIVOS COMPARTIDOS	**Permiten que varios usuarios compartan periféricos caros, como las impresoras de color.**
COMUNICACION	**Facilita la comunicación de persona a persona; por ejemplo, mediante el correo electrónico.**
FLEXIBILIDAD	**Difunde la carga de trabajo entre las máquinas disponibles, de la forma más eficaz, en cuanto a los costos.**

Ventajas de los Sistemas Distribuidos respecto de los PC

1.2.3. *Desventajas de los Sistemas Distribuidos*

El peor de los problemas de los sistemas distribuidos es el software, debido a que no se tiene mucha experiencia en el diseño, implantación y uso del software distribuido.

Un segundo problema potencial es el ocasionado por las redes de comunicación, ya que se pueden perder mensajes, por lo cual se requiere de un software especial para su manejo y éste puede verse sobrecargado. Al saturarse la red, ésta debe reemplazarse o añadirse una segunda. En ambos casos, hay que tender cables en uno o más edificios, con un gran costo; o bien, hay que reemplazar las tarjetas de interfaz de la red. Una vez que el sistema llega a depender de la red, la pérdida o saturación de ésta puede desvirtuar algunas de las ventajas que el sistema distribuido tenía.

Por último, la seguridad es, con frecuencia, un problema, debido a que si las personas pueden tener acceso a los datos en todo el sistema, entonces también pueden tener acceso a datos con los que no tienen que ver; por lo tanto, es preferible tener un computador personal aislado, sin conexiones de red con las demás máquinas, para que los datos se mantengan en secreto.

A pesar de estos problemas potenciales, se puede ver que las ventajas tienen mayor peso que las desventajas y, de hecho, es probable que en unos cuantos años muchas organizaciones conecten la mayoría de sus computadores a grandes sistemas distribuidos, para proporcionar un servicio mejor, más barato y más conveniente para sus usuarios.

SOFTWARE	**Existe poco software para los sistemas distribuidos en la actualidad.**
REDES	**La red se puede saturar o causar otros problemas.**
SEGURIDAD	**Un acceso sencillo también se aplica a datos secretos.**

Desventajas de los Sistemas Distribuidos.

1.3. **Conceptos de hardware**

Aunque todos los sistemas distribuidos constan de varios CPUs, existen diversas formas de organizar el hardware; en particular, en la forma de interconectarse y comunicarse entre sí. Se han propuesto diversos esquemas, pero ninguno de ellos ha tenido un éxito completo.

Es probable que la taxonomía más citada sea la de Flynn (1972), aunque es algo rudimentaria. Flynn eligió dos características, consideradas por él como esenciales: el número de flujos de instrucciones y el número de flujos de datos.

Un computador, con un solo flujo de instrucciones y un solo flujo de datos, se llama SISD ("Single Instruction Single Data"). Todos los computadores de un solo procesador caen dentro de esta categoría.

La siguiente categoría es SIMD ("Single Instruction Multiple Data"), con un flujo de instrucciones y varios flujos de datos. En este tipo, se ordenan los procesadores con una unidad que busca una instrucción y, después, ordena a varias unidades de datos para que la lleven a cabo, en paralelo, cada una con sus propios datos. Ciertos supercomputadores son SIMD.

La siguiente categoría es MISD (Multiple Instruction Single Data), con flujo de varias instrucciones y un solo flujo de datos. Ninguno de los computadores conocidos se ajusta a este modelo.

Por último viene MIMD (Multiple Instruction Multiple Data), que significa un grupo de computadores independiente, cada uno con su propio contador del programa, programas y datos. Todos los sistemas distribuidos son MIMD.

Los computadores MIMD se pueden dividir en dos grupos: aquellos que tienen memoria compartida, que, por lo general, se llaman ***Multiprocesadores*** y aquellos que no, que a veces reciben el nombre de ***Multicomputadores.*** La diferencia esencial es que un Multiprocesador tiene un solo espacio de direcciones virtuales, compartido por todos los CPU; y en un Multicomputador, cada máquina tiene su propia memoria particular.

Estas dos categorías se pueden subdividir, con base en la arquitectura de la red de interconexión, en ***Bus y con Conmutador.*** En la primera queremos indicar que existe una sola red, plano de base, bus, cable u otro medio que conecta todas las máquinas, como es el caso de la televisión comercial por cable; los sistemas con Conmutador no tienen una sola columna vertebral como la televisión por cable, sino que tienen cables individuales de una máquina a otra y utilizan varios patrones diferentes de cableado. Los mensajes se mueven a través de los cables y se hace una decisión explícita de conmutación en cada etapa, para dirigir el mensaje a lo largo de uno de los cables de salida.

En ciertos sistemas las máquinas están ***Fuertemente Acopladas*** y en otras están ***Débilmente Acopladas;*** en un sistema Fuertemente Acoplado el retraso que se experimenta, al enviar un mensaje de un computador a otro, es corto y la tasa de transmisión de datos, es decir, el número de bits por segundo que se puede transferir es alto; en un sistema Débilmente Acoplado ocurre lo contrario.

Los sistemas fuertemente acoplados tienden a utilizarse más como sistemas paralelos (para trabajar con un solo problema) y los débilmente acoplados tienden a utilizarse como sistemas distribuidos (para trabajar con varios problemas no relacionados entre sí), aunque esto no siempre es cierto.

En general, los multiprocesadores tienden a estar más fuertemente acoplados que los multicomputadores, puesto que pueden intercambiar datos a la velocidad de sus memorias, pero algunos multicomputadores, basados en fibras ópticas, pueden funcionar también con velocidad de memoria.

Aunque el estudio de multiprocesadores y multicomputadores, con base en bus y con conmutador, no afecta directamente el estudio de los sistemas operativos distribuidos, es importante tener un concepto claro acerca de cada uno de ellos.

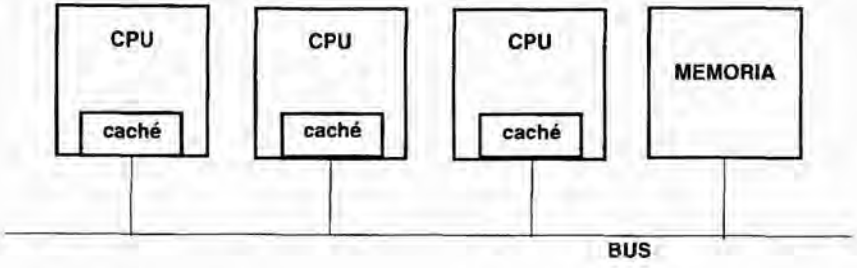
1.3.1. *Multiprocesadores con base en buses*

Los multiprocesadores con base en buses constan de cierto número de CPUs, conectados a un bus común, junto con un módulo de memoria. Una configuración sencilla consta de un plano de base ("backplane") de alta velocidad o tarjeta madre, en la cual se pueden insertar varias tarjetas de memoria y el CPU.

Para leer una palabra de memoria, un CPU coloca la dirección de la palabra deseada en las líneas de dirección del bus y una señal en las líneas adecuadas de control, para indicar que desea leer. La memoria responde y coloca el valor de la palabra en las líneas de datos, para permitir la lectura de ésta por parte del CPU solicitante. La escritura funciona de manera similar.

El problema con este esquema es que si sólo se dispone de 4 ó 5 CPUs; el bus estará, por lo general, sobrecargado y el rendimiento disminuirá en forma drástica. La solución es añadir una ***Memoria Caché*** de alta velocidad entre el CPU y el bus, donde se guardan las palabras de acceso reciente y no hay necesidad de utilizar el bus para ir a buscar, en la memoria, una palabra que

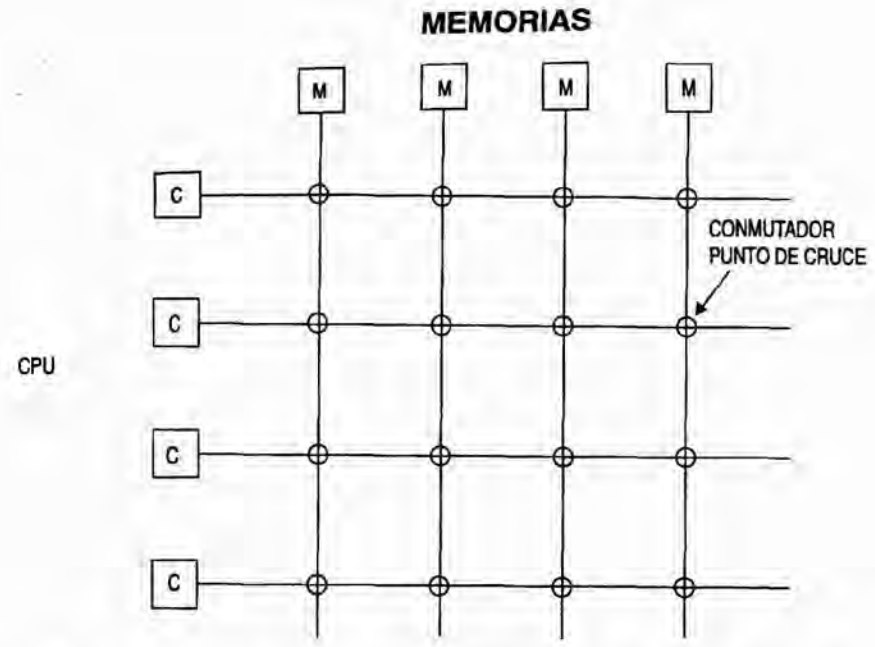
se encuentra en memoria caché. Pero también se presenta un problema de memoria incoherente, en el caso de que dos CPUs lean la misma palabra en sus respectivos cachés y uno de ellos la modifique y el otro lea el valor viejo de la palabra. Los investigadores han solucionado esto con el diseño de un caché que cuando una palabra sea escrita en él, también sea escrita en la memoria; y además, este caché mantiene un monitoreo permanente del bus, para saber cuándo una palabra que está en él ha sido modificada y, por lo tanto, también modificarla en él. Mediante el uso de este caché es posible colocar de 32 a 64 CPUs en el mismo bus.



1.3.2. *Multiprocesadores con conmutador*

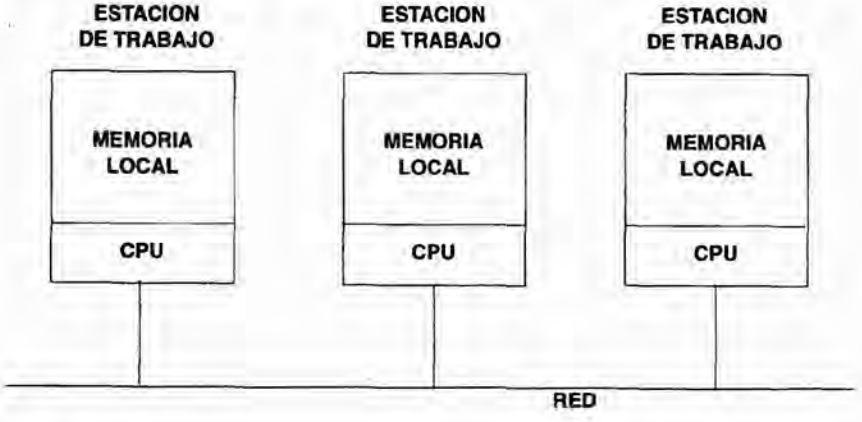
Para construir un multiprocesador con más de 64 procesadores, es necesario un método distinto para conectar cada CPU con la memoria. Una posibilidad es dividir la memoria en módulos y conectarlos a la CPU con un conmutador. Cada CPU y cada memoria tienen una conexión que sale de él. En cada intersección está un delgado ***Conmutador del punto de cruce*** electrónico, que el hardware puede abrir y cerrar. Cuando un CPU desea tener acceso a una memoria particular, el conmutador del punto de cruce que los conecta se cierra de manera momentánea, para que tenga lugar dicho acceso. Esta descripción es del ***Conmutador de Cruceta,*** pero los investigadores han dado a conocer otros tipos de conmutación, como es la ***Red Omega,*** que utiliza mucho menos conmutadores que el conmutador de cruceta y hace la misma función, pero tiene un problema de retraso, pues es muy lenta. Los investigadores diseñaron NUMA (Acceso no uniforme a la memoria), la cual tiene un rápido acceso a su memoria, pero un lento acceso a la memoria de los demás.

En resumen, los multiprocesadores basados en buses, incluso con cachés monitores, quedan limitados como máximo a 64 CPUs, por la capacidad del bus. Para rebasar estos límites es necesario una red de conmutador, como un conmutador de cruceta, una red omega o algo similar. Los grandes conmutadores de cruceta son muy caros y las grandes redes omega son caras y lentas. Las máquinas NUMA necesitan complejos algoritmos para un buen software de colocación. La conclusión es clara: La construcción de un multiprocesador grande, fuertemente acoplado y con memoria compartida, es difícil y cara.



1.3.3. *Multicomputadores con base en buses*

La construcción de un multicomputador es fácil. Cada CPU tiene una conexión directa con su propia memoria local. El único problema consiste en definir la forma en que los CPUs se comunican entre sí. Es claro que aquí también se necesita cierto esquema de interconexión, pero sólo es para la comunicación entre un CPU y otro. El volumen del tráfico será menor, varios órdenes, si se utiliza la red de interconexión para el tráfico CPU-Memoria; el sistema de interconexión puede tener una LAN de menor velocidad, en comparación con un "backplane".



1.3.4. *Multicomputadores con conmutador*

Se han propuesto y construido varias redes de interconexión, pero todas tienen la propiedad de que cada CPU tiene acceso directo y exclusivo a su propia memoria particular. Existen dos topologías populares, una ***Retícula*** y un ***Hipercubo.*** Las retículas son fáciles de comprender y se basan en las tarjetas de circuitos impresos. Se adecuan mejor a problemas, con una naturaleza bidimensional inherente, como la teoría de gráficos o la visión.

Un hipercubo es un cubo-n-dimensional. Cada vértice es un CPU. Cada arista es una conexión entre dos CPUs. Cada CPU tiene "n" conexiones con otros CPUs. Así, la complejidad del cableado aumenta en proporción logarítmica con el tamaño. Puesto que sólo se conectan los vecinos más cercanos, muchos mensajes deben realizar varios saltos, antes de llegar a su destino. Los hipercubos con 1.024 CPUs están disponibles en el mercado desde hace varios años y ya comienzan a estar disponibles los hipercubos, con hasta 16.384 CPUs.

1.4. Conceptos de software

Aunque el hardware es importante, el software lo es aún más. La imagen que un sistema presenta a sus usuarios y la forma de pensar de éstos, acerca del sistema, queda determinada, en gran medida, por el software del sistema operativo, no por el hardware.

Los sistemas operativos no se pueden colocar fácilmente dentro de unos estándares como el hardware. Por su propia naturaleza, el software es vago y amorfo. Aún así, es más o menos posible distinguir dos tipos de sistemas operativos para los sistemas de varios CPUs: ***Los Débilmente Acoplados y Los Fuertemente Acoplados.***

El software débilmente acoplado permite que las máquinas y usuarios de un sistema distribuido sean independientes entre sí, en lo fundamental, pero que interactúen, en cierto grado, cuando sea necesario. Consideremos un grupo de computadores personales, cada uno de los cuales tiene su propio CPU, su propia memoria, su propio disco duro y su propio sistema operativo, pero que comparten ciertos recursos, tales como las impresoras láser y las bases de datos, en una LAN. Este sistema está débilmente acoplado, puesto que las máquinas individuales se distinguen con claridad, cada una de las cuales tiene su propio trabajo por realizar. Si la red falla por alguna razón, las máquinas individuales continúan su ejecución en cierto grado considerable, aunque se puede perder cierta funcionalidad.

En el otro extremo, podríamos tener el caso de un multiprocesador dedicado a la ejecución de un único programa de ajedrez, en paralelo. A cada CPU se le asigna un tablero para su evaluación y éste ocupa su tiempo, en la evaluación de este tablero y de los tableros que se pueden generar a partir de él. Al terminar la evaluación, el CPU informa de sus resultados y se le proporciona un nuevo tablero para trabajar con él. El software para este sistema, es decir, el programa de aplicación y el sistema operativo necesario para soportarlo, están más fuertemente acoplados que en el caso anterior.

Con lo que hemos tratado hasta ahora, podemos ver que existen cuatro tipos de hardware distribuido y dos tipos de software distribuido. En teoría, deberían existir ocho combinaciones de hardware y software. Pero sólo existen cuatro, puesto que para el usuario la interconexión de la tecnología no es visible. Un multiprocesador es un multiprocesador, sin importar si utiliza un bus con cachés monitores o una red omega.

1.4.1. *Sistemas Operativos de Redes y NFS*

La combinación más común, en muchas organizaciones, es un software débilmente acoplado en un hardware igual. Un ejemplo típico es una red de estaciones de trabajo de ingeniería, conectadas mediante una LAN. En este modelo, cada usuario tiene una estación de trabajo para su uso exclusivo, lo mismo que su propio sistema operativo, en donde lo normal es que todos los comandos se ejecuten en forma local, precisamente en la estación de trabajo.

Sin embargo, a veces es posible que un usuario se conecte de manera remota con otra estación de trabajo, mediante un comando.

El efecto de este comando es convertir la propia estación de trabajo del usuario en una terminal remota, enlazada con la máquina remota. Los comandos escritos en el teclado se envían a la máquina remota y la salida de ésta se exhibe en la pantalla. En cualquier instante, sólo se puede utilizar una máquina y la selección de ésta se realiza en forma manual.

Las redes de las estaciones de trabajo también tienen un comando de copiado remoto, para copiar archivos de una máquina a otra, pero el usuario debe estar consciente de la posición de los archivos.

Aunque es mejor que nada, esta forma de comunicación es primitiva en extremo. Un método más conveniente de comunicación y distribución de la información consiste en proporcionar un sistema de archivos global compartido, accesible desde todas las estaciones de trabajo. Una o varias máquinas, llamadas ***Servidores de Archivos,*** soportan el sistema de archivos. Los servidores aceptan solicitudes de los programas de usuarios, los cuales se ejecutan en las otras máquinas (No servidoras), llamadas ***Clientes,*** para la lectura y escritura de archivos. Cada una de las solicitudes que llegue se examina, se ejecuta y la respuesta se envía de regreso.

Los servidores de archivos tienen, por lo general, un sistema jerárquico de archivos, cada uno de los cuales tiene un directorio raíz, con sus directorios y archivos. Las estaciones de trabajo pueden importar o montar estos sistemas de archivos, lo que aumenta sus sistemas locales de archivos con aquellos localizados en los servidores.

El sistema operativo por utilizar, en este tipo de ambiente, debe controlar tanto las estaciones de trabajo, en lo individual, como a los servidores de archivos, y también debe encargarse de la comunicación entre ellos. Es posible que todas las máquinas ejecuten el mismo sistema operativo, pero esto no es necesario. Si los clientes y los servidores ejecutan diversos sistemas, entonces, como mínimo, deben coincidir en el formato y significado de todos los mensajes que podrían intercambiar. En una situación como ésta, en la que cada máquina tiene un alto grado de autonomía y existen pocos requisitos a lo largo de todo el sistema, las personas se refieren a ella como un ***Sistema Operativo de Red.***

Uno de los mejores sistemas operativos de red es el ***Network File System,*** de *Sun Microsystems,* conocida en general como *NFS.* Es un sistema con amplio uso. *Sun Microsystems* diseñó e implantó *NFS,* en un principio para su uso en estaciones de trabajo con base en UNIX. Ahora lo soportan otros fabricantes tanto para Unix como para otros sistemas operativos (como MS-DOS). NFS soporta sistemas heterogéneos, por ejemplo, clientes de MS-DOS que hacían uso de servidores Unix. Ni siquiera se pide que las máquinas utilicen el mismo hardware. Es común encontrarse con clientes de MS-DOS que utilizan CPU de Intel 386 y obtienen servicios de servidores de archivo en Unix, con CPU de Motorola 68030 o SunSparc.

Existen tres aspectos importantes de NFS: la Arquitectura, el Protocolo y la Implantación.

