Introducción a los Sistemas Distribuidos

Introducción a los Sistemas Distribuidos

Desde el *inicio de la era de la computadora moderna* (1945), hasta cerca de 1985, solo se conocía la *computación centralizada [25, Tanenbaum].*

A partir de la *mitad de la década de los ochentas* aparecen dos avances tecnológicos fundamentales:

- Desarrollo de **microprocesadores** poderosos y económicos con arquitecturas de 8, 16, 32 y 64 bits.
- Desarrollo de **redes de área local** *(LAN)* de alta velocidad, con posibilidad de conectar cientos de máquinas a velocidades de transferencia de millones de bits por segundo (mb/seg).

Aparecen los **sistemas distribuidos**, en contraste con los *sistemas centralizados.*

Los sistemas distribuidos necesitan un software distinto al de los sistemas centralizados.

Los S. O. para sistemas distribuidos han tenido importantes desarrollos pero todavía existe un largo camino por recorrer.

Los usuarios pueden acceder a una *gran variedad de recursos computacionales:*

- De hardware y de software.

- Distribuidos entre un gran número de sistemas computacionales conectados.

Un importante antecedente de las redes de computadoras lo constituye Arpanet, iniciada en 1968 en los EE. UU.

Ventajas de los Sistemas Distribuidos con Respecto a los Centralizados

Una razón para la tendencia hacia la descentralización es la *economía.*

Herb Grosch formuló la que se llamaría *"Ley de Grosch" [25, Tanenbaum]:*

- El poder de cómputo de una cpu es proporcional al cuadrado de su precio:
  - Si se paga el doble se obtiene el cuádruple del desempeño.
- Fue aplicable en los años setentas y ochentas a la tecnología mainframe.
- No es aplicable a la tecnología del microprocesador:
  - La solución más eficaz en cuanto a costo es limitarse a un gran número de cpu baratos reunidos en un mismo sistema.

Los sistemas distribuidos generalmente tienen en potencia una proporción precio / desempeño mucho mejor que la de un único sistema centralizado.

Algunos autores distinguen entre:

- **Sistemas distribuidos:** están diseñados para que *muchos usuarios trabajen en forma conjunta.*
- **Sistemas paralelos:** están diseñados para lograr la *máxima rapidez en un único problema.*

En general se consideran *sistemas distribuidos, en sentido amplio,* a los sistemas en que:

- Existen *varias cpu conectadas* entre sí.
- Las distintas cpu *trabajan de manera conjunta.*

Ciertas *aplicaciones* son distribuidas en forma inherente:

- Ej.: sistema de automatización de una fábrica:
  - Controla los robots y máquinas en la línea de montaje.
  - Cada robot o máquina es controlado por su propia computadora.
  - Las distintas computadoras están interconectadas.

Una *ventaja potencial de un sistema distribuido* es una mayor *confiabilidad:*

- Al distribuir la carga de trabajo en muchas máquinas, la falla de una de ellas no afectara a las demás:
  - La carga de trabajo podría distribuirse.
- Si una máquina se descompone:
  - Sobrevive el sistema como un todo.

Otra ventaja importante es la posibilidad del *crecimiento incremental* o por incrementos:

- Podrían añadirse procesadores al sistema, permitiendo un desarrollo gradual según las necesidades.
- No son necesarios grandes incrementos de potencia en breves lapsos de tiempo.
- Se puede añadir poder de cómputo en pequeños incrementos.

Ventajas de los Sistemas Distribuidos con Respecto a las PC Independientes

Satisfacen la necesidad de muchos usuarios de *compartir ciertos datos* [25, Tanenbaum] :

- Ej.: sistema de reservas de líneas aéreas.

También con los sistemas distribuidos se pueden *compartir otros recursos como programas y periféricos costosos:*

- Ej.: impresoras láser color, equipos de fotocomposición, dispositivos de almacenamiento masivo (ej.: cajas ópticas), etc.

Otra importante razón es lograr una *mejor comunicación entre las personas:*

- Ej.: correo electrónico:
  - Posee importantes ventajas sobre el correo por cartas, el teléfono y el fax:
    - Velocidad, disponibilidad, generación de documentos editables por procesadores de texto, etc.

La *mayor flexibilidad* es también importante:

- La carga de trabajo se puede difundir (distribuir) entre las máquinas disponibles en la forma más eficaz según el criterio adoptado (por ej. costos).
- Los equipos distribuidos pueden no ser siempre PC:
  - Se pueden estructurar sistemas con grupos de PC y de computadoras compartidas, de distinta capacidad.