- ***La Arquitectura de NFS:*** La idea fundamental de NFS es permitir que una colección arbitraria de clientes y servidores compartan un sistema de archivos común. En la mayoría de los casos, todos los clientes y servidores están en la misma LAN, pero esto no es necesario. Es posible ejecutar el NFS en una WAN. Tomamos a los clientes y servidores como máquinas distintas, pero NFS también permite que cada máquina sea un cliente y un servidor al mismo tiempo.

- La característica básica de la arquitectura de NFS es que los servidores exportan directorios y los clientes los montan de manera remota. Si dos o más clientes montan el mismo directorio al mismo tiempo, ellos se pueden comunicar y compartir archivos en sus directorios comunes. Un programa para un cliente puede crear un archivo y un programa para otro cliente distinto, puede leer dicho archivo. Una vez llevados a cabo los montajes, no hay que hacer nada especial para lograr compartir los archivos. Los archivos compartidos están ahí, en la jerarquía de directorios de varias máquinas, y se puede leer o escribir en ellos de la manera usual. Esta sencillez es uno de los grandes atractivos de NFS.

- ***Protocolos de NFS:*** Puesto que uno de los objetivos de NFS es soportar un sistema heterogéneo, en donde los clientes y servidores podrían ejecutar distintos sistemas operativos en hardware diverso, es esencial que la interfaz entre los clientes y servidores esté bien definida. NFS logra este objetivo mediante la definición de dos protocolos cliente-servidor. Un ***Protocolo*** es un conjunto de solicitudes que envían los clientes a los servidores, junto con las respuestas correspondientes, enviadas por los servidores de regreso a los clientes. Mientras un servidor reconozca y pueda manejar todas las solicitudes en los protocolos, no necesita saber algo de sus clientes. En forma análoga, los clientes consideran a los servidores como "cajas negras". El primer protocolo de NFS maneja el montaje. Un cliente puede enviar el nombre de una ruta de acceso a un servidor y solicitar el permiso para montar ese directorio en alguna parte de su jerarquía de directorios. El segundo protocolo de NFS es para el acceso a los directorios y archivos. Los clientes pueden enviar mensajes a los servidores para el manejo de los directorios y la lectura o escritura de archivos. Además, también pueden tener acceso a los atributos de archivo, tales como su modo, tamaño y fecha de la última modificación.

- **Implantación de NFS:** La implantación del código del cliente y del servidor es independiente de los protocolos. Consta de tres capas: la capa superior es la capa de llamadas al sistema; ésta maneja las llamadas del tipo OPEN, READ y CLOSE. Después de analizar la llamada y verificar los parámetros, llama a la segunda capa, la capa del sistema virtual de archivos (VFS). La tarea de la capa VFS es mantener una tabla, con una entrada por cada archivo abierto, análoga a la tabla de ínodos, para los archivos abiertos en Unix. Después de descubrir el nombre de la máquina donde se localiza el directorio por montar, se relaciona esa máquina buscando un "file handle" para el directorio remoto. Si el directorio existe y está disponible para su montaje remoto, el servidor regresa, entonces, un "file handle" para el directorio. Por último, se transfiere el "file handle" al núcleo.

1.4.2. *Sistemas realmente distribuidos*

El siguiente paso, en la evolución, es el de un software fuertemente acoplado en un hardware débilmente acoplado (es decir, multicomputadores). El objetivo de un sistema de este tipo es crear la ilusión, en las mentes de los usuarios, de que toda la red de computadores es un solo sistema de tiempo compartido, en vez de una colección de máquinas diversas. Esto nos lleva a nuestra definición de sistema distribuido: *un Sistema Distribuido es aquel que se ejecuta en una colección de máquinas sin memoria compartida, pero que aparece ante sus usuarios como un solo computador.*

Para que un sistema sea realmente distribuido, debe existir un mecanismo de comunicación global entre los procesos, de forma que cualquier proceso puede hablar con cualquier otro; además, no tiene que haber distintos mecanismos en distintas máquinas o distintos mecanismos para la comunicación local o la comunicación remota; también debe existir un esquema global de protección.

1.4.3. *Sistemas de multiprocesador con tiempo compartido*

La última combinación que queremos analizar es la de un software fuertemente acoplado en un hardware igual. Aunque existen varias máquinas de propósito especial en esta categoría (como las máquinas dedicadas a las bases de datos), los ejemplos más comunes de propósito general son los multiprocesadores, como los fabricados por Sequent y Encore, que operan como un sistema de tiempo compartido de Unix, sólo que con varios CPU en lugar de uno solo.

La característica clave de este tipo de sistema es la existencia de una sola cola para la ejecución: una lista de todos los procesos, en el sistema, que no están bloqueados en forma lógica y listos para su ejecución. La cola de ejecución es una estructura de datos contenida en la memoria compartida.

1.5 *Aspectos del diseño*

1.5.1. Transparencia

Un sistema es transparente cuando hace que las personas piensen que la colección de máquinas es tan sólo un sistema de tiempo compartido de un solo procesador, es decir, cuando se le oculta la distribución del sistema y piensan que un solo computador es el que hace todo el trabajo. Existen cinco tipos de transparencia:

TRANSPARENCIA DE LOCALIZACION.	**Los usuarios no pueden indicar la localización de los recursos.**
TRANSPARENCIA DE MIGRACION	**Los recursos se pueden mover a voluntad, sin cambiar sus nombres.**

TRANSPARENCIA DE REPLICA.	**Los usuarios no pueden indicar el número de copias existentes.**
TRANSPARENCIA DE CONCURRENCIA.	**Varios usuarios pueden compartir recursos de manera automática.**
TRANSPARENCIA DE PARALELISMO.	**Las actividades pueden ocurrir en paralelo, sin el conocimiento de los usuarios.**

1.5.2. *Flexibilidad*

Es importante que el sistema sea flexible, ya que apenas estamos aprendiendo a construir sistemas distribuidos. Es probable que este proceso tenga muchas salidas falsas y una considerable retroalimentación. Las decisiones de diseño que ahora parezcan razonables podrían demostrar ser incorrectas posteriormente.

Existen dos escuelas de pensamiento en cuanto a la estructura de los sistemas distribuidos. Una escuela dice que cada máquina debe ejecutar un núcleo tradicional que proporcione la mayoría de los servicios. la otra sostiene que el núcleo debe proporcionar lo menos posible y que el grueso de los servicios del sistema operativo se obtenga a partir de los servidores al nivel usuario. La primera escuela, conocida como el núcleo monolítico, opera en el actual sistema operativo y centralizado básico, aumentado con las capacidades de la red y la integración de los servicios remotos. La segunda escuela, conocida como el micronúcleo, opera en los sistemas distribuidos diseñados actualmente.

1.5.3. *Confiabilidad*

Uno de los objetivos originales de la construcción de sistemas distribuidos fue el hacerlos más confiables que los sistemas con un único procesador. La idea es que si una máquina falla, alguna otra máquina se encargue del trabajo.

1.5.4. *Desempeño*

La construcción de un sistema distribuido transparente, flexible y confiable no hará que usted gane premios si es muy lento. En particular, cuando se ejecuta una aplicación en un sistema distribuido, no debe parecer peor que su ejecución en un único procesador.

Se pueden utilizar diversas métricas del desempeño. El tiempo de respuesta es uno, pero también lo son el rendimiento (número de trabajos por hora), el uso del sistema y la cantidad consumida de la capacidad de la red. El problema del desempeño se complica por el hecho que la comunicación, factor esencial en un sistema distribuido (y ausente en un sistema con un único procesador), es algo lenta por lo general. Así, para optimizar el desempeño, con frecuencia hay que minimizar el número de mensajes.

1.5.5. *Escalamiento - Extensión*

La mayoría de los sistemas distribuidos está diseñada para trabajar con unos cuantos cientos de CPUs. Es posible que los sistemas futuros tengan mayores órdenes de magnitud y que las soluciones que funcionen bien para 200 máquinas fallen de manera total para 200 millones; por esto, es importante que al construir los sistemas distribuidos se piense en el número de máquinas que en un futuro se le acondicionarán.

2. COMUNICACIONES EN LOS SISTEMAS DISTRIBUIDOS

La diferencia más importante entre un sistema distribuido y un sistema de único procesador es la comunicación entre los procesos. En un sistema con un solo procesador, la mayor parte de la comunicación entre procesos supone, de manera implícita, la existencia de la memoria compartida. Un ejemplo típico es el problema de los lectores escritores, en donde un proceso escribe en un "buffer" compartido y otro proceso lo lee. Incluso, en las formas más básicas de sincronización, como el semáforo, hay que compartir una palabra (La variable del semáforo). En un sistema distribuido no existe tal memoria compartida, por lo que toda la naturaleza de la comunicación entre procesos debe replantearse a partir de cero.

Comenzaremos con el análisis de las reglas a las que se deben apegar los procesos respecto de la comunicación

conocida como protocolo. Para los sistemas distribuidos, estos protocolos toman con frecuencia la forma de varias capas, cada una con sus propias metas y reglas; luego analizaremos con más detalle el modelo cliente-servidor, del cual tanto se ha hablado. Finalmente, veremos la forma en que se intercambian los mensajes y las muchas opciones disponibles para los diseñadores del sistema.

2.1. **Redes y Comunicaciones**

Como vimos en la primera parte, el disponer de una red es uno de los requisitos para poder hablar de un sistema operativo completamente distribuido; por ello, se hace importante estudiar las características de las comunicaciones, en los diferentes tipos de redes, bien sea en las redes del área local LAN o en las redes de área amplia WAN. Estas últimas permitirán, en un futuro, ofrecer la posibilidad de conexión en el mundo, entre computadores y usuarios; por ello se deben acordar protocolos o reglas de comunicación estándar entre los usuarios, e incluso entre los diferentes tipos de redes.

En el exterior existen diferentes organismos, los cuales han tratado de establecer protocolos comunes para seguir en todas las comunicaciones que se puedan llevar a cabo en los diferentes tipos de redes. Entre los organismos más destacados vale la pena mencionar la Organización Internacional para la Internacionalización (ISO), con su modelo de referencia para la interconexión de sistemas abiertos OSI; el Comité Consultivo Internacional para la Telegrafía y las Telecomunicaciones (CCITT) y el Instituto Nacional Americano de Normas (ANSI).

Entre los conjuntos de normas más importantes podemos citar el OSI, que abarca una arquitectura de servicios y protocolos y es comúnmente usado entre las redes de área amplia, y el TCP/IP, utilizado entre los computadores del Departamento de Defensa de los Estados Unidos.

2.2. **Protocolos con capas**

Debido a la ausencia de memoria compartida, todas las comunicaciones en los sistemas distribuidos se basan en la transferencia de mensajes. Cuando el proceso A quiere comunicarse con el proceso B, construye primero un mensaje en su propio espacio de direcciones. Entonces, ejecuta una llamada al sistema para que éste busque el mensaje y lo envíe, a través de la red, hacia B. Como vemos, esto parece sencillo, pero, para evitar confusiones, A y B deben coincidir en el significado de los bits que se envíen. Por ejemplo, si A envía un mensaje, en código ASCII, y B espera un mensaje en el código EBCDIC, la comunicación no será óptima.

Como vemos, se necesitan muchos puntos de común acuerdo para establecer una buena comunicación. ¿Cuántos voltios hay que utilizar para la señal correspondiente a un bit cero y cuántos para un bit uno? ¿Cómo sabe el receptor cuál es el último bit del mensaje? ¿Cómo puede detectar si un mensaje ha sido dañado o perdido, y qué debe hacer si lo descubre? ¿Qué longitud tienen los números, cadenas y otros elementos de datos y cuál es la forma en que están representados? De este ejemplo se puede deducir que se necesitan varios puntos de acuerdo con una amplia gama de niveles, desde los detalles de bajo nivel de los bits hasta los detalles de alto nivel, acerca de la forma como debe expresarse la información.

2.2.1. ***Modelo de referencia para la interconexión de sistemas abiertos OSI***

Este modelo está diseñado para permitir la comunicación entre los sistemas abiertos. Un sistema abierto es aquel preparado para comunicarse con cualquier otro sistema abierto, mediante reglas estándar que gobiernan el formato, el contenido y el significado de los mensajes recibidos. Estas reglas se formalizan en lo que se llaman *protocolos*. En términos básicos, un protocolo es un acuerdo de la forma en que debe desarrollarse la comunicación. Cuando se presentan una mujer y un hombre, ella puede optar por extender la mano. El, a su vez, puede decidir si estrechársela o besársela según si, por ejemplo, ella es una abogada en una junta de negocios o una princesa europea en un baile formal. La violación del protocolo hará que la comunicación sea más difícil o, tal vez, imposible.

El modelo OSI distingue entre dos tipos generales de protocolos: los orientados hacia las conexiones y los que no tienen ninguna conexión; los primeros, antes de intercambiar los datos entre el emisor y el receptor, negocian el protocolo por utilizar, mientras que los segundos no necesitan ninguna configuración previa para empezar a transmitir el mensaje. Un ejemplo de comunicación sin conexión podría ser el depositar una carta en un buzón, ya que el receptor no espera que le llegue la carta que en ese momento ha sido depositada y mucho menos ha podido establecer una conexión previa con el emisor para negociar las normas que se deberían utilizar en el envío de la carta.

En el modelo OSI, la comunicación se divide hasta en siete niveles o capas, como se muestra en la Figura 2.1. Cada capa se encarga de un aspecto específico de la comunicación; de esta forma, el problema se puede dividir en piezas manejables, cada una de las cuales se puede resolver en forma independiente de las demás. Cada capa proporciona una interfaz con la capa siguiente. La interfaz consiste en un conjunto de operaciones, para definir el servicio que la capa está preparada a ofrecer a sus usuarios.

2.2.2. ***Capas del modelo OSI***

El modelo OSI cuenta con siete capas, una serie de interfaces y siete protocolos, los que garantizarán que la comunicación entre el emisor y el receptor sea eficiente y confiable, como se muestra en la Figura 2.1. Las siete capas del modelo OSI y su utilidad se describirán a continuación:

- La capa física: Se preocupa por la transmisión de los ceros y los unos. El número de voltios por utilizar (0 y 1), el número de bits por segundo que se pueden enviar, el hecho que la transmisión se lleve a cabo en ambas direcciones, en forma simultánea, son todos aspectos claves en la capa física. Además, el tamaño y forma del conector en la red, así como el número de PINS y el significado de cada uno son temas de interés para dicha capa. El protocolo de la capa física controlará que, cuando una máquina envíe el bit 0, sea realmente recibido como un bit 0 y no como un bit 1.

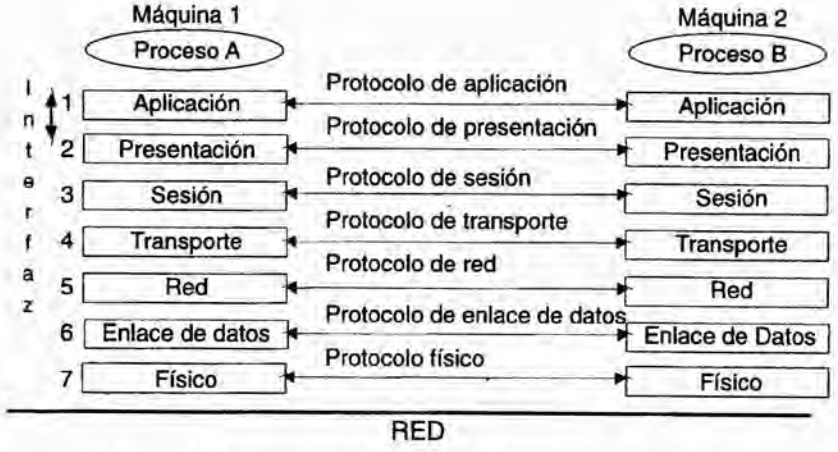


Figura 2.1. *Capas, interfaces y protocolos en el modelo OSI.*

- La capa de enlace de los datos: Esta capa no sólo detecta los errores sino que los corrige. Lo que hace es agrupar los bits en unidades, que a veces se llaman marcos, y revisar que cada marco se reciba en forma correcta; los marcos están conformados por tres campos: un bit al inicio, un bit al final y otro de verificación, conocido como CKECKSUM; este último contendrá la suma de todos los bits del marco, así, en el momento en que el mensaje llegue al receptor, éste sólo tendrá que volver a calcular la suma de los bits del marco y compararla con el contenido del CKECKSUM, para determinar si la información llegó correctamente; en caso contrario, el receptor pedirá al emisor la retransmisión de los datos.
- La capa de la red: Esta capa se encarga de enrutar los datos del emisor, para que lleguen por el camino más corto posible al receptor. Existen dos protocolos comúnmente usados en las capas de red: el orientado a las conexiones, denominado X.25, utilizado con frecuencia por las compañías telefónicas, y el orientado a la no conexión, llamado IP de protocolo internet, el cual es usado por el Departamento de Defensa de los Estados Unidos.
- La capa de transporte: Esta capa se encarga de recibir el mensaje proveniente de la capa de sesión, lo parte en pequeños pedazos, de forma que cada una se ajuste a un único paquete; le asigna a cada uno un número secuencial y después los envía al receptor. Existen diferentes protocolos de transporte, entre ellos están desde el TPO hasta el TP4, protocolos oficiales de transporte ISO; el TCP, protocolo de transporte del Departamento de Defensa de los Estados Unidos; la combinación TCP/IP, protocolo de transporte de la mayoría de los sistemas UNIX, y el UDP, utilizado por los programas de usuario que no necesitan las conexiones.
- La capa de sesión: Esta capa es una versión mejorada de la capa de transporte. Proporciona el control de diálogos, con el fin de mantener un registro de la parte que está hablando en cierto momento y proporciona facilidades en la sincronización. En la práctica, pocas aplicaciones están interesadas en la capa de sesión y rara vez se le soporta.
- La capa de presentación: Esta capa se encarga de dar un formato al mensaje del emisor, para hacerlo entendible por el receptor; dicho formato será una especie de registro, con campos de tamaño adecuado para albergar datos como nombres, direcciones, etc.
- La capa de aplicación: Esta capa se encarga de establecer protocolos para actividades comunes entre terminales remotas; los más conocidos son el protocolo de correo electrónico X.400 y el servidor de directorios X.500.

Para entender mejor el porqué de la existencia de las diferentes capas, en el modelo de comunicaciones OSI, veamos un ejemplo:

Consideremos la comunicación entre dos compañías, *Zyppy Airlines* y su proveedor, *Mushy Meals. Inc.* Cada mes, el jefe de servicios de pasajeros, en *Zippy,* le pide a su secretaria que haga contacto con la secretaria del gerente de ventas en *Mushy,* para ordenar cien mil cajas de pollo. Por lo general, las órdenes se solicitan por medio de la oficina postal. Sin embargo, como el servicio postal es malo, en cierto momento las dos secretarias deciden abandonarlo y comunicarse por fax. Lo pueden hacer sin molestar a sus jefes, puesto que su protocolo se refiere a la transmisión física de las órdenes y a su contenido. En forma análoga, el jefe de servicios de pasajeros decide no pedir el pollo y pedir el nuevo especial de *Mushy,* un filete, sin que esa decisión afecte a las secretarias. Lo que debemos observar es que aquí tenemos dos capas: Los jefes y las secretarias, y cada capa tiene su propio protocolo: temas de discusión y tecnología, que se pueden cambiar independientemente el uno del otro.

De la misma forma, cuando un proceso A de la máquina 1 desea comunicarse con el proceso B de la máquina 2, construye un mensaje y lo transfiere a la capa de aplicación en su máquina; el software de dicha capa agregará un encabezado y hará seguir el mensaje por medio de la interfaz hasta la capa de presentación, en donde se le agregará otro encabezado y así sucesivamente se repetirá el proceso hasta llegar a la capa de enlace de datos, como se muestra en la Figura 2.2. Una vez la máquina 1 tenga el mensaje, con los encabezados correspondientes, lo enviará a la máquina 2, la cual, realizando el proceso inverso, lo interpretará en la forma correcta.

EJEMPLO DE UN MENSAJE ENVIADO DE ACUERDO CON EL MODELO OSI

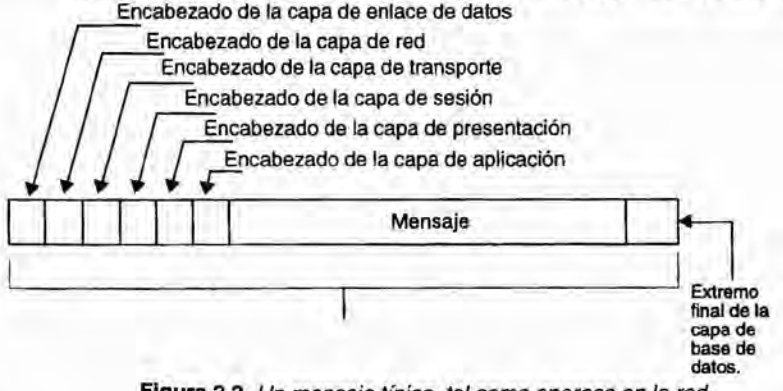


Figura 2.2. *Un mensaje típico, tal como aparece en la red.*

2.3. El modelo cliente-servidor

A primera vista, los protocolos con capas, a lo largo de las líneas OSI, se ven como una forma fina de organizar un sistema distribuido. En efecto, un emisor establece una conexión; es decir, una especie de entubamiento de bits con el receptor y, entonces, empieza a bombear los bits que llegan sin error, en orden, al receptor. Surge entonces la pregunta: ¿qué podría estar mal en este modelo?

A pesar de que el modelo OSI parece ser la forma más eficiente para controlar la comunicación entre el emisor y el receptor, esto sólo se puede afirmar si es utilizado en redes de tipo WAN, ya que, por tratarse de redes en las que los terminales se encuentran distanciados en muchas ocasiones por kilómetros, no importará tener que recurrir a un modelo tan complicado como el OSI, que a pesar de demandar gran cantidad de tiempo en la comunicación, nos brindará la certeza de contar con datos confiables, sin importar las diferencias que puedan existir, tanto en hardware como en el sistema operativo entre el emisor y el receptor, diferencias que podrán ser sorteadas fácilmente gracias a sus siete capas de control.

Para las redes de tipo LAND no es aconsejable utilizar este modelo, ya que por tratarse de comunicaciones entre terminales con características similares, las cuales no están separadas por grandes distancias, el utilizar OSI implicaría controles innecesarios y, por ende, desperdicio de tiempo de la CPU y un costo excesivo comparado con sus beneficios.

Por ello, se hace necesario estructurar el sistema operativo de una forma diferente, dando origen así al modelo cliente servidor, éste tendrá un sistema operativo estructurado por grupos de procesos en cooperación, llamados servidores, que ofrecerán servicios a los usuarios, llamados clientes. Las máquinas de los clientes y servidores ejecutan, por lo general, el mismo micronúcleo y ambos se ejecutan como procesos del usuario.

Para evitar un gasto excesivo en los protocolos orientados a la conexión, tales como OSI o TCP/IP, lo usual es que el modelo cliente-servidor se base en un protocolo solicitud/respuesta, sencillo y sin conexión. El cliente envía un mensaje al servidor, para pedir cierto servicio, por ejemplo, la lectura de un bloque de cierto archivo. El servidor hace el trabajo y regresa los datos solicitados o un código de error, para indicar la razón por la cual un trabajo no se pudo llevar a cabo, como se muestra en la Figura 2.3. La principal ventaja de dicho modelo es su sencillez, ya que el cliente envía un mensaje y obtiene una respuesta; por su parte, como se ve en la Figura 2.4, la pila de capas del modelo es mucho más corta que la de OSI, contando únicamente con tres capas: la física, la de enlace de datos y la de solicitud y respuesta.

TRANSFERENCIA DE MENSAJES EN EL MODELO CLIENTE SERVIDOR

Figura 2.3.

TRANSFERENCIA DE MENSAJES EN EL MODELO CLIENTE SERVIDOR

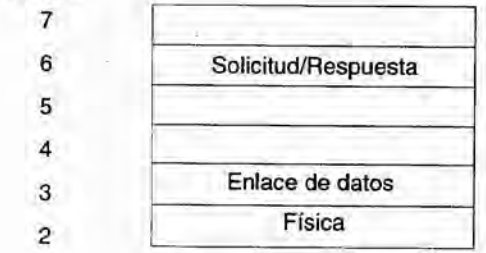


Figura 2.4.

Debido a esta estructura tan sencilla se pueden reducir los servicios de comunicación que presta el micronúcleo; por ejemplo, a dos llamadas al sistema, una para el envío de mensajes y otra para recibirlos. Estas llamadas al sistema se pueden pedir a través de procedimientos de librerías, como SEND (DEST, &MPTR) y RECEIVE (ADDR, & MPTR). La primera envía el mensaje al que apunta MPTR, en un proceso que se identifica como DEST, y provoca que quien hace la llamada se bloquee hasta cuando se envíe el mensaje. La segunda hace que quien hizo la llamada se bloquee hasta que reciba un mensaje. Cuando llega un mensaje, éste se copia en el "buffer" al que apunta MPTR y quien hizo la llamada se desbloquea. El parámetro ADDR determina la dirección en la cual escucha el receptor.

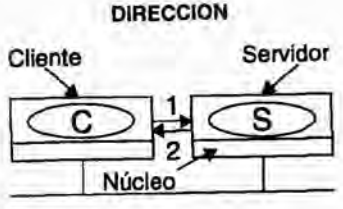
2.3.1. *Dirección*

Para que un cliente pueda enviar un mensaje a un servidor, debe conocer la dirección de éste. Si sólo existe un proceso en ejecución en la máquina-destino cuando se envía el mensaje, el núcleo sabrá qué hacer con el mensaje recibido (dárselo al único proceso en ejecución). Sin embargo, ¿qué ocurre si existen varios procesos en ejecución en la máquina-destino? ¿Cuál de ellos obtiene el mensaje? El núcleo no tiene forma de decidir. En consecuencia, un esquema que utilice las direcciones en la red para la identificación de los procesos indica que sólo se puede ejecutar un proceso en cada máquina. Aunque esta limitación no es fatal, a veces es una seria restricción, razón por la cual surgen tres tipos de métodos para la dirección de los procesos en un modelo cliente-servidor. El primero integra el "machine-number" al código del cliente; es decir, se envían mensajes a los procesos en lugar de enviárselos a las máquinas. Aunque este método elimina toda la ambigüedad acerca de quién es el verdadero receptor, presenta el problema de cómo identificar los procesos. Un esquema común consiste en utilizar nombres con dos partes, para especificar tanto la máquina como el proceso. Por ejemplo, el número 243.4 me indicará que estoy refiriéndome a la máquina 243 y al proceso 4, como se muestra en la Figura 2.5.

El segundo método será dejar que los procesos elijan sus direcciones al azar entre un espacio de direcciones grande y disperso, como lo es el de enteros binarios de 64 bits. La probabi-

lidad de que dos procesos elijan el mismo número es muy pequeña y el método puede utilizarse en sistemas más grandes. Sin embargo, aquí también existe un problema: ¿Cómo sabe el núcleo emisor a cuál máquina enviar el mensaje? La respuesta es simple; el núcleo emisor transmite un paquete especial de localización, con la dirección del proceso-destino, a toda la red; por su parte, los núcleos verifican si la dirección en circulación es la suya y en caso que lo sea, regresa un mensaje diciendo "aquí estoy", con su dirección en la red; es decir, el número de la máquina. En ese momento el núcleo emisor utiliza, entonces, esa dirección y la captura para evitar el envío de otra transmisión la próxima vez que necesite al servidor, como se muestra en la Figura 2.6.

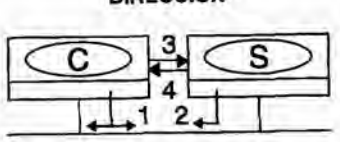
El tercer método, para el direccionamiento de procesos en un modelo-cliente servidor, consiste en utilizar una máquina adicional, llamada servidor de nombres, que me servirá para la asociación, en código ASCII, de los nombres de los servicios con las direcciones de las máquinas, como se muestra en la Figura 2.7; el servidor de nombres le suministra al cliente el número de la máquina donde se localiza, en ese momento, el servidor. Una vez obtenida la dirección, se puede enviar la solicitud de manera directa.



1: Solicitud a 243.0

2: Respuesta a 199.0

Figura 2.5. *Dirección máquina-proceso.*



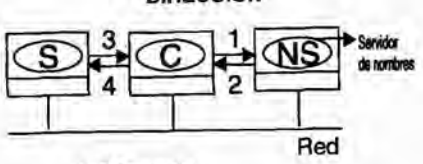
1: Transmisión

2: Aquí estoy

3: Solicitud

4: Respuesta

Figura 2.6. *Dirección de procesos con transmisión.*



1: Búsqueda

2: Respuesta a NS

3: Solicitud

4: Respuesta

Figura 2.7: *Búsqueda de dirección por medio de un servidor de nombres.*

2.3.2. *Primitivas de bloqueo vs. no bloqueo*

Las primitivas de transferencia de mensajes, descritas hasta el momento, reciben el nombre de primitivas de bloqueo, a veces llamadas primitivas síncronas. Cuando un proceso llama un SEND, especifica un destino y un "buffer" dónde enviar ese destino. Mientras se envía el mensaje, el proceso del emisor se bloquea. La instrucción que sigue a la llamada SEND se ejecuta cuando el mensaje se envía en su totalidad, como se muestra en la Figura 2.8. De manera análoga, una llamada a RECEIVE regresa el control cuando realmente se recibe un mensaje y éste se coloca en el "buffer" de mensajes a donde apunta el parámetro.

En RECEIVE, el proceso se suspende hasta cuando llega un mensaje, incluso, aunque tarde varias horas. En ciertos sistemas el receptor puede especificar de quiénes quiere recibir mensajes, en cuyo caso permanece bloqueado hasta que llegue un mensaje del emisor especificado.

Una alternativa a las primitivas con bloqueo son las primitivas sin bloqueo o asíncronas, como se muestra en la Figura 2.9. Si SEND no tiene bloqueo, regresa de inmediato el control a quien hizo la llamada, antes de enviar el mensaje. La ventaja de este esquema es que el proceso del emisor puede continuar su cómputo, en forma paralela con la transmisión del mensaje, en vez de tener inactivo al CPU.

La elección entre las primitivas con bloqueo o sin bloqueo la hacen, por lo general, los diseñadores del sistema, aunque en algunos cuantos sistemas se dispone de ambos y los usuarios pueden elegir su favorito. Sin embargo, la ventaja de desempeño que ofrecen las primitivas sin bloqueo se ve afectada por una seria desventaja: el emisor puede modificar el "buffer" de mensajes cuando se envíe el mensaje; en caso de que se modificara, se enviarían datos traslapados. Para solucionar dicho problema, podríamos optar por copiar el mensaje a un "buffer" interno del núcleo y, así, permitir al proceso continuar con el envío del mensaje y, a la vez, dejar el bloqueo.

Otra alternativa sería la primitiva sin bloqueo con interrupción, en la cual, una vez se ha terminado de enviar el mensaje, se interrumpe al emisor para avisarle que ya puede utilizar el "buffer".

Figura 2.8. *Una primitiva senda con bloqueo.*

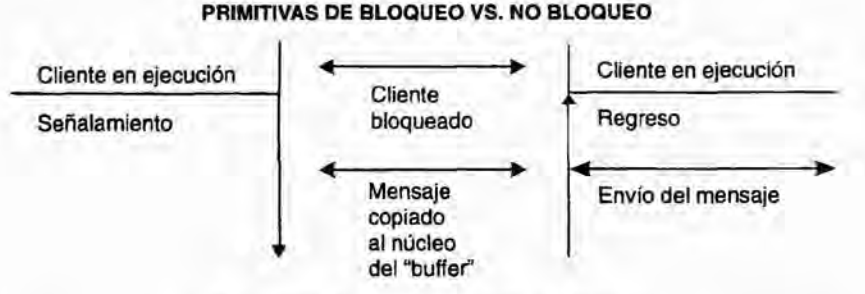


Figura 2.9. *Una primitiva senda sin bloqueo.*

2.3.3. Primitivas almacenadas en "buffer" vs. no almacenadas

Las primitivas descritas hasta ahora son esencialmente primitivas no almacenadas. Esto significa que una dirección se refiere a un proceso específico. Una llamada RECEIVE (ADDR, &M) le indica al núcleo de la máquina en dónde se ejecuta; además, que el proceso que hace la llamada escucha a la dirección ADDR y está preparada para recibir el mensaje enviado a esa dirección.

PRIMITIVAS ALMACENADAS VS. NO ALMACENADAS

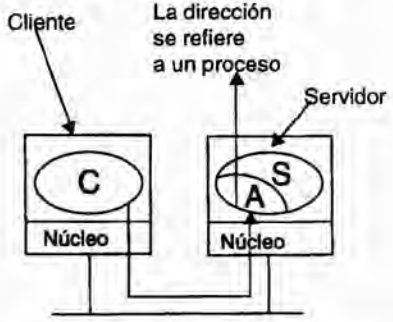


Figura 2.10. *Transferencia de mensajes sin almacenamiento en buffer.*

Se dispone de un único "buffer" de mensajes, al que apunta M, con el fin de capturar el mensaje por llegar. Cuando el mensaje llega el núcleo receptor, lo copia al "buffer" y elimina el bloqueo del proceso receptor, como se muestra en la Figura 2.10.

Un ejemplo de primitiva almacenada es aquél en el cual los mensajes que llegan al servidor son almacenados en un "buffer" llamado buzón, como se ilustra en la Figura 2.11.

PRIMITIVAS ALMACENADAS VS. NO ALMACENADAS

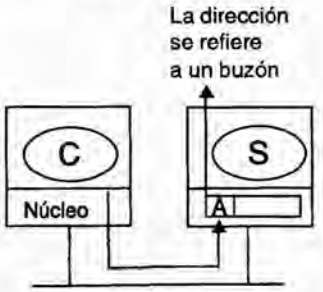


Figura 2.11. *Transferencia de mensajes con almacenamiento en buffer.*

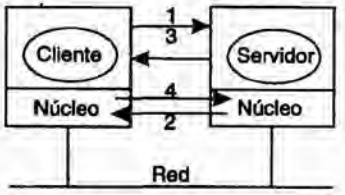
2.3.4. *Primitivas confiables vs. no confiables*

Hemos supuesto de una manera implícita que cuando un cliente envía un mensaje, el servidor lo recibirá. Los mensajes se pueden perder, lo cual afecta la semántica del modelo de transferencia de mensajes. Cuando se utilizan las primitivas por bloqueo y el cliente envía un mensaje, se le suspende hasta que el mensaje ha sido enviado. Sin embargo, cuando vuelve a iniciar, no existe garantía alguna de que el mensaje ha sido entregado. El mensaje podría haberse perdido. Existen tres distintos enfoques de este problema.

El primero consiste en volver a definir la semántica del SEND para hacerlo no confiable. El sistema no da garantía alguna acerca de la entrega de los mensajes. La implantación de una comunicación confiable se deja enteramente en manos de los usuarios. La oficina de correos funciona de esta manera. Cuando usted deposita una carta en un buzón, la oficina de correos hace lo mejor por entregarla, pero no promete nada.

El segundo método exige que el núcleo de la máquina receptora envíe un reconocimiento al núcleo de la máquina emisora. Sólo cuando se reciba este reconocimiento el núcleo emisor liberará el proceso hacia los usuarios (clientes). El reconocimiento va de un núcleo a otro; ni el cliente ni el servidor ven alguna vez un reconocimiento, de la misma forma que la solicitud de un cliente a un servidor es reconocida por el núcleo del servidor; la respuesta del servidor, de regreso al cliente, es reconocida por el núcleo del cliente. Así, una solicitud de respuesta consta de cuatro mensajes, como se muestra en la Figura 2.12.

PRIMITIVAS CONFIABLES VS. NO CONFIABLES

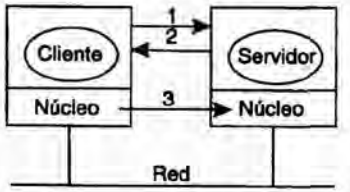


1. Solicitud (del cliente al servidor)
2. Reconocimiento (de núcleo a núcleo)
3. Respuesta del servidor al cliente
4. Reconocimiento (de núcleo a núcleo)

Figura 2.12. *Mensajes reconocidos en forma individual.*

El tercer método aprovecha el hecho que la comunicación cliente-servidor se estructura como una solicitud del cliente al servidor, seguida de una respuesta del servidor al cliente. En este método, el cliente se bloquea después de enviar un mensaje. El núcleo del servidor no envía de regreso un reconocimiento sino que la misma respuesta funciona como tal. Así, el emisor permanece bloqueado hasta que regresa la respuesta. Si tarda demasiado, el núcleo emisor puede volver a enviar la solicitud, para protegerse contra la posibilidad de una pérdida del mensaje. Este método se muestra en la Figura 2.13.

PRIMITIVAS CONFIABLES VS. NO CONFIABLES



1. Solicitud (del cliente al servidor)
2. Respuesta (del servidor al cliente)
3. Reconocimiento (de núcleo a núcleo)

Figura 2.13. *La respuesta se utiliza como reconocimiento de la solicitud.*

El problema con el modelo básico cliente-servidor es que, desde el punto de vista conceptual, la comunicación entre procesos se maneja como E/S, ya que dicha comunicación utiliza uno de los modelos más sencillos de protocolos (solicitud y respuesta). Ahora, cuando una colección de procesos, por ejemplo servidores duplicados de archivo, tiene que comunicarse con otra como grupo, se necesita algo más. Los sistemas de tipo ISIS proporcionan una solución a este problema: la comunicación en grupo. ISIS ofrece una gama de primitivas, de las cuales la más importante es CBCAST. CBCAST ofrece una semántica de comunicación débil con base en la causalidad y se implanta mediante la inclusión de vectores de números secuenciales en cada mensaje, para permitir al receptor que revise si el mensaje se debe entregar de manera inmediata o retrasarse hasta que lleguen algunos mensajes anteriores; éstos no serán tratados por ser casos particulares y más complicados.

De esta forma vemos que, gracias al modelo cliente-servidor, se puede entender un poco mejor cómo se desarrollan las comunicaciones entre procesos en una red de terminales inteligentes y, además, los diferentes aspectos que se

deberían tener en cuenta si se quisiera desarrollar un sistema de estas características; entre ellos aspectos de diseño, tales como si se utilizaran primitivas almacenadas o no almacenadas, confiables o no confiables, con bloqueo o sin bloqueo, etc.

3. SINCRONIZACION EN SISTEMAS DISTRIBUIDOS

Un aspecto muy importante es la forma en que los procesos cooperan y se sincronizan entre sí. Por ejemplo, la forma de implantar las regiones críticas o asignar los recursos en un sistema distribuido.

En los sistemas que sólo cuentan con una CPU, los problemas relativos a las regiones críticas, la exclusión mutua y la sincronización se resuelven, en general, mediante métodos tales como los semáforos y los monitores. Estos métodos no son adecuados para su uso en los sistemas distribuidos, puesto que siempre se basan (de manera implícita) en la existencia de la memoria compartida.

3.1. Sincronización de relojes

La sincronización es más compleja en los sistemas distribuidos que en los centralizados, puesto que los primeros deben utilizar algoritmos distribuidos.

Por lo general no es posible (o recomendable) reunir toda la información relativa al sistema en un solo lugar y después dejar que cierto proceso la examine y tome una decisión, como se hace en el caso centralizado.

En general los algoritmos distribuidos tienen las siguientes propiedades:

1. La información relevante se distribuye entre varias máquinas.
2. Los procesos toman las decisiones sólo con base en la información disponible en forma local.
3. Debe evitarse un único punto de fallo en el sistema.
4. No existe un reloj común o alguna otra fuente precisa del tiempo global.

Los primeros tres puntos indican que es inaceptable reunir toda la información en un solo lugar para su procesamiento; además, el hecho que sólo exista un solo punto de fallo, como éste, hace que el sistema no sea confiable. La idea es que un sistema distribuido debería ser más confiable que las máquinas individuales. Si alguna de ellas falla, el resto puede continuar su funcionamiento.