Desventajas de los Sistemas Distribuidos

El principal problema es el software, ya que el diseño, implantación y uso del software distribuido presenta numerosos inconvenientes [25, Tanenbaum].

Los *principales interrogantes* son los siguientes:

- ¿Qué tipo de S. O., lenguaje de programación y aplicaciones son adecuados para estos sistemas?.
- ¿Cuánto deben saber los usuarios de la distribución?.
- ¿Qué tanto debe hacer el sistema y qué tanto deben hacer los usuarios?.

La *respuesta a estos interrogantes no es uniforme* entre los especialistas, pues existe una gran *diversidad de criterios* y de *interpretaciones* al respecto.

Otro problema potencial tiene que ver con las *redes de comunicaciones,* ya que se deben considerar problemas debidos a pérdidas de mensajes, saturación en el tráfico, expansión, etc.

El hecho de que sea fácil *compartir los datos* es una ventaja pero se puede convertir en un gran problema, por lo que la seguridad debe organizarse adecuadamente.

En general se considera que las ventajas superan a las desventajas, si estas últimas se administran seriamente.

Conceptos de Hardware

Todos los sistemas distribuidos constan de varias cpu, organizadas de diversas formas, especialmente respecto de [25, Tanenbaum]:

- La *forma de interconectarlas* entre sí.
- Los *esquemas de comunicación* utilizados.

Existen diversos *esquemas de clasificación* para los sistemas de cómputos con varias cpu:

- Uno de los mas conocidos es la *"Taxonomía de Flynn":*
  - Considera como características esenciales el *número de flujo de instrucciones* y el *número de flujos de datos.*
  - La clasificación incluye equipos **SISD, SIMD, MISD** y **MIMD.**

SISD (Single Instruction Single Data: un flujo de instrucciones y un flujo de datos):

- Poseen un único procesador.

SIMD (Single Instruction Multiple Data: un flujo de instrucciones y varios flujos de datos):

- Se refiere a ordenar procesadores con una unidad de instrucción que:
  - Busca una instrucción.
  - Instruye a varias unidades de datos para que la lleven a cabo en paralelo, cada una con sus propios datos.
- Son útiles para los cómputos que repiten los mismos cálculos en varios conjuntos de datos.

MISD (Multiple Instruction Single Data: un flujo de varias instrucciones y un solo flujo de datos):

- No se presenta en la práctica.

MIMD (Multiple Instruction Multiple Data: un grupo de computadoras independientes, cada una con su propio contador del programa, programa y datos):

- **Todos los sistemas distribuidos son de este tipo.**

Un avance sobre la clasificación de Flynn incluye la división de las computadoras *MIMD* en dos grupos:

- *Multiprocesadores:* poseen memoria compartida:
  - Los distintos procesadores comparten el mismo espacio de direcciones virtuales.
- *Multicomputadoras:* no poseen memoria compartida:
  - Ej.: grupo de PC conectadas mediante una red.

Cada una de las categorías indicadas se puede *clasificar según la arquitectura de la red de interconexión* en:

- *Esquema de bus:*
  - Existe una sola red, bus, cable u otro medio que conecta todas las máquinas:
    - Ej.: la televisión por cable.
- *Esquema con conmutador:*
  - No existe una sola columna vertebral de conexión:
    - Hay múltiples conexiones y varios patrones de conexionado.
    - Los mensajes de mueven a través de los medios de conexión.
    - Se decide explícitamente la conmutación en cada etapa para dirigir el mensaje a lo largo de uno de los cables de salida.
    - Ej.: el sistema mundial telefónico público.
- Otro aspecto de la clasificación considera el *acoplamiento entre los equipos:*
- *Sistemas fuertemente acoplados:*
  - El retraso al enviar un mensaje de una computadora a otra es corto y la tasa de transmisión es alta.
  - Generalmente se los utiliza como sistemas paralelos.
- *Sistemas débilmente acoplados:*
  - El retraso de los mensajes entre las máquinas es grande y la tasa de transmisión es baja.
  - Generalmente se los utiliza como sistemas distribuidos.

Generalmente los multiprocesadores están más fuertemente acoplados que las multicomputadoras.

Multiprocesadores con Base en Buses

Constan de cierto número de cpu conectadas a un bus común, junto con un módulo de memoria (ver Figura 7.1 [25, Tanenbaum]).

Figura 7.1: Multiprocesadores con base en un bus.