El último punto de la lista también es crucial. En un sistema centralizado, el tiempo no tiene ambigüedades. Cuando un proceso desea conocer la hora, llama al sistema y el núcleo se lo dice. En un sistema distribuido no es trivial poner de acuerdo todas las máquinas en la hora.

3.1.1. *Relojes lógicos*

Casi todas las computadoras tienen un circuito para el registro del tiempo denominado *cronómetro,* el cual es, por lo general, un cristal de cuarzo trabajado con precisión. Cuando se mantiene sujeto a tensión, un cristal de cuarzo oscila con una frecuencia bien definida, que depende del tipo de cristal, la forma en que se corte y la magnitud de la tensión. A cada cristal se le asocian dos registros, un *contador* y un *registro mantenedor (holding register)*. Cada oscilación del cristal decrementa en uno al contador. Cuando el contador toma el valor cero, se genera una interrupción y el contador se vuelve a cargar mediante el registro mantenedor. De esta forma es posible programar un cronómetro, de modo que genere una interrupción con la frecuencia que se desee. Cada interrupción recibe el nombre de una *marca de reloj*.

Cuando se trabaja con varias máquinas, cada una con su propio reloj, la situación es muy distinta de la que se presenta con un solo sistema. Aunque la frecuencia de un oscilador de cristal es muy estable, es imposible garantizar que los cristales de computadoras distintas oscilen con la misma frecuencia. En la práctica, cuando un sistema tiene n computadores, los n cristales correspondientes oscilarán a tasas un poco distintas, lo que provoca una pérdida de sincronía en los relojes (de software) y que, al leerlos, tengan valores distintos. La diferencia entre los valores del tiempo se llama *distorsión del reloj*. Como consecuencia de esta distorsión, podrían fallar los programas que esperan el tiempo correcto, asociado a un archivo, objeto, proceso o mensaje; esto es independiente del sitio donde haya sido generado (es decir, el reloj utilizado).

En un artículo clásico, Lamport (1978) demostró que la sincronización de relojes es posible y presentó un algoritmo para lograr esto. Lamport señaló que la sincronización de relojes no tiene que ser absoluta. Si dos procesos no interactúan, no es necesario que sus relojes estén sincronizados, puesto que la carencia de sincronización no será observable y, por tanto, no podría provocar problemas. Además, señaló que lo que importa, por lo general, no es que todos los procesos estén de acuerdo de manera exacta en la hora sino que coincidan en el orden en que ocurren los eventos.

Para la mayoría de los fines, basta que todas las máquinas coincidan en la misma hora. Para una cierta clase de algoritmos, lo que importa es la consistencia interna de los relojes, no su cercanía particular al tiempo real. Para estos algoritmos se conviene en hablar de los relojes como *relojes lógicos*. Para sincronizar los relojes lógicos, Lamport definió una relación llamada *ocurre antes de (happens-before)*. La expresión $a \rightarrow b$ se lee: "*a* ocurre antes de *b*" e indica que todos los procesos coinciden en que primero ocurre el evento *a* y después el evento *b*. Esta relación se puede observar de manera directa en dos situaciones:

1. Si *a* y *b* son eventos en el mismo proceso y *a* ocurre antes de *b*, entonces $a \rightarrow b$ es verdadero.
2. Si *a* es el evento del envío de un mensaje por un proceso y *b* es el evento de la recepción del mensaje por otro proceso, entonces $a \rightarrow b$ también es verdadero. Un mensaje no se puede recibir antes de ser enviado o al mismo tiempo en que se envía, puesto que tarda en llegar una cantidad finita de tiempo.

Si dos eventos *x* y *y* están en procesos diferentes que no intercambian mensajes, entonces $x \rightarrow y$ no es verdadero, pero tampoco lo es $y \rightarrow x$. Se dice que estos eventos son *concurrentes*, lo que significa que nada se puede decir (o se necesita decir) acerca del momento en el que ocurren o cuál de ellos es el primero.

Lo que necesitamos es una forma de medir el tiempo, tal que a cada evento *a* le podamos asociar un valor del tiempo C (*a*) en el que todos los procesos estén de acuerdo. El tiempo del reloj C siempre debe ir hacia adelante (creciente) y nunca hacia atrás (decreciente). Se pueden hacer correcciones al tiempo, al sumar un valor positivo al reloj, pero nunca se le debe restar un valor positivo.

En la Figura 3.1 (a) vemos un ejemplo en donde existen tres procesos diferentes que se ejecutan en distintas máquinas, cada una con su propio reloj y velocidad. Cada reloj corre a una razón constante, sólo que las razones son distintas, debido a las diferencias en los cristales. Al tiempo 6, el proceso 0 envía el mensaje A al proceso 1; el reloj del proceso 1 lee 16 cuando el mensaje llega. Si el mensaje acarrea el tiempo de inicio, entonces el proceso 1 concluirá

que tardó 10 marcas de reloj en hacer el viaje. El mensaje B, de 1 a 2, tarde 16 marcas que, al igual que A, es un valor plausible.

El problema está en los mensajes C y D, puesto que van de un valor mayor a uno menor; estos valores son imposibles y ésta es la situación que hay que evitar.

En la solución propuesta por Lamport, para la asignación de tiempos a los eventos, Figura 3.1. (b), cada mensaje acarrea el tiempo de envío de acuerdo con el reloj del emisor. Cuando un mensaje llega y el reloj del receptor muestra un valor anterior al tiempo en que se envió el mensaje, rápidamente el receptor adelanta su reloj para que tenga una unidad más que la del tiempo de envío.

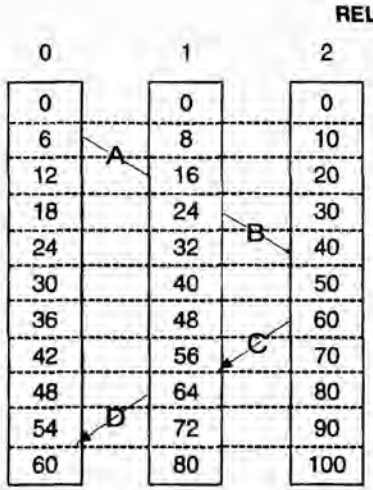


Figura 3.1 (a). *Tres procesos, cada uno con su propio reloj. Los relojes corren a diferentes velocidades.*

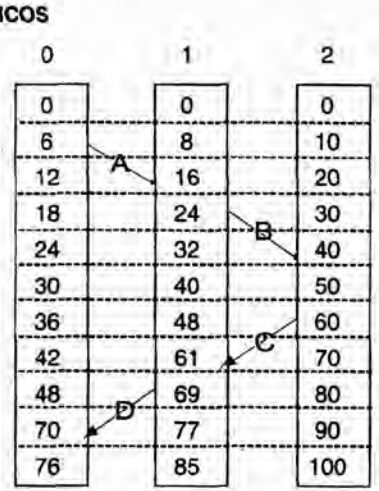


Figura 3.1 (b). *El algoritmo de Lamport corrige los errores.*

Este algoritmo satisface nuestras necesidades para el tiempo global, con una pequeña adición, que entre cualesquiera dos eventos, el reloj debe marcar al menos una vez. Si un proceso envía o recibe dos mensajes en serie muy rápidamente, debe avanzar su reloj en (al menos) una marca entre ellos.

En ciertas situaciones existe un requisito adicional: dos eventos no deben ocurrir exactamente al mismo tiempo. Para lograr esto podemos asociar el número del proceso en que ocurre el evento y el extremo inferior del tiempo, separados por un punto decimal. Así, si ocurren eventos en los procesos 1 y 2, ambos en el tiempo 40, entonces el primero se convierte en 40.1 y el segundo en 40.2.

Por medio de este método tenemos ahora una forma de asignar un tiempo a todos los eventos en un sistema distribuido, con las siguientes condiciones:

1. Si *a* ocurre antes de *b* en el mismo proceso, C (*a*) < C(*b*).
2. Si *a* y *b* son el envío y recepción de un mensaje, C (*a*) < C(*b*).
3. Para todos los eventos *a* y *b*, C(*a*) ≠ C(*b*).

3.1.2. *Relojes físicos*

Cuando existe la restricción adicional de que los relojes no sólo deben ser iguales sino que además no deben desviarse del tiempo real más allá de cierta magnitud, los relojes reciben el nombre de *relojes físicos.* Para estos sistemas se necesitan relojes físicos externos. Por razones de eficiencia y redundancia, por lo general, son recomendables varios relojes físicos, lo cual implica dos problemas:

1. ¿Cómo sincronizar los relojes con el mundo real?
2. ¿Cómo sincronizar los relojes entre sí?

Con la invención del reloj atómico, en 1948, fue posible medir el tiempo de manera mucho más exacta y en forma independiente de todo el ir y venir de la Tierra, al contar las transiciones del átomo de cesio 133. Los físicos retomaron de los astrónomos la tarea de medir el tiempo y definieron el segundo como el tiempo que tarda el átomo cesio 133 para hacer exactamente 9.192.631.770 transiciones.

Actualmente cerca de 50 laboratorios en el mundo tienen relojes de cesio 133. En forma periódica, cada laboratorio le indica a la *Oficina Internacional de la Hora en París (BIH)* el número de marcas de su reloj. La oficina hace un promedio de estos números para producir el *tiempo atómico internacional (TAI),* que aunque es muy estable y está a disposición de todos los que quieran comprarse un reloj de cesio, existe un serio problema con él; 86.400 segundos TAI, a un tiempo cerca de tres milisegundos menor que el de un día solar medio, lo que significaría que, con el paso de los años, el medio día ocurriría cada vez más temprano.

La BIH resolvió el problema mediante la introducción de *segundos de salto,* siempre que la discrepancia entre el TAI y el tiempo solar creciera hasta 800 milisegundos constantes. Esta corrección da lugar a un sistema de tiempo basado en los segundos TAI, pero que permanece en fase con el movimiento aparente del sol; se le llama *tiempo coordenado universal* (UTC).

Para proporcionar UTC a las personas que necesitan un tiempo preciso, *el Instituto Nacional del Tiempo Estándar* (NIST) opera una estación de radio de onda corta, con las siglas WWV, desde Fort Collins, Colorado. WWV transmite un pulso corto al inicio de cada segundo UTC.

Varios satélites terrestres también ofrecen un servicio UTC, tal como el *Satélite de Ambiente Operacional Geoestacionario (GEOS).*

3.1.3. *Algoritmos parar la Sincronización de Relojes*

Si una máquina tiene un receptor WWV, entonces el objetivo es hacer que todas las máquinas se sincronicen con ella. Si ninguna máquina tiene receptores WWV, entonces cada máquina lleva el registro de su propio tiempo y el objetivo es mantener el tiempo de todas las máquinas, tan cercano como sea posible.

El algoritmo de Berkeley

En este caso, el servidor de tiempo (en realidad, un demonio para el tiempo) está activo y realiza un muestreo periódico de todas las máquinas para preguntarles el tiempo. Con base en las respuestas, calcula un tiempo promedio y les indica a todas las demás máquinas que avancen su reloj a la nueva hora o que disminuyan la velocidad del mismo, hasta lograr cierta reducción específica. Este método es adecuado para un sistema donde no exista un receptor de WWV. La hora del demonio, para el tiempo, debe ser establecida en forma manual por el operador, de manera periódica.

En la Figura 3.2 (a), a las 3:00, el demonio para el tiempo indica, a las demás máquinas, su hora y les pregunta las suyas. En (b), las máquinas responden, con la diferencia de sus horas como la del demonio. Con estos números, el demonio calcula el tiempo promedio y le dice a cada máquina cómo ajustar su reloj (c).

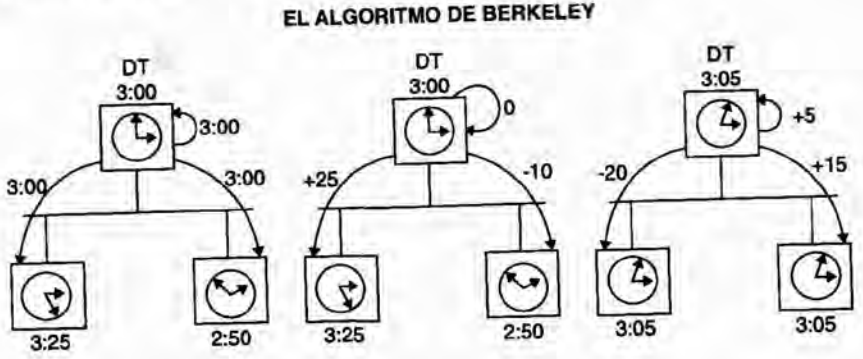


Figura 3.2 (a). *El demonio para el tiempo pregunta a todas las otras máquinas por el valor de sus relojes.* **(b).** *Las máquinas contestan.* **(c)** *El demonio para el tiempo les dice a todas la forma de ajustar sus relojes.*

Algoritmos con promedio

El método anterior es altamente centralizado. Una clase de algoritmos de reloj descentralizada trabaja para dividir el tiempo en intervalos de resincronización de longitud fija. El i-ésimo intervalo inicia en $T0 + iR$ y va hasta $T0 + (i + 1)\ R$ donde $T0$ es un momento ya acordado en el pasado y R es un parámetro del sistema. Al inicio de cada intervalo, cada máquina transmite el tiempo actual, según su reloj. Puesto que los relojes de las diversas máquinas no corren precisamente a la misma velocidad, estas transmisiones no ocurrirán exactamente en forma simultánea.

Después de que una máquina transmite su hora, inicializa un cronómetro local, para reunir las demás transmisiones que lleguen en cierto intervalo S. Cuando llegan todas las transmisiones, se ejecuta un algoritmo para calcular una nueva hora para ellos. El algoritmo más sencillo consiste en promediar los valores de todas las demás máquinas.

Una ligera variación de este tema consiste en descartar primero los m valores más grandes y los m valores más pequeños y promediar el resto. El hecho de descartar los valores extremos se puede considerar como autodefensa contra m relojes fallidos que envían mensajes sin sentido.

3.2. Exclusión mutua

Los sistemas con varios procesos se programan más fácilmente mediante las regiones críticas. Cuando un proceso debe leer o actualizar ciertas estructuras de datos compartidas, primero entra en una región crítica para lograr la exclusión mutua y garantizar que ningún otro proceso utilizará las estructuras de datos al mismo tiempo. Analizaremos algunos ejemplos de implantación de las regiones críticas y exclusión mutua en los sistemas distribuidos.

3.2.1. *Un algoritmo centralizado*

La forma más directa de lograr la exclusión mutua, en un sistema distribuido, es simular la forma en que se lleva a cabo en un sistema con un único procesador. Se elige un proceso como el coordinador; siempre que un proceso desea entrar en una región crítica envía un mensaje de solicitud al coordinador, donde se indica la región crítica en la que desea entrar y pide permiso. Si ningún otro proceso está por el momento en esa región crítica, el coordinador envía una respuesta para el permiso. Figura 3.3 (a).

Cuando llega la respuesta, el proceso solicitante entra en la región crítica. Supongamos ahora que otro proceso, 2, Figura 3.3 (b), pide permiso para entrar en la misma región crítica. El coordinador sabe que un proceso distinto ya se encuentra en esta región, por lo que no puede otorgar el permiso. El método exacto utilizado para negar el permiso depende de la máquina. En la Figura 3.3 (b) el coordinador sólo se abstiene de responder, con lo cual se bloquea el proceso 2, que espera una respuesta. Otra alternativa consiste en enviar una respuesta que diga "permiso denegado". De cualquier manera, forma en una cola la solicitud de 2, por el momento.

Cuando el proceso 1 sale de la región crítica, envía un mensaje al coordinador, para liberar su acceso exclusivo, Figura 3.3 (c). El coordinador extrae el primer elemento de la cola de solicitudes diferidas y envía a ese proceso un mensaje que otorga el permiso. Si el proceso estaba bloqueado (es decir, éste es el primer mensaje que se le envía), elimina el bloqueo y entra en la región crítica.

Es fácil ver que el algoritmo garantiza la exclusión mutua: el coordinador sólo deja que un proceso esté en cada región crítica a la vez. También es justo, puesto que las solicitudes se aprueban en el orden en que se reciben. Ningún proceso espera por siempre. Este esquema también es fácil de implantar y sólo requiere tres mensajes, por cada uso de una región crítica (solicitud, otorgamiento, liberación).

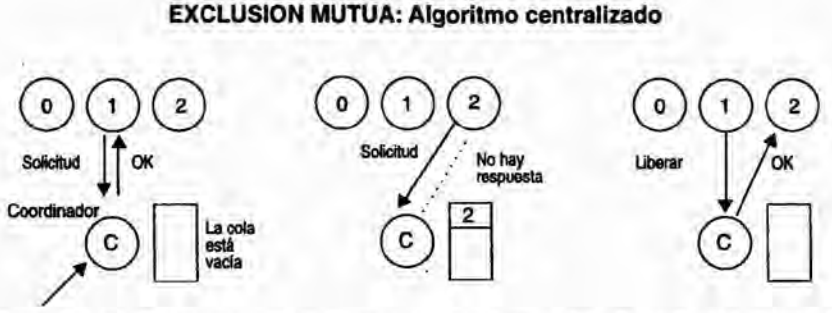


Figura 3.3 (a). *El proceso 1 pide permiso al coordinador para entrar en una región crítica. El permiso es concedido.* **(b).** *El proceso 2 pide, entonces, permiso para entrar en la misma región crítica.* **(c).** *Cuando el proceso 1 sale de la región crítica, se lo dice al coordinador, el cual responde, entonces, a 2.*

El método centralizado también tiene limitaciones. El coordinador es un único punto de fallo, por lo que, si se descompone, todo el sistema se puede venir abajo. Si los procesos se bloquean, por lo general después de hacer una solicitud, no pueden distinguir entre "un coordinador muerto" y un "permiso negado", puesto que en ambos casos no reciben una respuesta. Además, en un sistema de gran tamaño, un único coordinador puede convertirse en un cuello de botella para el desempeño.

3.2.2. *Un algoritmo distribuido*

Ricart yAgrawala (1981) hicieron más eficaz el artículo de Lamport (1978), relativo a la sincronización de los relojes, que fue el primer algoritmo distribuido.

El algoritmo de Ricart y Agrawala requiere la existencia de un orden total de todos los eventos en el sistema; debe ser claro cuál de ellos ocurrió primero.

Cuando un proceso desea entrar en una región crítica, construye un mensaje con el nombre de ésta, su número de proceso y la hora actual. Entonces, envía el mensaje a todos los demás procesos y, de manera conceptual, a él mismo.

Cuando un proceso envía un mensaje de solicitud de otro proceso, la acción que realice depende de su estado con respecto de la región crítica nombrada en el mensaje. Hay que distinguir tres casos:

1. Si el receptor no está en la región crítica y no desea entrar en ella, envía de regreso un mensaje *OK* al emisor.
2. Si el receptor ya está en la región crítica, no responde sino que forma la solicitud en una cola.
3. Si el receptor desea entrar en la región crítica pero no lo ha logrado todavía, compara la marca de tiempo en el mensaje recibido con la marca contenida en el mensaje que envió cada uno. La menor de las marcas gana. Si el mensaje recibido es menor, el receptor envía de regreso un mensaje *OK*. Si su propio mensaje tiene una marca menor, el receptor forma la solicitud en una cola y no envía nada.

Después de enviar las solicitudes que piden permiso para entrar en una región crítica, un proceso espera hasta que alguien más obtenga el permiso. Tan pronto llegan todos los permisos, pueden entrar en la región crítica. Cuando sale de ella, envía mensajes *OK* a todos los procesos en su cola y elimina a todos los elementos de la cola.

En el ejemplo de la Figura 3.4 (a) se muestra qué sucede si dos procesos intentan entrar en la misma región crítica de manera simultánea. El proceso 0 envía, a todos, una solicitud con la marca de tiempo 8, mientras que al mismo tiempo el proceso 2 envía, a todos, una solicitud con la marca de tiempo 12. El proceso 1 no se interesa en entrar en la región crítica, por lo que envía *OK* a ambos emisores. Los procesos 0 y 2 ven el conflicto y comparan las marcas. El proceso 2 pierde y envía un *OK* a 0. El proceso 0 entra en la región crítica, Figura 3.4 (b). Cuando termina, retira la solicitud de 2 de la cola y envía un *OK* a 2 y éste entra en la región crítica, Figura 3.4 (c).

Como en el caso del algoritmo centralizado, la exclusión mutua queda garantizada, sin bloqueo ni inanición. El único punto de fallo es reemplazado por *n* puntos de fallo. Si cualquier proceso falla, no podrá responder a las solicitudes. Este silencio será interpretado (incorrectamente) como una negación del permiso, con lo que se bloquearán los siguientes intentos de los demás procesos para entrar en todas las regiones críticas.

Es posible mejorar un poco este algoritmo. Por ejemplo, la obtención del permiso de todos para entrar en una región crítica es realmente redundante. Todo lo que se necesita es un método para evitar que dos procesos entren en la misma región crítica al mismo tiempo.

EXCLUSION MUTUA: Algoritmo distribuido

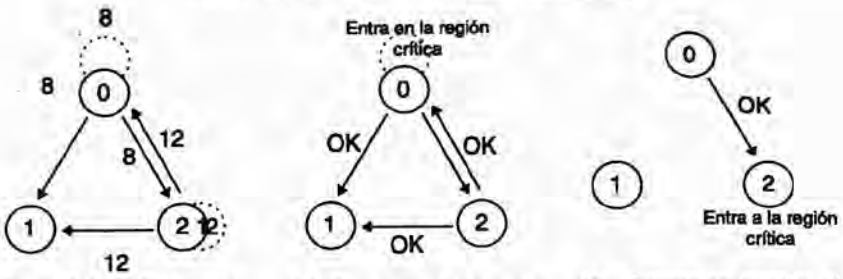


Figura 3.4(a). *Dos procesos desean entrar en la misma región crítica en el mismo momento.*

(b). *El proceso 0 tiene una marca de tiempo menor, por lo que gana.*

(c). *Cuando se termina el proceso 0, envía un OK, por lo que 2 puede ahora entrar en la región crítica.*

3.2.3. *Un algoritmo de anillo de fichas (Token Ring)*

Este es un método completamente distinto para lograr la exclusión mutua en un sistema distribuido. Se tiene por ejemplo una red basada en un bus, Figura 3.5 (a). En software se construye un anillo lógico y a cada proceso se le asigna una posición en el anillo, Figura 3.5 (b). Las posiciones en el anillo se pueden asignar en orden numérico de las direcciones de la red o mediante algún otro medio. Lo importante es que cada proceso sepa quién es el siguiente en la fila después de él.

Al iniciar el anillo, se le da al proceso 0 (cero) una *ficha*, la cual circula en todo el anillo. Se transfiere del proceso *k* al proceso *k +1* en mensajes puntuales. Cuando un proceso obtiene la ficha de su vecino, verifica si intenta entrar en una región crítica. En ese caso, el proceso entra en la región, hace todo el trabajo necesario y sale de la región. Después de salir, pasa la ficha a lo largo del anillo. No se permite entrar en una segunda región crítica con la misma ficha.

Si un proceso recibe la ficha de su vecino y no está interesado en entrar en una región crítica, sólo la vuelve a pasar. En consecuencia, cuando ninguno de los procesos desea entrar en una región crítica, las fichas sólo circulan a gran velocidad en el anillo.

EXCLUSION MUTUA: Anillo de fichas (*Token Ring*)

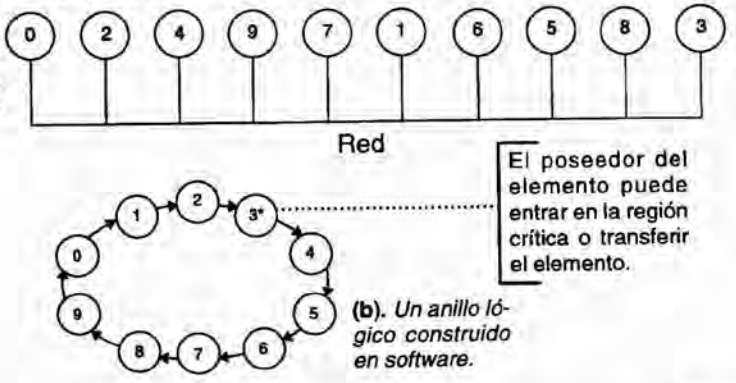


Figura 3.5
(a) *Un grupo no ordenado de procesos en una red.*

(b). *Un anillo lógico construido en software.*

También este algoritmo tiene problemas. Si la ficha llega a perderse, debe ser regenerada. De hecho es difícil detectar su pérdida, puesto que la cantidad de tiempo entre las apariciones sucesivas de la ficha en la red no está acotada. El hecho que la ficha no se haya observado durante una hora no significa su pérdida; tal vez alguien la está utilizando.

El algoritmo también tiene problemas si falla un proceso. Si pedimos un reconocimiento a cada proceso que reciba la ficha, entonces se detectará un proceso muerto si su vecino intenta darle la ficha y fracasa en el intento. En ese momento el proceso muerto se puede eliminar del grupo y el poseedor de la ficha puede enviar ésta, por encima de la cabeza del proceso muerto, al siguiente miembro, o al siguiente después de éste, en caso necesario.

3.3. Algoritmos de elección

Muchos de los algoritmos distribuidos necesitan que un proceso actúe como coordinador, iniciador, secuenciador o que desempeñe, en cierta forma, algún papel especial. Si todos los procesos son idénticos, no existe forma de elegir uno de ellos como especial. En consecuencia, supondremos que cada proceso tiene un número único; por ejemplo, su dirección en la red. En general los algoritmos de elección intentan localizar el proceso con el máximo número del proceso y designarlo como coordinador.

Además, también supondremos que cada proceso sabe el número del proceso de todos los demás; lo que sí desconoce es si los procesos están activos o inactivos. El objetivo de un algoritmo de elección es garantizar que al iniciar una elección, ésta concluya con el acuerdo de todos los procesos con respecto a la identidad del nuevo coordinador.

3.3.1. *El algoritmo del grandulón*

Diseñado por García-Molina (1982). Cuando un proceso observa que el coordinador ya no responde a las solicitudes, inicia una elección. Un proceso *P* realiza una elección de la siguiente manera:

1. *P* envía un mensaje *elección* a los demás procesos, con un número mayor.
2. Si nadie responde, *P* gana la elección y se convierte en el coordinador.
3. Si uno de los procesos, con un número mayor, responde, toma el control. El trabajo de *P* termina.

En cualquier momento un proceso puede recibir un mensaje *elección* de uno de sus compañeros, con un número menor. Cuando esto sucede, el receptor envía de regreso un mensaje *OK* al emisor para indicar que está vivo y que tomará el control. El receptor realiza entonces una elección, a menos que ya esté realizando alguna. En cierto momento, todos los procesos se rinden, menos uno, el cual se convierte en el nuevo coordinador. Anuncia su victoria al enviar un mensaje a todos los procesos para indicarles que, a partir de ese momento, es el nuevo coordinador.

Si un proceso inactivo se activa, realiza una elección. Si ocurre que es el proceso en ejecución, con el número máximo, ganará la elección y tomará para sí el trabajo del coordinador. Así, siempre gana el tipo más grande; de ahí el nombre.

En la Figura 3.6 vemos el funcionamiento de este algoritmo. El grupo consta de ocho procesos, numerados de 0 al 7. El proceso 7 era el coordinador pero acaba de fallar. El proceso 4 lo nota y envía mensajes *elección* a los procesos mayores que el 5, 6 y 7, Figura 3.6 (a). Los procesos 5 y 6 responden *OK* Figura 3.6 (b); el proceso 4 sabe que en este momento ha terminado su trabajo. En la Figura 3.6 (c), tanto 5 como 6 realizan elecciones enviando mensajes a procesos mayores que ellos. En la Figura 3.6 (d) el proceso 6 indica a 5 que tomará el control. Cuando está listo para asumir el control, el proceso 6 lo anuncia, para lo cual envía mensajes *coordinador* a todos los procesos en ejecución.

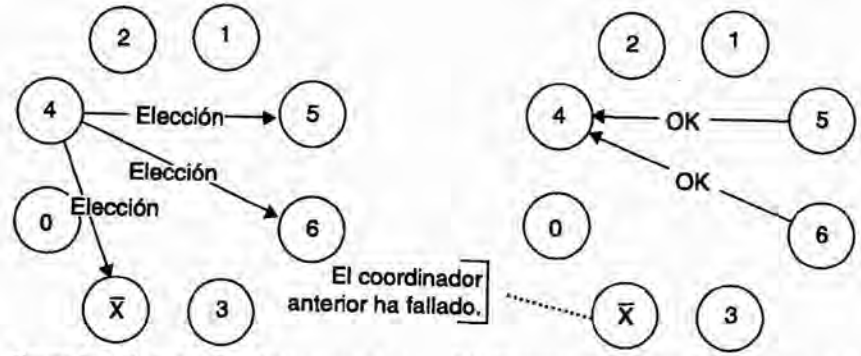
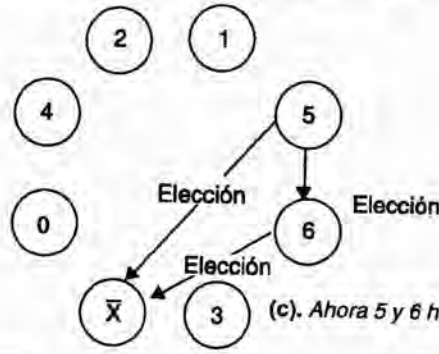
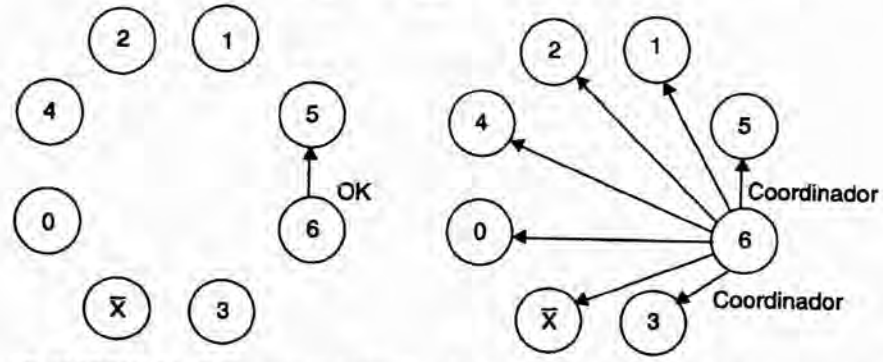


Figura 3.6 (a). *El proceso 4 hace una elección.* **(b).** *El proceso 5 y 6 responde e indica a 4 que se detenga.*



(c). *Ahora 5 y 6 hacen una elección.*



(d). *El proceso 6 indica a 5 que se detenga.* **(c).** *El proceso 6 gana y se lo informa a todos.*

3.3.2. *Un algoritmo de anillo*

Suponemos que los procesos tienen un orden, físico o lógico, de modo que cada proceso conoce a su sucesor. Cuando algún proceso observa que el coordinador no funciona, construye un mensaje *elección* con su propio número de proceso y envía el mensaje a su sucesor. Si éste está inactivo, el emisor pasa sobre el sucesor y va hacia el siguiente número del anillo o al siguiente de éste, hasta localizar un proceso en ejecución. En cada paso el emisor añade su propio número de proceso a la lista en el mensaje.

En cierto momento el mensaje regresa a los procesos que lo iniciaron. Ese proceso reconoce este evento cuando recibe un mensaje de entrada con su propio número de proceso. En este punto el mensaje escrito cambia a *coordinador* y circula de nuevo, esta vez para informar a todos los demás quién es el coordinador (el miembro de la lista con el número máximo) y quiénes son los miembros del nuevo anillo. Una vez ha circulado, el mensaje se elimina y todos se ponen a trabajar. En la Figura 3.7 vemos qué ocurre si dos procesos, 2 y 5, descubren simultáneamente que ha fallado el coordinador, proceso 7. Cada uno construye un mensaje *elección* y comienza a circularlo. En cierto momento ambos mensajes habrán dado la vuelta y ambos se convertirán en mensajes *coordinador*, con los mismos miembros y en el mismo orden. Cuando han dado una vuelta completa, ambos serán eliminados.

ALGORITMOS DE ELECCION: Algoritmo de anillo

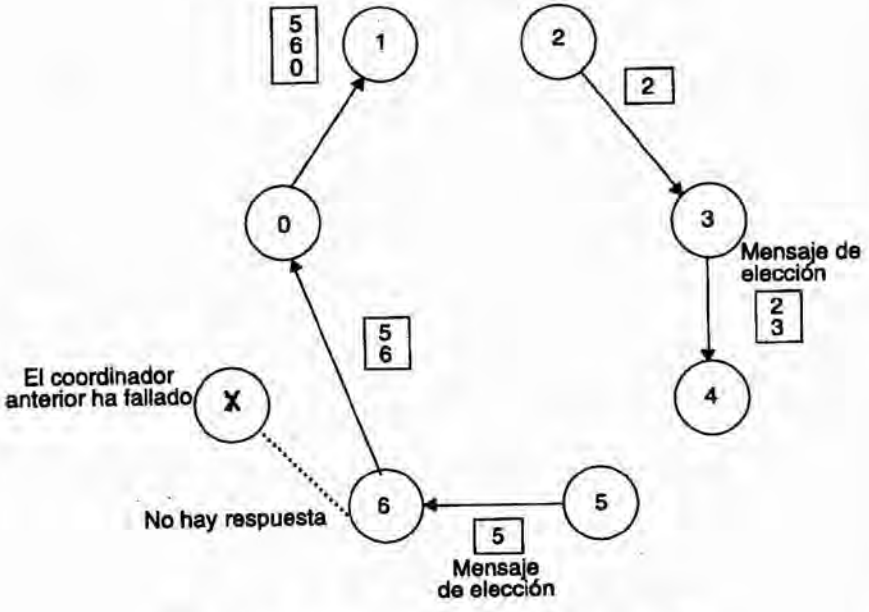


Figura 3.7(a) *Algoritmo de elección mediante un anillo.*

3.4. Bloqueos en sistemas distribuidos

Son más difíciles de evitar, prevenir e incluso detectar, además de ser más difíciles de curar cuando se les sigue la pista, puesto que toda la información relevante está dispersa en muchas máquinas.

Existen cuatro estrategias usuales para el manejo de los bloqueos:

1. El algoritmo del avestruz (ignorar el problema).
2. Detección (permitir que ocurran los bloqueos, detectarlos e intentar recuperarse de ellos).
3. Prevención (hacer que los bloqueos sean imposibles desde el punto de vista estructural).
4. Evitarlos (evitar los bloqueos mediante la asignación cuidadosa de los recursos).

Las cuatro estrategias son aplicables de manera potencial a los sistemas distribuidos. Nuestro análisis de los bloqueos de los sistemas distribuidos se centrará solamente en dos de las técnicas: detección y prevención.

3.4.1. *Detección distribuida de bloqueos*

Detección centralizada de bloqueos

Aunque cada máquina mantiene la gráfica de recursos de sus propios procesos y recursos, un coordinador central mantiene la gráfica de recursos de todo el sistema (la unión de todas las gráficas individuales). Cuando el coordinador detecta un ciclo, elimina uno de los procesos para romper el bloqueo.

A diferencia del caso centralizado, donde se dispone de toda la información de manera automática en el lugar correcto, en un sistema distribuido esta información se debe enviar de manera explícita.

Detección distribuida de bloqueos

Un algoritmo típico es el algoritmo de *Chandy-Misra-Haas*. Este permite que los procesos soliciten varios recursos (por ejemplo, cerraduras) al mismo tiempo, en vez de uno cada vez. Al permitir las solicitudes simultáneas de varios procesos, la fase de crecimiento de una transacción se puede realizar mucho más rápido. La consecuencia de este cambio al modelo es que un proceso puede esperar ahora a dos o más recursos en forma simultánea.

El algoritmo *Chandy-Misra-Haas* se utiliza cuando un proceso debe esperar cierto recurso.

Un algoritmo alternativo consiste en que cada proceso añada su identidad al final del mensaje de exploración, de modo que cuando regrese al emisor inicial se enliste todo el ciclo. El emisor puede ver, entonces, cuál de los procesos tiene el número más grande y eliminarlo, o bien enviar un mensaje que le pida que se elimine a sí mismo. De cualquier forma, si varios procesos descubren el mismo ciclo al mismo tiempo, todos elegirán la misma víctima.

3.4.2. *Prevención distribuida de bloqueos*

Consiste en el diseño cuidadoso del sistema, de modo que los bloqueos sean imposibles, desde el punto de vista estructural. Entre las distintas técnicas se incluye el permitir a los procesos que sólo conserven un recurso a la vez; exigir a los procesos que soliciten todos sus recursos desde un principio y hacer que todos los procesos liberen todos sus recursos cuando soliciten uno nuevo.

En un sistema distribuido con tiempo global y transacciones atómicas son posibles otros algoritmos prácticos. Ambos se basan en la idea de asociar,

a cada transacción, una marca de tiempo global al momento de su inicio.

La idea detrás de este algoritmo es que cuando un proceso está a punto de bloquearse, en espera de un recurso que está utilizando otro proceso, se verifica cuál de ellos tiene la marca de tiempo mayor (es decir, es más joven).

Podemos permitir, entonces, la espera sólo si el proceso en estado de espera tiene una marca inferior (más viejo) que el otro. Otra alternativa consiste en permitir la espera de procesos sólo si el proceso que espera tiene una marca mayor (es más joven) que el otro proceso, en cuyo caso las marcas aparecen en la cadena en forma descendente.

Aunque las dos opciones previenen los bloqueos, es más sabio dar prioridad a los procesos más viejos. Se han ejecutado durante más tiempo, por lo que el sistema ha invertido mucho en ellos y es probable que conserven más recursos.

En la Figura 3.8 (a), un proceso antiguo desea un recurso que mantiene un proceso joven; debemos permitir al proceso que continúe. En (b), un proceso joven desea un recurso que mantiene un proceso antiguo; en este caso, lo debemos eliminar. Este algoritmo se denomina *espera-muerte.*

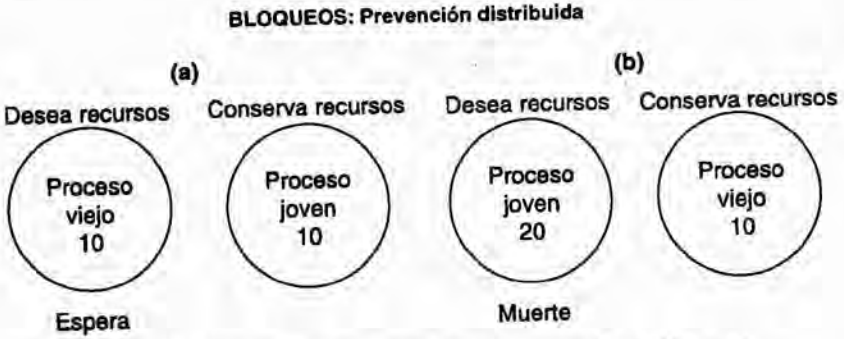


Figura 3.8. *Algoritmo espera-muerte para la prevención de bloqueos.*

Figura 3.9. *Algoritmo herida-muerte para la prevención de bloqueos.*

La Figura 3.9 (a) recibe el nombre *Derecho de prioridad* y la (b) se denomina *espera*. Este algoritmo se llama *herida-espera*. Si un proceso antiguo desea un recurso mantenido por uno joven, el proceso antiguo ejerce su derecho de prioridad sobre el joven, Figura 3.9 (a). Es probable que el joven vuelva a iniciar su ejecución de manera inmediata, con lo que intentaría adquirir de nuevo el recurso; esto lleva a la situación de la Figura 3.9 (b).

4. PROCESOS Y PROCESADORES EN SISTEMAS DISTRIBUIDOS

Aunque los procesos son un concepto muy importante en los sistemas de un procesador, se tratará de hacer énfasis en los aspectos del manejo de procesos que frecuentemente no son estudiados en el contexto de los sistemas operativos clásicos.