Un bus típico posee al menos [25, Tanenbaum]:

- 32 líneas de direcciones.
- 32 líneas de datos.
- 30 líneas de control.

Todos los elementos precedentes *operan en paralelo.*

Para *leer* una palabra de memoria, una cpu:

- Coloca la dirección de la palabra deseada en las líneas de direcciones del bus.
- Coloca una señal en las líneas de control adecuadas 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 esta por parte de la cpu solicitante.

Para *grabar* el procedimiento es similar.

Solo existe una memoria, la cual presenta la propiedad de la ***coherencia:***

- Las modificaciones hechas por una cpu se reflejan de inmediato en las subsiguientes lecturas de la misma o de otra cpu.

El problema de este esquema es que *el bus tiende a sobrecargarse* y el rendimiento a disminuir drásticamente; la solución es añadir una *memoria caché de alta velocidad entre la cpu y el bus:*

- El caché guarda las palabras de acceso reciente.
- Todas las solicitudes de la memoria pasan a través del caché.
- Si la palabra solicitada se encuentra en el caché:
  - El caché responde a la cpu.
  - No se hace solicitud alguna al bus.
- Si el caché es lo bastante grande:
  - La *"tasa de encuentros"* será alta y la cantidad de tráfico en el bus por cada cpu disminuirá drásticamente.
  - Permite *incrementar el número de cpu.*

Un importante problema debido al uso de cachés es el de la *"incoherencia de la memoria":*

- Supongamos que las cpu *"A"* y *"B"* leen la misma palabra de memoria en sus respectivos cachés.

- *"A"* escribe sobre la palabra.
- Cuando *"B"* lee esa palabra, obtiene un valor anterior y no el valor recién actualizado por *"A"*.

Una solución consiste en lo siguiente:

- Diseñar las caché de tal forma que *cuando una palabra sea escrita al caché, también sea escrita a la memoria.*
- A esto se denomina *"caché de escritura"*.
- No causa tráfico en el bus el uso de *"caché para la lectura"*.
- Sí causa tráfico en el bus:
  - El no uso de caché para la lectura.
  - Toda la escritura.

Si *todos los cachés* realizan un monitoreo constante del bus:

- Cada vez que un caché observa una escritura a una dirección de memoria presente en él, puede eliminar ese dato o actualizarlo en el caché con el nuevo valor.
- Estos cachés se denominan *"cachés monitores"*.

Un diseño con cachés monitores y de escritura es coherente e invisible para el programador, por lo que es muy utilizado en multiprocesadores basados en buses.

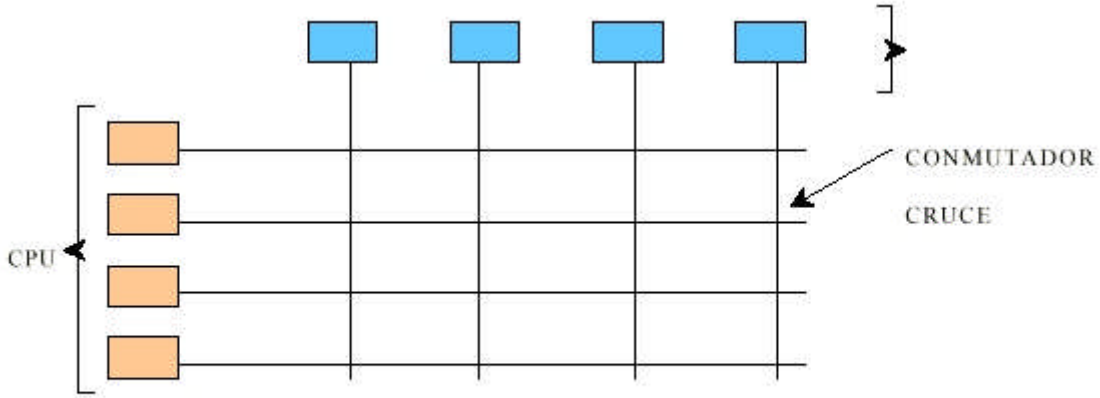
Multiprocesadores con Conmutador

El esquema de *multiprocesadores con base en buses* resulta apropiado para *hasta aproximadamente 64 procesadores [25, Tanenbaum].*

Para superar esta cifra es necesario un *método distinto de conexión* entre procesadores (cpu) y memoria.