En la mayoría de los sistemas distribuidos se presenta la posibilidad de tener muchos hilos ("threads") de control dentro de un proceso, lo cual llevará a tener ventajas significantes, pero también se presentarán varios problemas que serán estudiados más adelante.

4.1. Hilos

Los hilos se inventaron para permitir la combinación del paralelismo con la ejecución secuencial y el bloqueo de las llamadas al sistema. En la mayoría de los sistemas operativos tradicionales, cada proceso tiene un espacio de direcciones y un único hilo de control (puede ser ésta una aproximada definición de un proceso); a pesar de que frecuentemente existan situaciones en donde se desea tener varios hilos de control que compartan el mismo espacio de direcciones, pero que se ejecuten de manera cuasi-paralela, como si fuesen procesos independientes (siendo su única *diferencia* que comparten el mismo espacio de direcciones).

En la Figura 4.1, parte (a), se tiene una máquina con tres procesos. Cada uno de ellos posee su propio contador del programa, su propia pila, su propio conjunto de registros y su propio espacio de direcciones. Los procesos no tienen ninguna relación entre sí; la única manera posible de comunicación sería mediante las primitivas de comunicación entre procesos del sistema, como los semáforos, monitores o mensajes. En la parte (b) se tiene otra máquina con un solo proceso, con la única diferencia que éste tiene varios hilos de control, los cuales generalmente son llamados **hilos** o **procesos ligeros.** En muchas ocasiones se pueden considerar los hilos como unos miniprocesos. Cada hilo se ejecuta de manera estrictamente secuencial y tiene su propio contador del programa y una pila para llevar un registro de su posición.

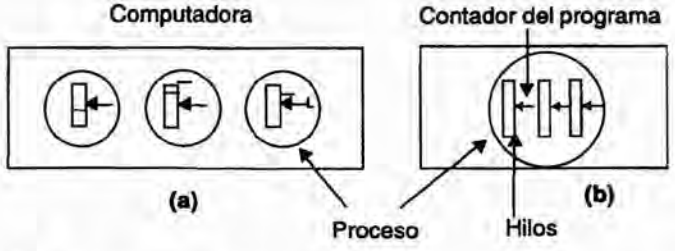


Figura 4.1 **(a).** *Tres procesos con un hilo cada uno.*
(b). *Un proceso con tres hilos.*

Los hilos comparten la CPU de la misma forma que lo realizan los procesos: primero se ejecuta un hilo y después otro (tiempo compartido).

Solamente teniendo un multiprocesador se podrían ejecutar realmente en paralelo.

Otra característica de los hilos es que éstos pueden crear hilos hijos y se pueden bloquear en espera de que se terminen los llamados al sistema, al igual que en los procesos regulares. Mientras un hilo se encuentre bloqueado se puede ejecutar otro hilo del mismo proceso (igualmente ocurre cuando se bloquea un proceso).

Claro está que los hilos no son tan independientes como ocurre en los procesos distintos; todos los hilos tienen el mismo espacio de direcciones, lo cual quiere decir que comparten las mismas variables globales. Cada hilo puede tener acceso a cada dirección virtual, y puede realizar las siguientes tareas: leer, escribir o limpiar de manera completa la pila de otro hilo. Como se puede observar, no existe ninguna protección entre los hilos, ya que podría llegar a ser casi imposible y, en muchos casos, hasta innecesario.

Además de poseer el mismo espacio de direcciones, todos los hilos comparten el mismo conjunto de archivos abiertos, procesos hijos, cronómetros, señales, etc., como se ilustra en la Figura 4.2.

Elementos por hilo
Contador del prog.
Pila
Conj. de registros
Hilos de los hijos
Estado

Elementos por proceso
Espacio de dirección
Variables globales
Archivos abiertos
Procesos fijos
Cronómetros
Señales
Semáforos
Información contable.

Figura 4.2 *Conceptos por hilo y por proceso*

En los procesos tradicionales (procesos con único hijo) los hilos pueden tener uno de los siguientes estados: en ejecución, bloqueado, listo o terminado. Un hilo en ejecución posee la CPU y se encuentra activo. Un hilo bloqueado espera que otro elimine el bloqueo (como ocurre en un semáforo). Un hilo listo está programado para su ejecución, la cual la realizará cuando llegue su debido turno. Y por último, un hilo terminado es aquél que ha hecho su salida pero que todavía no es recogido por su padre.

En la Figura 4.3 se ilustra un ejemplo de servidor de archivos, el cual muestra las diferentes formas de organizaciones de hilos en un proceso. En la parte (a) se muestra una primera organización en la que un hilo, el servidor, lee las solicitudes de trabajo en el buzón del sistema. Después de examinar la solicitud, él elige un hilo trabajador inactivo (bloqueado) y le envía la correspondiente solicitud, lo cual se realiza, con frecuencia, al escribir un apuntador el mensaje, en una palabra especial asociada a cada hilo, dando como resultado que el servidor despierte al trabajador dormido. Cuando el trabajador despierta, hace la correspondiente verificación por medio del bloque Caché compartido, el cual puede ser accesado por todos los hilos

y dirá si se puede satisfacer esa solicitud.

En el modelo de equipo, parte (b), todos los hilos son iguales y cada uno obtiene y procesa sus propias solicitudes. Como se puede observar en la figura, no hay servidor, y a veces llega un trabajo que un hilo no puede manejar, ya que cada hilo se especializa en manejar cierto tipo de trabajo. Para este caso, se puede manejar una cola de trabajo pendiente. Con este tipo de organización, un hilo debe verificar primero la cola de trabajo, antes de buscar en el buzón del sistema.

En la figura, parte (c), se ilustra el modelo de entubamiento, en el cual el primer hilo genera ciertos datos y los transfiere al siguiente, para su procesamiento. Los datos pasan de hilo en hilo y, en cada etapa, se lleva a cabo cierto tipo de procesamiento. Esto puede llegar a ser una buena solución en problemas como el de productores y consumidores, pero no sería adecuado para servidores de archivos. Los entubamientos se utilizan ampliamente en muchas áreas de los sistemas de cómputo, desde la estructura interna de la CPU RISC, hasta las líneas de comandos de UNIX.

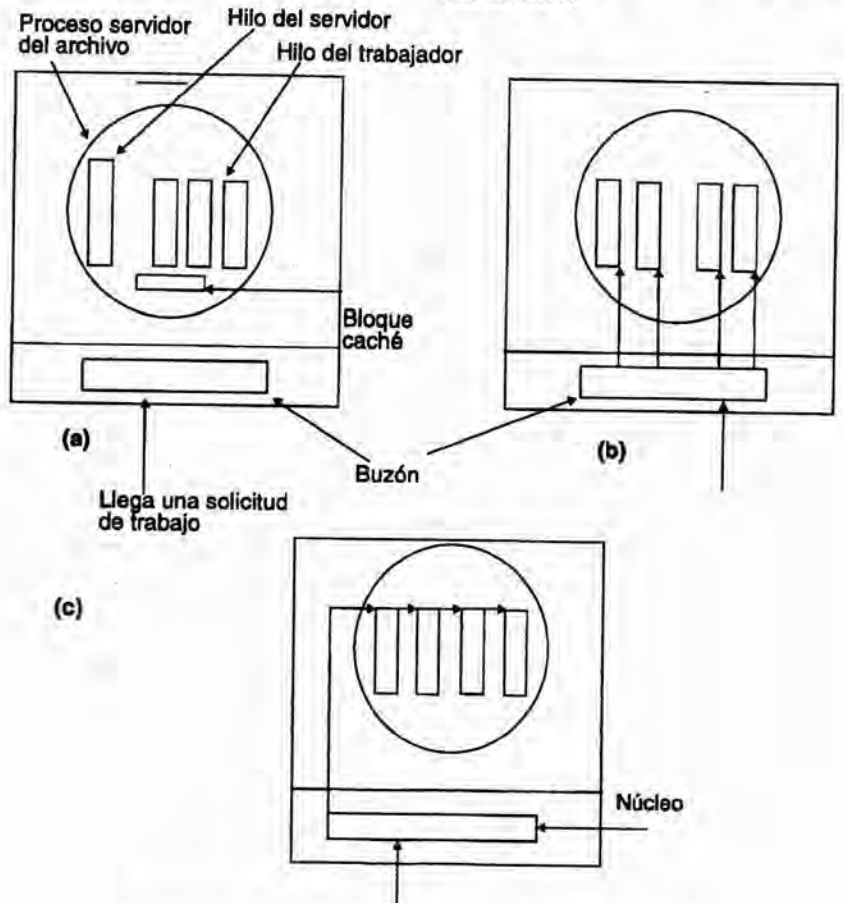


Figura 4-3. *Tres organizaciones de hilos en un proceso.*

4.2. Modelos de sistemas

Como ya es sabido, los procesos se ejecutan en procesadores; en un sistema tradicional sólo existe un único procesador, mientras que un sistema distribuido, dependiendo del aspecto del diseño, puede tener varios procesadores. Los procesadores de un sistema distribuido se pueden organizar de varias formas. En este parágrafo se tendrán en cuenta los principales, como son: el modelo de estación de trabajo y el modelo de pilas de procesadores. Estos dos modelos se basan en diferentes filosofías, en lo que concierne a un sistema distribuido.

4.2.1. *El modelo de estación de trabajo*

El modelo de estación de trabajo es un sistema que consta de estaciones de trabajo (computadoras personales de alta calidad), dispersas en un edificio o un lugar de trabajo y conectadas entre sí por medio de una LAN de alta velocidad, como se ilustra en la Figura 4.6.

SISTEMA BASADO EN EL MODELO DE LA PILA DE PROCESADORES

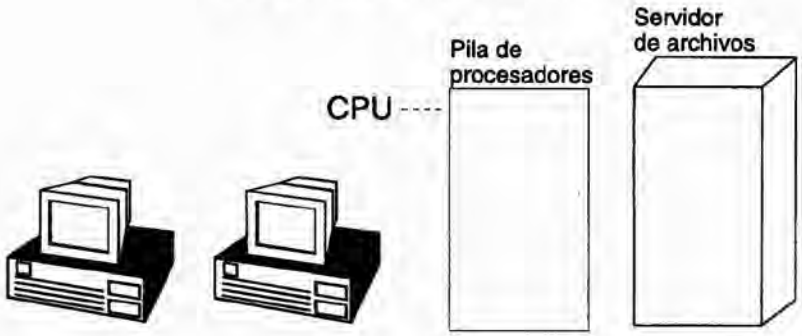


Figura 4.6

Las estaciones de trabajo pueden estar ubicadas en diferentes partes, como en las oficinas de trabajo, donde cada una de ellas se dedica a un único usuario; otras pueden estar en sitios públicos y tener distintos usuarios en el transcurso del día. Es posible que en algún instante de tiempo, ambas estaciones pueden tener un único usuario conectado a ella y tener, entonces, un "poseedor" o no tenerlo (estar inactiva).

Existen dos clases de estaciones de trabajo: estaciones de trabajo sin disco o estaciones de trabajo con disco. Si las estaciones carecen de disco, el sistema de archivos debe ser implantado mediante uno o varios servidores de archivos en la red. Las solicitudes de lectura o escritura serán enviadas a un servidor de archivos, el cual realizará el correspondiente trabajo y enviará de regreso las respuestas.

La popularidad de las estaciones de trabajo sin disco se debe a su precio, el cual es mucho más bajo que si se tuviese una estación equipada con un pequeño disco, razón por la cual se emplean uno o dos servidores de archivos con discos enormes y rápidos, a los cuales se tiene acceso mediante la LAN. Otras características que hacen populares a las estaciones de trabajo sin disco son: su fácil mantenimiento y su flexibilidad; por todo esto, la gran mayoría de estas estaciones se encuentra instalada en universidades y empresas.

Si la estación de trabajo tiene sus propios discos, podrán ser utilizados de las siguientes maneras:

1. Paginación y archivos temporales.

   En este modelo los discos locales se utilizan exclusivamente para la paginación y los archivos temporales no compartidos, que se puedan eliminar al final de la sesión.

2. Paginación, archivos temporales y binarios del sistema.

   Presenta las mismas características que el punto anterior, con la única variante que los discos locales también contienen los programas en binario (ejecutables), tales como compiladores, editores de texto y manejadores de correo electrónico. Cuando se llama a alguno de estos programas, se le busca en el disco local y no en el servidor de archivos, lo cual reduce la carga en la red.

3. Paginación, archivos temporales, binarios del sistema y ocultamiento de archivos.

   Este método consiste en utilizar, de forma explícita, los discos locales como "cachés" (además de utilizarlos para la paginación, los archivos temporales y los binarios). Los usuarios pueden cargar los archivos desde los servidores de archivos hasta sus propios discos; leerlos y escribir en ellos, de manera local y después regresar los archivos modificados, reduciendo la carga en la red, al mantener los archivos modificados al final de la sesión.

4. Un sistema local de archivos completo.

   En este caso cada máquina tiene su propio sistema de archivos autocontenido, con la gran posibilidad de acceder a los sistemas de archivos de otras máquinas, lo cual da como resultado reducir el contacto con el mundo exterior, generando así poca carga a la red, pero ocasionaría su principal desventaja, que será la de convertirse en un sistema operativo de red y no en un verdadero sistema distribuido.

Ventajas de las estaciones de trabajo

1. Los usuarios tienen una cantidad fija de poder de cómputo exclusivo, garantizando así su propio tiempo de respuesta.
2. Los programas gráficos sofisticados pueden ser muy rápidos, ya que pueden tener un rápido acceso a la pantalla.
3. Cada usuario tiene un alto grado de autonomía y puede asignar sus recursos correspondientes como lo juzgue necesario.
4. Los discos locales hacen posible que los trabajos continúen si el servidor de archivos llega a fallar.

4.2.2. *El modelo de la pila de procesadores*

El método de la pila de procesadores se ilustra en la Figura 4.7. Como se puede observar en el gráfico, en este modelo se les proporcionan terminales gráficas de alto rendimiento, denotadas en la figura con la letra X. Esta implementación se hace debido a que los usuarios desean tener una interface gráfica, de alta calidad y un buen desempeño.

La creación del modelo de la pila de procesadores surge para complementar la idea de las estaciones de trabajo sin disco. Como el sistema de archivos se concentra en un pequeño número de servidores, para economizar la escala, se tomó la idea de hacer lo mismo para los servidores de computadoras. Teniendo como base lo anterior, se colocan

todas las CPU en gabinetes de gran tamaño, dentro del cuarto de máquinas, reduciendo así los costos de energía y los de empaquetamiento, con lo cual se produce un mayor poder de cómputo por una misma cantidad de dinero, como también se permite el uso de terminales más baratas. Todo el poder de cómputo radica en poder tener "estaciones de trabajo inactivas", a las que se puede tener acceso de manera dinámica. Los usuarios podrán obtener las CPU que sean necesarias, durante períodos cortos, y después regresarlas para que otros usuarios puedan disponer de ellas.

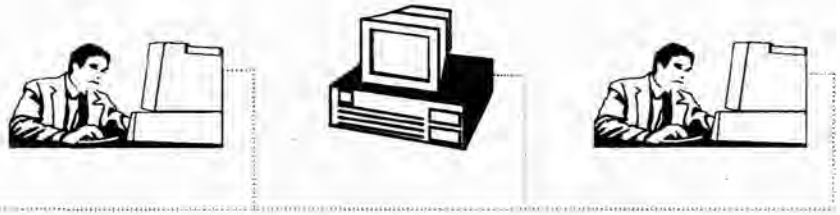


Figura 4.7 *Red de estaciones de trabajo personal.*

El poder de cómputo se ha centralizado como una pila de procesadores, debido a la teoría de colas, donde los usuarios generan, en forma aleatoria, solicitudes de trabajo a un servidor, y cuando éste se encuentre ocupado, los usuarios se forman para el servicio y se procesan de acuerdo con su turno, como se observa en la Figura 4.8.

Figura 4.8. *Un sistema básico con colas.*

Una de las grandes aplicaciones de la pila de procesadores ocurre cuando varios usuarios se encuentran trabajando en un gran proyecto de software, los cuales llevan a cabo grandes simulaciones, ejecutan grandes programas de inteligencia artificial, obteniendo un fácil manejo y acceso al manejar una pila; mientras que el trabajar en estaciones de trabajo acarrearía enormes dificultades y no sería nada sencillo.

4.2.3. *Un modelo híbrido*

Este modelo es una combinación de los dos modelos anteriormente presentados, el cual combina las ventajas que posee cada uno, con la única diferencia de que resultaría mucho más costoso.

El modelo consiste en proporcionar a cada usuario una estación de trabajo personal y además tener una pila de procesadores. El trabajo interactivo se puede llevar a cabo en las estaciones de trabajo, garantizando su respuesta. Este modelo hace que las estaciones inactivas no sean utilizadas, lo cual hará más sencillo el sistema y además todos los procesos no interactivos se ejecutan en la pila de procesadores, así como todo el cómputo pesado en general.

La gran ventaja de este modelo es que proporciona una respuesta interactiva mucho más rápida, como también un uso eficiente en los recursos y presenta un diseño sencillo.

4.3. Asignación de procesadores

Como ya hemos mencionado anteriormente, un sistema distribuido consta de varios procesadores. Estos pueden organizarse como una colección de estaciones de trabajo personales, o como una pila pública de procesadores o de alguna forma híbrida. No importa cuál utilicemos, necesitaremos un algoritmo para decidir qué proceso se va a ejecutar y en qué máquina. Este parágrafo tendrá como principal objetivo estudiar los algoritmos que serán utilizados, para determinar qué proceso se asigna a un determinado procesador, dando como resultado una "asignación de procesadores".

4.3.1. *Modelos de asignación*

Las estrategias de asignación de procesadores se pueden dividir en dos grandes categorías. La primera, llamada no migratoria, al crearse un proceso, toma la decisión de dónde colocarlo. Una vez colocado el proceso en una máquina, permanece allí hasta que termine; no se puede mover, así exista sobrecarga para la máquina o existan otras máquinas inactivas. La segunda es la de los algoritmos migratorios, en donde un proceso se puede trasladar aunque haya iniciado su ejecución. Las estrategias migratorias permiten tener un mejor balance de carga, pueden resultar más complejas y tendrían un efecto fundamental en el diseño del sistema.

El objetivo que tienen los algoritmos de asignación de procesos, en los procesadores, es maximizar el uso de la CPU; es decir, maximizar el uso de ciclos de CPU que se ejecutan en beneficio de los trabajos del usuario por cada hora de tiempo real. Al tratar de maximizar el uso de la CPU, se trata de evitar todo el costo que se tiene por el tiempo inactivo de la CPU. Otro objetivo importante es la minimización del tiempo promedio de respuesta.

4.3.2. *Aspectos del diseño de algoritmos de asignación de procesadores*

Las principales decisiones que debe tomar un diseñador se pueden resumir en los siguientes aspectos:

1. Algoritmos determinísticos vs. Heurísticos. Los determinísticos son adecuados cuando se sabe de antemano todo aquello acerca de los procesos; se supone que se tiene una lista completa de todos los procesos o necesidades de cómputo, de archivos, de comunicación. Con esta información sería posible hacer una asignación perfecta, creando así una mejor asignación.

   Los heurísticos son aquellos sistemas en donde la carga es totalmente impredecible. Las solicitudes de trabajo dependen de quien las esté haciendo, y pueden variar de manera drástica cada hora, e incluso cada minuto. La asignación de procesadores, en estos sistemas, utiliza técnicas ad hoc.
2. Algoritmos centralizados vs. Distribuidos. La recolección de la información, en un lugar, permite tomar una mejor decisión, pero coloca una máquina pesada en la máquina central.

El hecho de ser centralizado, en vez de maximizar el uso de la CPU, busca darle a cada procesador, de una estación de trabajo, una parte justa del poder de cómputo.

3. Algoritmos óptimos vs. Subóptimos. Se pueden obtener soluciones óptimas tanto en los sistemas centralizados como en los descentralizados, pero, por regla general, son más caras que las subóptimas, pues hay que recolectar más información y procesarla un poco más. En la vida real, la mayoría de los sistemas distribuidos buscan soluciones subóptimas, heurísticas y distribuidas, debido a la dificultad para obtener las óptimas.
4. Algoritmos iniciados por el emisor vs. Iniciados por el receptor. También es llamado política de localización. Después de que la política de transferencia ha decidido deshacerse de un proceso, la política de localización debe decidir dónde enviarlo. Esta política no puede ser local, debido a que se necesita información de la carga, en todas partes, para tener una óptima decisión: sin embargo, esta información se puede dispersar de dos formas; en la primera, los emisores son quienes inician el intercambio de información y, en la segunda, es el receptor el que toma la iniciativa.

A continuación, en la Figura 4.9, mencionaremos un ejemplo sencillo para denotar el anterior caso:

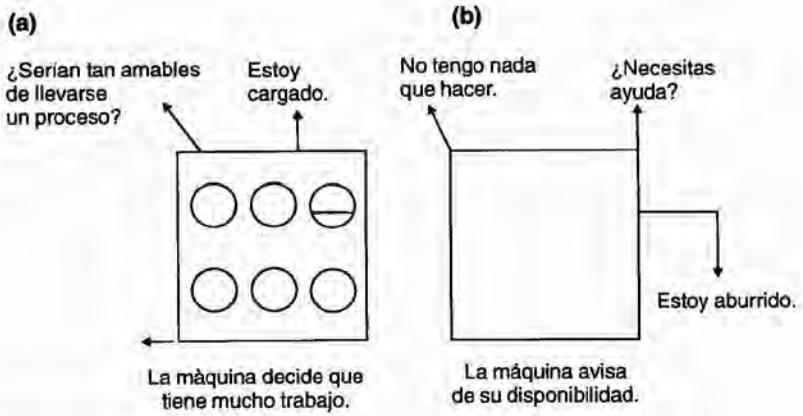


Figura 4.9 (a). *Un emisor en búsqueda de una máquina inactiva*
(b). *Un receptor en búsqueda de trabajo por realizar*

En la parte (a) se ilustra una máquina sobrecargada, la cual envía una solicitud de ayuda a las demás máquinas para que descarguen el nuevo proceso en otra. En este caso, el emisor es quien toma la iniciativa, para localizar más ciclos de la CPU.

En la parte (b), una máquina inactiva o sobrecargada que tiene poco trabajo y está preparada para más. Su principal objetivo es localizar una máquina dispuesta a darle trabajo.

5. Algoritmos locales vs. Globales. Este aspecto se relaciona con lo que se denomina política de transferencia. Debido a que cuando se está a punto de generar un proceso, hay que tomar una decisión para ver si dicho proceso se ejecuta o no en la máquina que lo genera; si esta máquina está muy ocupada, hay que transferir el nuevo proceso a otro lugar. Para el caso del algoritmo local, se hace la verificación en la máquina, para ver si ésta se encuentra por debajo de su nivel, para poder asignar el nuevo proceso; para el algoritmo global es mejor recolectar toda la información acerca de la carga, antes de decidir si la máquina puede o no recibir al proceso.

Los algoritmos locales son sencillos, pero no óptimos, mientras que los globales sólo dan un resultado mejor pero a un costo mayor.

5. SISTEMAS DISTRIBUIDOS DE ARCHIVOS

El corazón de cualquier sistema distribuido es el sistema distribuido de archivos. Como en el caso de los sistemas con un único procesador, la tarea del sistema de archivos, en los sistemas distribuidos, es almacenar los programas y los datos y tenerlos disponibles cuando sea necesario. En el caso de un sistema distribuido, es importante distinguir entre los conceptos de servicio de archivos y el servidor de archivos. El **servicio de archivos** es la especificación de los servicios que el sistema ofrece a sus clientes. Describe las primitivas disponibles, los parámetros que utilizan y las acciones que llevan a cabo.

Para los clientes, el servicio de archivos define con precisión el servicio con que pueden contar, pero no dice nada con respecto a su implantación. De hecho el servicio de archivos especifica la interfaz del sistema de archivos con los clientes.

Por el contrario, un **despachador de archivos** es un proceso que se ejecuta en alguna máquina y ayuda con la implantación del servicio de archivos. Un sistema puede tener uno o varios servidores de archivos, pero en un sistema distribuido con un diseño adecuado los clientes no deben ser conscientes de la forma de implantar el sistema de archivos. En particular, no deben conocer el número de servidores de archivos, su posición o función. Todo lo que saben es que, al llamar los procedimientos especificados en el servicio de archivos, el trabajo necesario se lleva a cabo de alguna manera y se obtienen los resultados solicitados. De hecho, los clientes ni siquiera deben darse por enterados de que el servicio de archivos es distribuido. Lo ideal es que se vea como un sistema de archivos normal, de un único procesador.

Puesto que un servidor de archivos es un proceso del usuario que se ejecuta en una máquina, un sistema puede contener varios servidores de archivos, cada uno de los cuales ofrece un servicio de archivos distinto. De esa forma es posible que una terminal tenga varias ventanas y que en algunas de ellas se ejecuten programas en UNIX y en otras programas en MS-DOS, sin que esto provoque conflictos.

Los diseñadores del sistema se encargan de que los servidores ofrezcan los servicios de archivo específicos, como UNIX o MS-DOS. El tipo y el número de servicios de archivo disponibles puede cambiar con la evolución del sistema.

5.1. Diseño de los sistemas distribuidos de archivos

En esta sección analizaremos las características de los sistemas distribuidos de archivos, desde el punto de vista del usuario.

Un sistema distribuido tiene dos componentes razonablemente distintos: el verdadero servicio de archivos y el servicio de directorios. El servicio de archivos se encarga de las operaciones en los archivos individuales, como la lectura, la escritura y la adición, en tan-

to que el servicio de directorios se encarga de crear y manejar directorios, añadir y eliminar archivos de los directorios, etc.

5.1.1. *La interfaz del servicio de archivos*

En muchos sistemas, como UNIX y MS-DOS, un archivo es una secuencia de bits sin interpretación alguna. El significado y la estructura de la información en los archivos queda a cargo de los programas de aplicación; esto es irrelevante para el sistema operativo.

Sin embargo, en los "mainframes" existen muchos tipos de archivos, cada uno con distintas propiedades. Por ejemplo, un archivo se puede estructurar como una serie de registros, con llamadas al sistema operativo, para leer o escribir en un registro particular.

Por lo general se puede especificar el registro mediante su número o el valor de cierto campo. En el segundo caso, el sistema operativo mantiene el archivo como un árbol B o alguna otra estructura de datos adecuada, o bien utiliza tablas de dispersión, para localizar con rapidez los registros.

Debido a que la mayoría de los sistemas distribuidos son planeados para ambientes UNIX o MS-DOS, la mayoría de los servidores de archivos soporta el concepto de archivo, como una secuencia de bits, en vez, de una secuencia de registros con cierta clave. Los archivos pueden tener **atributos**, que son partes de información relativas al archivo, pero que no son parte del archivo propiamente dicho. Los atributos típicos son el propietario, el tamaño, la fecha de creación y el permiso de acceso.

Por lo general, el servicio de archivos proporciona primitivas para leer y escribir alguno de los atributos.

Otro aspecto importante del modelo de archivo es si los archivos se pueden modificar después de su creación. Lo normal es que sí se puedan modificar, pero en algunos sistemas distribuidos, las únicas operaciones de archivo son CREATE y READ. Una vez creado un archivo, éste no podrá ser modificado, y se dice que este archivo es **inmutable.**

La protección en los sistemas distribuidos utiliza, en esencia, las mismas técnicas de los sistemas con un único procesador: **posibilidades y listas para control de acceso.** En el caso de las posibilidades, cada usuario tiene un cierto tipo de boleto, llamado **posibilidad,** para cada objeto al que tiene acceso. La posibilidad determina los tipos de acceso permitidos.

Todos los esquemas de **lista para control de acceso** le asocian a cada archivo una lista implícita o explícita de los usuarios que pueden tener acceso al archivo y los tipos de acceso permitidos para cada uno de ellos.

El esquema de UNIX es una lista para control de acceso, simplificada con bits que controlan la lectura, la escritura y la ejecución de cada archivo, en forma independiente para el propietario, el grupo de propietarios, y todas las demás personas.

Los servicios de archivos se pueden dividir en dos tipos: la dosificación depende de si soportan un **modelo carga/descarga** o un **modelo de acceso remoto.** En el modelo de carga/descarga, el servicio de archivo sólo proporciona dos operaciones principales: la lectura de un archivo y la escritura en un archivo.

La operación de lectura transfiere todo un archivo, de uno de los servidores de archivo, al cliente solicitante.

La operación de escritura transfiere todo un archivo, en sentido contrario; del cliente al servidor. Así, el modelo conceptual es el traslado de archivos completos, en alguna de las direcciones. Los archivos se pueden almacenar en la memoria o en un disco local.

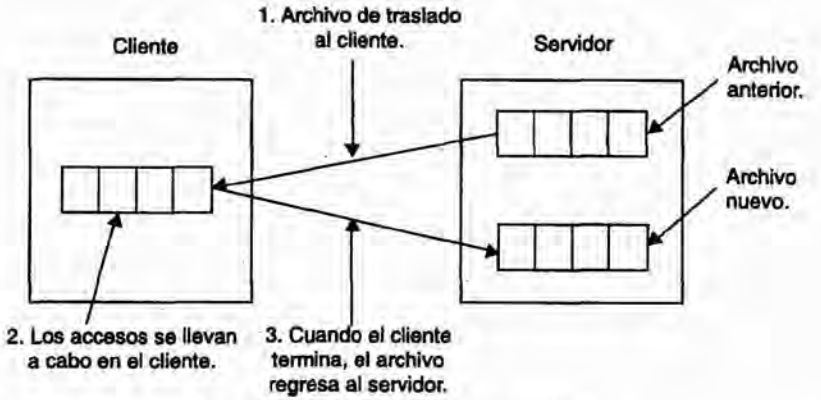


Figura 5.1. *El modelo carga/descarga.*

La ventaja de este modelo es la sencillez del concepto. Los programas de aplicación buscan los archivos que necesiten y después los utilizan de manera local. Los archivos modificados o nuevos se escriben, de regreso, al terminar el programa.

El otro tipo de servicio de archivos es el **modelo de acceso remoto.** En este modelo, el servicio de archivos proporciona un gran número de operaciones para abrir y cerrar archivos, leer y escribir partes de archivos, moverse a través de un archivo, examinar y modificar los atributos de un archivo, etc.

Mientras que en el modelo carga/descarga el servicio de archivos sólo proporciona el almacenamiento físico y la transferencia, en este caso el sistema de archivos se ejecuta en los servidores y no en los clientes. Su ventaja consiste en que no necesita mucho espacio por parte de los clientes, a la vez que elimina la necesidad de transferir archivos completos cuando sólo se necesita una parte de ellos.

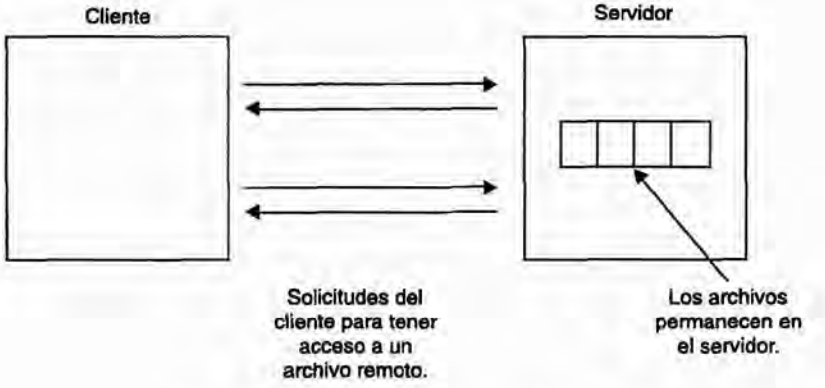


Figura 5.2. *El modelo de acceso remoto.*

5.1.2. *La interfaz del servidor de directorios*

La otra parte del servicio de archivos es el servicio de directorios, el cual proporciona las operaciones para crear y eliminar directorios, nombrar o cambiar el nombre de los archivos y mover éstos de un directorio a otro.

El servicio de directorios define un alfabeto y una sintaxis para formar los nombres de archivos y directorios. Lo común es que los nombres de archivos tengan de y hasta cierto número máximo de letras, números y ciertos caracteres especiales. Algunos sistemas dividen los nombres de archivo en dos partes, usualmente separadas mediante un punto. La segunda parte del nombre, llamada la extensión del archivo, identifica el tipo de éste. Otros sistemas utilizan un atributo explícito para este fin, en vez de utilizar una extensión dentro del nombre.

Todos los sistemas distribuidos permiten que los directorios contengan subdirectorios, para que los usuarios puedan agrupar los archivos relacionados entre sí. De acuerdo con esto, se dispone de operaciones para la creación y eliminación de directorios, así como para introducir, eliminar y buscar archivos en ellos. También, los subdirectorios pueden contener sus propios subdirectorios y así sucesivamente, lo que conduce a un árbol de directorios, el cual es conocido como **sistema jerárquico de archivos.** La Figura 5.3 muestra un árbol de cinco directorios:

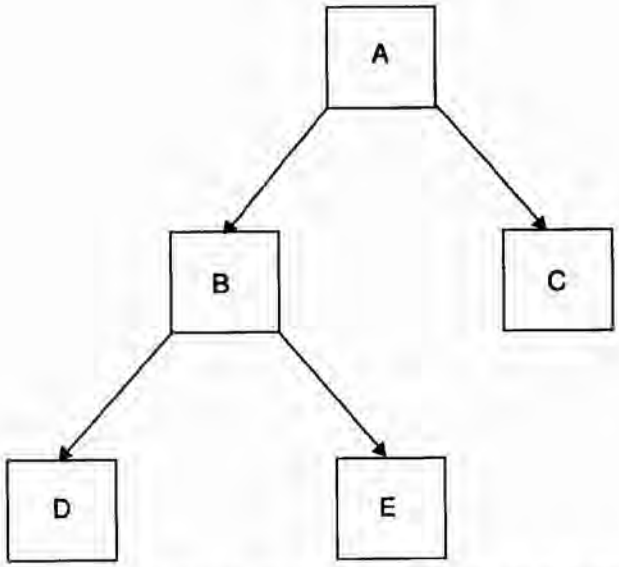


Figura 5.3. *Un árbol de directorios contenido en una máquina.*

En ciertos sistemas es posible crear enlaces o apuntadores hacia un directorio arbitrario. Estos se pueden colocar en cualquier directorio, lo que permite construir no sólo árboles sino gráficas arbitrarias de directorios, las cuales son más poderosas. La distinción entre árboles y gráficas es de particular importancia en un sistema distribuido.

La naturaleza de la dificultad se puede apreciar más claramente en la Figura 5.4. En esta gráfica, el directorio *D* tiene un enlace con el directorio *B*. El problema aparece cuando se elimina el enlace de *A* a *B*. En una jerarquía con estructura de árbol sólo se puede eliminar un enlace con un directorio, si el directorio al cual apunta es vacío. En una gráfica se permite la eliminación de un enlace mientras exista al menos otro. Mediante un contador de referencias, el cual se muestra en la esquina superior derecha de cada directorio de la figura, se puede determinar si el enlace por eliminar es el último. Después de eliminar el enlace de *A* a *B*, el contador de referencias de *B* se reduce de 2 a 1, lo cual está bien sobre el papel. Sin embargo, ahora no es posible llegar a *B* desde la raíz del sistema de archivos (A). Los tres directores *B*, *D* y *E* y todos sus archivos se convierten en huérfanos.

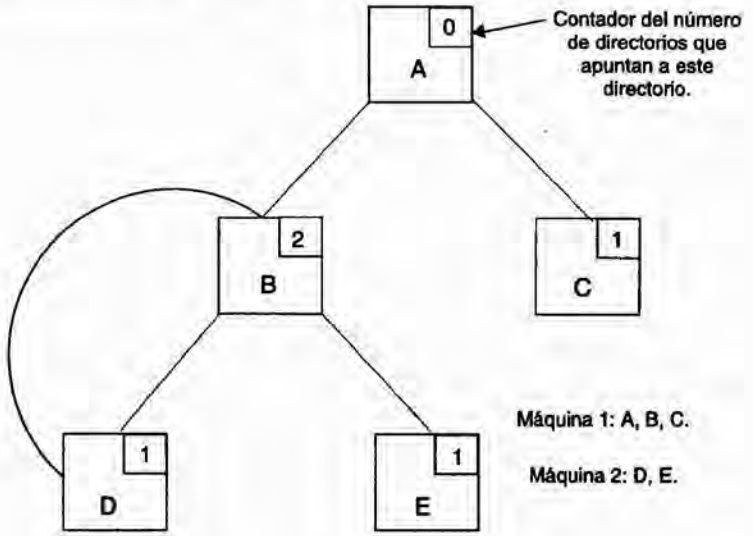


Figura 5.4. *Una gráfica de directorios en dos máquinas.*

Este problema también existe en los sistemas centralizados, pero es más serio en los distribuidos. Si todo está en una máquina, es posible descubrir los directorios huérfanos, puesto que toda la información está en un solo lugar. Se puede detener toda la actividad de los archivos y recorrer la gráfica desde la raíz, para señalar todos los directorios alcanzables. Al final de este proceso, se sabe que todos los directorios no marcados son inalcanzables. En un sistema distribuido existen varias máquinas y no se puede detener toda la actividad, por lo que es supremamente difícil tomar una foto instantánea.

Otro aspecto fundamental, en el diseño de cualquier sistema distribuido de archivos, es si todas las máquinas y procesos deben tener la misma visión de la jerarquía de los directorios.

5.1.2.1. Nombres de los archivos y directorios

Existen tres métodos usuales para nombrar los archivos y directorios en un sistema distribuido:

1. Nombre de la máquina + ruta de acceso, como */máquina/ruta* o *máquina:ruta.*
2. Montaje de sistemas de archivos remotos en la jerarquía local de archivos.

3. Un único espacio de nombres que tenga la misma apariencia en todas las máquinas.

Los primeros dos son fáciles de implantar, especialmente como una forma de conectar sistemas ya existentes, que no estaban diseñados para su uso distribuido. El tercer método es difícil y requiere de un diseño cuidadoso, pero es necesario si se quiere lograr el objetivo de que el sistema distribuido actúe como una única computadora.

5.1.2.2. Nombres de dos niveles

La mayoría de los sistemas distribuidos utiliza cierta forma de nombres con dos niveles. Los archivos tienen **nombres simbólicos** para el uso de las personas, pero también pueden tener **nombres binarios** internos para uso del propio sistema. Lo que los directorios hacen, en realidad, es proporcionar una asociación entre estos dos nombres. Para las personas y los programas, es conveniente utilizar nombres simbólicos, pero para el uso dentro del propio sistema, estos nombres son demasiado grandes y difíciles. Así, cuando un usuario abre un archivo o hace referencia a un nombre simbólico, el sistema busca de inmediato el nombre simbólico, en el directorio apropiado, para obtener el nombre binario, el cual utilizará para localizar realmente el archivo. A veces los nombres binarios son visibles a los usuarios y a veces no.

Un esquema más general para los nombres es que el nombre binario indique el servidor y un archivo específico en ese servidor. Este método permite que un directorio en un servidor contenga un archivo en un servidor distinto.

Otra alternativa, que a veces es preferible, es utilizar un **enlace simbólico,** el cual es una entrada de directorio asociada a una cadena (servidor, nombre de archivo), la cual se puede buscar en el servidor correspondiente para encontrar el nombre binario. El propio enlace simbólico es sólo el nombre de una ruta de acceso.

Otra idea más es utilizar las posibilidades como los nombres binarios. En este método, la búsqueda de un nombre en ASCII produce una posibilidad, la cual puede tomar varias formas. Un último método, que a veces está presente en un sistema distribuido pero casi nunca en uno centralizado, es la posibilidad de buscar un nombre en ASCII y obtener no uno sino varios nombres binarios. Por lo general, éstos representan al archivo original y todos sus respaldos.