Una posibilidad es dividir la memoria en módulos y conectarlos a las cpu con un *"conmutador de cruceta" (cross-bar switch):*

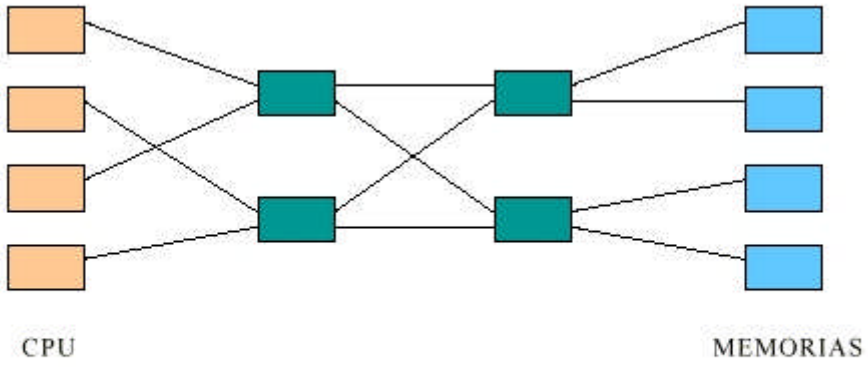
- Cada cpu y cada memoria tiene una conexión que sale de él.
- En cada intersección está un *"conmutador del punto de cruce" (crosspoint switch)* electrónico que el *hardware* puede abrir y cerrar:
  - Cuando una cpu desea tener acceso a una memoria particular, el conmutador del punto de cruce que los conecta se cierra momentáneamente.
- La virtud del conmutador de cruceta es que *muchas cpu pueden tener acceso a la memoria al mismo tiempo:*
  - Aunque no a la misma memoria simultáneamente.
- Lo *negativo* de este esquema es el *alto número de conmutadores:*
  - Para *"n"* cpu y *"n"* memorias se necesitan *"n"* x *"n"* conmutadores (ver Figura 7.2 [25, Tanenbaum]).



Figura˜7.2: Conmutador de cruceta.

El número de conmutadores del esquema anterior puede resultar prohibitivo:

- Otros esquemas precisan *menos conmutadores*, por ej., la *"red omega"* (ver Figura 7.3 [25, Tanenbaum]):
  - Posee conmutadores 2 x 2:
    - Cada uno tiene 2 entradas y 2 salidas.
    - Cada conmutador puede dirigir cualquiera de las entradas en cualquiera de las salidas.
    - Eligiendo los estados adecuados de los conmutadores, cada cpu podrá tener acceso a cada memoria.
  - Para *"n" cpu y "n" memorias* se precisan:
    - *"n"* etapas de conmutación.
    - Cada etapa tiene $log_2 n$ conmutadores para un total de $n \log_2 n$ conmutadores; este número es menor que *"n" x "n"* del esquema anterior, pero *sigue siendo muy grande* para *"n"* grande (ver Tabla 7.1 y Figura 7.4).



Figura˜7.3: Red omega de conmutación.

n	$\log_2 n$	$n * \log_2 n$	n * n
50	5,64385619	282	2.500
75	6,22881869	467	5.625
100	6,64385619	664	10.000
125	6,96578428	871	15.625
150	7,22881869	1.084	22.500
175	7,45121111	1.304	30.625
200	7,64385619	1.529	40.000
1.024	10	10.240	1.048.576

Tabla 7.1: Conmutador de cruceta versus red omega.