5.1.3. *Semántica de los archivos compartidos*

Si dos o más usuarios comparten el mismo archivo, es necesario definir con precisión la semántica de la lectura y la escritura para evitar problemas.

Existen cuatro maneras de compartir archivos en un sistema distribuido. La primera que analizaremos es conocida como **semántica de UNIX** y consiste en que cada operación en un archivo es visible a todos los procesos de manera instantánea. En los sistemas con un único procesador que permiten a los procesos compartir archivos, como UNIX, la semántica establece que si una operación READ sigue después de una operación WRITE, READ regresa el valor recién escrito. De manera análoga, cuando dos WRITE se realizan en serie y después se ejecuta un READ, el valor que se lee es el almacenado en la última escritura. De hecho, el sistema impone en todas las operaciones un orden absoluto con respecto del tiempo y siempre regresa el valor más reciente. En un sistema con un único procesador, esto es muy fácil de comprender y tiene una implantación directa. En un sistema distribuido, la semántica de UNIX se puede lograr fácilmente, mientras sólo

exista un servidor de archivos y los clientes no oculten los archivos. Todas las instrucciones READ y WRITE pasan en forma directa al servidor de archivos, que los procesa en forma secuencial. Este método proporciona la semántica de UNIX, excepto por un problema menor: los retrasos en la red pueden hacer que un READ, ocurrido un microsegundo después de un WRITE, llegue primero al servidor y que obtenga el valor anterior.

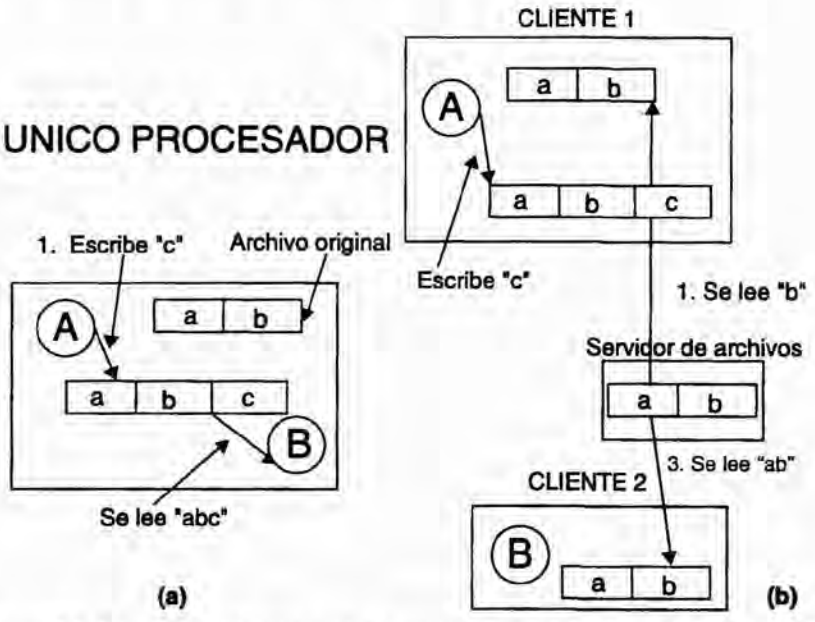


Figura 5.5 (a). *En un solo procesador, cuando una READ va después de un WRITE, el valor regresado por READ es el valor recién escrito.*

(b) *En un sistema distribuido con ocultamiento, el valor regresado puede ser obsoleto.*

El segundo método se llama **semántica de sesión,** el cual se caracteriza porque ningún cambio es visible a otros procesos, hasta que el archivo se cierra; en vez de pedir que un READ vea los efectos de todos los WRITE anteriores, uno puede tener una nueva regla que diga: "Los cambios a un archivo abierto sólo pueden ser vistos en un principio por el proceso (o la máquina) que modificó el archivo. Los cambios serán visibles a los demás procesos (o máquinas) sólo cuando se cierre el archivo".

Un método completamente distinto de la semántica de los archivos compartidos, en un sistema distribuido, es que todos los archivos sean **inmutables.** Así no existe forma de abrir un archivo para escribir en él. En efecto, las únicas operaciones en los archivos son CREATE y READ. No existen actualizaciones; es más fácil compartir y replicar.

Una cuarta vía para enfrentar el uso de los archivos compartidos en un sistema distribuido es usar las transacciones atómicas. Para tener acceso a un archivo o grupo de archivos, un proceso

ejecuta en primer lugar cierto tipo de primitiva BEGINTRANSACTION, para señalar que lo que sigue debe ejecutarse de manera indivisible. Después vienen las llamadas al sistema para leer o escribir en uno o más archivos. Al terminar el trabajo, se ejecuta una primitiva ENDTRANSACTION. La propiedad fundamental de este método es que el sistema garantiza que todas las llamadas contenidas dentro de la transacción se llevarán a cabo en orden, sin interferencias de otras transacciones concurrentes. Si dos o más transacciones se realizan al mismo tiempo, el sistema garantiza que el resultado final es el mismo que si se ejecutasen en cierto orden secuencial (indeterminado).

5.2. Implantación de un sistema distribuido de archivos

En la sección anterior describimos varios aspectos de los sistemas distribuidos de archivos desde el punto de vista del usuario. En esta sección veremos la forma en que se implantan dichos sistemas.

5.2.1. *Uso de archivos*

Satyanarayanan (1981) hizo un amplio estudio sobre los patrones de uso de los archivos. Vamos a presentar, a continuación, las principales conclusiones de este estudio. Para comenzar, la mayoría de los archivos está por debajo de los 10K; esta observación sugiere la factibilidad de la transferencia de archivos completos, en vez de bloques de disco, entre el servidor y el cliente. Puesto que la transferencia de archivos completos es más sencilla y eficiente, esta idea debe tomarse en cuenta. Por supuesto, algunos archivos son de gran tamaño, por lo que también se deben tomar medidas preventivas con respecto a éstos.

Una observación interesante es que la mayoría de los archivos tiene tiempos de vida cortos. En otras palabras, un patrón común es crear un archivo, leerlo y después eliminarlo.

El hecho que unos cuantos archivos se compartan, argumenta en favor del ocultamiento por parte del cliente. Como ya se ha visto, el ocultamiento complica la semántica, pero si es raro el uso de los archivos compartidos, podría ser mejor el ocultamiento por parte del cliente y aceptar las consecuencias de la semántica de sesión en favor de un mejor desempeño.

Para terminar, la existencia de distintas clases de archivos sugiere que, tal vez, se deberían utilizar mecanismos diferentes para el manejo de las distintas clases. Los binarios del sistema necesitan estar presentes en diferentes partes, pero es raro que se modifiquen, por lo que tal vez se podrían duplicar en varias partes, aun cuando esto implica una actualización ocasional compleja.

Los compiladores y los archivos temporales son cortos, no compartidos y desaparecen con rapidez, por lo que deben mantener su carácter local mientras sea posible. Los buzones electrónicos se actualizan con frecuencia, pero es raro que se compartan, por lo que su duplicación no sirve de mucho.

5.2.2. *Estructura del sistema*

En esta sección analizaremos algunas de las formas de organización interna de los servidores de archivos y directorios, con atención especial a los métodos alternativos. Comenzaremos con una sencilla pregunta:"¿Son distintos los clientes y los servidores?"

En ciertos sistemas no existe distinción alguna entre un cliente y un servidor. Todas las máquinas ejecutan el mismo software básico, de manera que una máquina que desee dar servicio de archivos al público en general es libre de hacerlo. Este ofrecimiento del servicio de archivos consiste sólo en exportar los nombres de los directorios seleccionados, de modo que otras máquinas tengan acceso a ellos. En otros sistemas el servidor de archivos y el de directorios sólo son programas del usuario, por lo que se puede configurar un sistema para que ejecute o no el software, del cliente o del servidor, en la misma máquina, como se desee. Y en el otro extremo, están los sistemas donde los clientes y los servidores son máquinas esencialmente distintas, ya sea en términos de hardware o de software. Los servidores y clientes pueden ejecutar incluso versiones diferentes del sistema operativo. Aunque la separación de funciones es un poco más transparente, no existe una razón fundamental para preferir un método por encima de los demás.

Un segundo aspecto de la implantación, en donde difieren los sistemas, es la forma de estructurar el servicio a archivos y directorios. Una forma de organización consiste en combinar ambos en un único servidor, que maneje todas las llamadas a directorios y archivos. Sin embargo, la otra posibilidad es separarlos. En este caso, la apertura de un archivo exige ir hasta el servidor de directorios, para asociar su nombre simbólico con el nombre binario, y después ir hasta el servidor de archivos, con el nombre en binario y llevar a cabo la lectura o escritura real del archivo.

El argumento en favor de la separación es que las dos funciones no tienen una relación real entre sí, por lo que es más flexible mantenerlas separadas. Por ejemplo, se puede implantar un servidor de directorios en MS-DOS y otro servidor de directorios en UNIX, donde ambos utilicen el mismo servidor de archivos para el almacenamiento físico. También es probable que la separación de funciones produzca un software más sencillo. Un contraargumento es que el hecho de contar con dos servidores requiere de una mayor comunicación.

El aspecto estructural final por considerar es si los servidores de archivos, directorios o de otro tipo deben contener la información sobre el estado de los clientes. Este aspecto tiene una controversia moderada, donde existen dos escuelas de pensamiento en competencia. Una escuela piensa que los servidores no deben contener los estados, es decir, "ser sin estado". En otras palabras, cuando un cliente envía una solicitud a un servidor, éste la lleva a cabo, envía la respuesta y elimina de sus tablas internas toda la información relativa a dicha solicitud. El servidor no guarda información alguna relativa a los clientes entre las solicitudes. La otra escuela de pensamiento sostiene que es correcto que los servidores conserven la información de estado de los clientes entre las solicitudes.

En el caso de un servidor sin estado, cada solicitud debe estar autocontenida. Debe contener todo el nombre del archivo y el ajuste dentro de éste, para que el servidor pueda realizar el trabajo. Esta información aumenta la longitud del mensaje.

Otra forma de ver la información de estado es considerar lo que ocurre si un servidor falla y todas sus tablas se pierden de manera irremediable. Al volver a arrancar el servidor, éste ya no tiene idea de la relación entre los clientes y los archivos abiertos por éstos. Fracasarán entonces los intentos posteriores por leer y escribir en archivos abiertos y la recuperación, entonces, queda a cargo totalmente de los clientes. En consecuencia, los servidores sin estado tienden a ser más tolerantes a los fallos que los que mantienen los estados, lo cual es un argumento a favor de los primeros.

En general, las ventajas de los servidores sin estado son: tolerancia a las fallas; no necesita llamadas open/close; no se desperdicia el espacio del servi-

dor en las tablas; no existe límite para el número de archivos abiertos y no hay problemas si un cliente falla. Las ventajas de los servidores con estado consisten en un mejor desempeño, los mensajes de solicitud más cortos, es posible la lectura adelantada, es más fácil la idempotencia y es posible la cerradura de archivos.

5.2.3. *Ocultamiento*

En un sistema cliente-servidor, cada uno con su memoria principal, y un disco, existen cuatro lugares donde se pueden almacenar los archivos o partes de archivos: el disco del servidor, la memoria principal del servidor, el disco del cliente o la memoria principal del cliente, como se muestra en la Figura 5.6.

El lugar más directo para almacenar todos los archivos es el disco del servidor. Ahí existe mucho espacio y los archivos serían accesibles a todos los clientes. Además, con sólo una copia de cada archivo no surgen problemas de consistencia.

El problema con el uso del disco del servidor es el desempeño. Antes de que un cliente pueda leer un archivo, éste debe ser transferido primero del disco del servidor a la memoria principal del servidor y luego, a través de la red, a la memoria principal del cliente. Ambas transferencias tardan cierto tiempo.

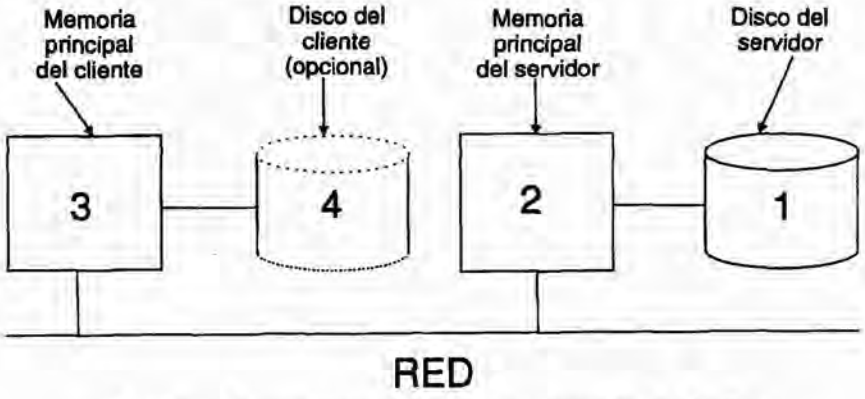


Figura 5.6. *Cuatro lugares para guardar archivos o partes de ellos.*

Se puede lograr un desempeño mucho mejor si se ocultan (es decir, se conservan) los archivos de más reciente uso en la memoria principal del servidor. Un cliente que lea un archivo ya presente en el caché del servidor elimina la transferencia del disco, aunque se deba realizar la transferencia a la red.

Puesto que la memoria principal siempre es menor que el disco, se necesita un algoritmo para determinar los archivos o partes de archivos que deben permanecer en el caché.

Este algoritmo debe resolver dos problemas. El primero es el de la unidad que maneja el caché. Puede manejar archivos completos o bloques del disco. Si se ocultan los archivos completos, éstos se pueden almacenar en forma adyacente en el disco, lo cual permite transferencias a alta velocidad entre la memoria y el disco, así como un buen desempeño en general. Sin embargo, el ocultamiento de bloques de disco utiliza el caché y el espacio en disco en forma más eficiente.

El segundo es que el algoritmo debe decidir qué hacer si se utiliza toda la capacidad del caché y hay que eliminar a alguien. Aquí se puede utilizar cualquiera de los algoritmos comunes de ocultamiento, pero como las referencias al caché son poco frecuentes, comparadas con las referencias a la memoria, por lo general es factible una implantación exacta de LRU mediante listas ligadas: Cuando hay que eliminar a alguien de la memoria, se elige al más antiguo. Si existe una copia actualizada en el disco, simplemente se descarta la copia del caché. En caso contrario, primero se actualiza el disco.

El mantenimiento de un caché en la memoria principal del servidor es fácil de lograr y es totalmente transparente a los clientes. Puesto que el servidor puede mantener sincronizadas sus copias en memoria y en disco, desde el punto de vista de los clientes sólo existe una copia de cada archivo, por lo que no hay problemas de consistencia.

5.2.4. *Réplica*

Con frecuencia los sistemas distribuidos de archivos proporcionan la réplica de archivos como servicio a sus clientes. En otras palabras, se dispone de varias copias de algunos archivos, donde cada copia está en un servidor de archivos independiente. Las principales razones para la existencia de tal servicio son:

1. Aumentar la confiabilidad al disponer de respaldos independientes de cada archivo. Si un servidor falla o se pierde en forma permanente, no se pierden los datos.
2. Permitir el acceso a un archivo, aun cuando falle un servidor de archivos. El fallo de un servidor no debe hacer que todo el sistema se detenga hasta cuando aquél se pueda volver a arrancar.
3. Repartir la carga de trabajo entre varios servidores. Al crecer el tamaño del sistema, el hecho de tener todos los archivos en un único procesador se puede convertir en un cuello de botella. Con varios archivos duplicados en dos o más servidores, se puede utilizar el que tenga menos carga.

Las dos primeras se relacionan con el mejoramiento de la confiabilidad y la disponibilidad; la tercera se refiere al desempeño.

Existen tres formas de llevar a cabo la réplica: la réplica explícita de archivos, la réplica retrasada de archivos y la réplica de archivos mediante un grupo.

CONCLUSIONES

- Los Sistemas Distribuidos son la tecnología que se va a imponer en el mundo, debido a la economía que se alcanza con su implantación en las grandes organizaciones.
- No existen todavía aplicaciones distribuidas para los pocos sistemas operativos que tratan de manejar la distribución, como Windows NT.
- La diferencia más importante entre un Sistema Distribuido y un sistema de un único procesador es la comunicación entre procesos.
- Para la organización de los procesadores se pueden tener en cuenta dos modelos básicos: el modelo de estación de trabajo y el de la pila de procesadores. Las ventajas que presenta cada uno son muy importantes, ya que le permiten tener a cada usuario su propia estación de trabajo y aprovechar las estaciones de trabajo inactivas para ejecutar procesos; en las pilas de procesadores, todas las instalaciones son un recurso compartido que puede ser usado dependiendo de las necesidades de cada usuario.

- Cuando se tiene una colección de procesadores para que éstos sean asignados a los procesos, se debe tener la ayuda de los algoritmos de asignación de procesadores.
- El sistema distribuido de archivos es el corazón de cualquier Sistema Distribuido y su tarea consiste en almacenar los programas y los datos y tenerlos disponibles cuando sea necesario.
- La implantación de un sistema distribuido de archivos implica la toma de varias decisiones: ver si el sistema es con estado o sin él, si se debe hacer el ocultamiento y la forma de manejar la réplica de archivos.
- Definitivamente nos encontramos en la era de las comunicaciones y, debido a que toda actividad administrativa se lleva a cabo por medio de comunicaciones orales o escritas, en muchos casos entre interlocutores que se encuentran a grandes distancias, se hace necesario contar con redes de terminales inteligentes que permitan a dichos interlocutores comunicarse en forma confiable y poder compartir la información de manera transparente.

BIBLIOGRAFIA

- *Sistemas operativos modernos,* Andrew S. Tanembaum. Editorial Prentice-Hall Hispanoamericana S.A.; México, 1992.
- *Internet-Working With TCP/IP.* Volumen III. Client-Server Programming and Applications.
  Douglas E. Corner and David L. Stevens. Department of Computer Sciences. Purdue University.
  Prentice-Hall Internacional, Inc.

LA ULTIMA LECCIÓN*

DISCURSO DE GRADO
Promoción Decimosegunda
Cali, 11 de febrero de 1995

ALFONSO OCAMPO LONDOÑO
Rector del ICESI

Volvemos hoy con gran alegría, pero al mismo tiempo con nostalgia, a graduar a 154 de nuestros alumnos, 95 en Pregrado en Administración de Empresas e Ingeniería de Sistemas e Informática y 59 de Postgrado, y entre éstos la primera promoción de la especialización en Ingeniería del Software. Con ellos ya conformamos un cuerpo de cerca de 3.000 egresados que hemos entregado a la sociedad, a las empresas, a sus compañías familiares y las propias.

Hemos, desde que ustedes iniciaron sus estudios, recalcado que es un gran privilegio llegar a una Universidad, ya que sólo dos de cada cien que inician la primaria lo logran y es aún mayor privilegio el de hacer formación avanzada o de postgrado. Se tiene, pues, una deuda social que tiene que pagarse, a sus padres, familiares, amigos, patrones, empresas, a quienes les han servido, y en general a la sociedad y al Estado que les han dado las condiciones o infraestructura social para poder llegar a esta importante etapa de sus vidas. Esta deuda la deben pagar creando riqueza en Colombia y participando en la mejoría de sus condiciones de competitividad en un mundo que se ha globalizado o internacionalizado y en el cual la competencia no va a ser sólo en productos, sino en la calidad del talento humano que se tiene y que es nuestra tarea. Por ello, en este acto, en un punto especial y muy emotivo, los graduandos le hacen un homenaje de gratitud a sus padres, cónyuges y amigos, que los han ayudado.

En el mundo de hoy, lo que se considera la mayor riqueza es el conocimiento y la información y por lo tanto la formación del ser humano, es la tarea más importante para hacer. Las riquezas naturales contribuyen al bienestar, solamente si hay un equipo humano que las sepa aprovechar, para que sus beneficios se extiendan a todos y no sólo a unos pocos. Lo que debe importar es el beneficio para la totalidad de la comunidad y no el aprovechar todo, sólo

* Este discurso continúa una tradición universitaria en que el rector da la primera y la última lección.