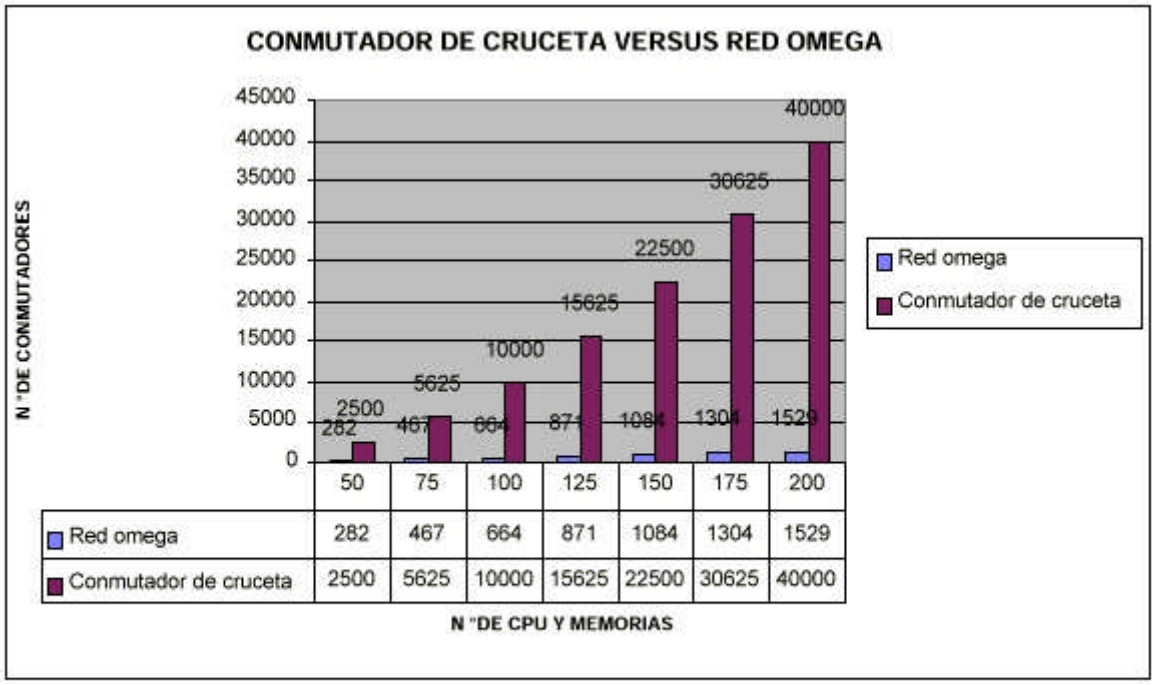


Figura 7.4: Conmutador de cruceta versus red omega.

Un *problema importante en la red omega* es el retraso:

- Ej.: si *"n"* = 1024 existen según la tabla anterior:
  - 10 etapas de conmutación de la cpu a la memoria.
  - 10 etapas para que la palabra solicitada de la memoria regrese.
  - Si la cpu es de 50 mhz, el tiempo de ejecución de una instrucción es de 20 nseg.

    - Si una solicitud de la memoria debe recorrer 20 etapas de conmutación (10 de ida y 10 de regreso) en 20 nseg:
        - El tiempo de conmutación debe ser de 1 nseg.
        - El multiprocesador de 1024 cpu necesitará 10240 conmutadores de 1 nseg.
        - El costo será alto.

Otra *posible solución* son los esquemas según *sistemas jerárquicos:*

- Cada cpu tiene asociada cierta *memoria local.*
- El acceso será muy rápido a la propia memoria local y más lento a la memoria de las demás cpu.
- Esto se denomina esquema o *"máquina NUMA"* (Acceso No Uniforme a la Memoria):
    - Tienen un mejor tiempo promedio de acceso que las máquinas basadas en redes omega.
    - La colocación de los programas y datos en memoria es crítica para *lograr que la mayoría de los accesos sean a la memoria local de cada cpu.*

Multicomputadoras con Base en Buses

Es un esquema *sin memoria compartida [25, Tanenbaum].*

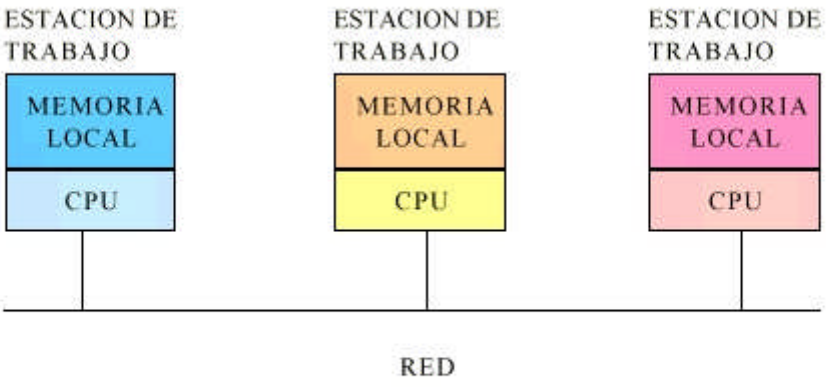
Cada cpu tiene una *conexión directa con su propia memoria local.*

Un *problema importante* es la forma en que las *cpu se comuniquen entre sí.*

El tráfico es solo entre una cpu y otra; el volumen de tráfico será varios órdenes de magnitud menor que si se utilizara la red de interconexión para el tráfico cpu - memoria.

Topológicamente es un esquema similar al del multiprocesador basado en un bus.

Consiste generalmente en una colección de estaciones de trabajo en una *LAN (red de área local)* (ver Figura 7.5 [25, Tanenbaum]).



Figura~7.5: Multicomputadora que consta de estaciones de trabajo en una LAN.

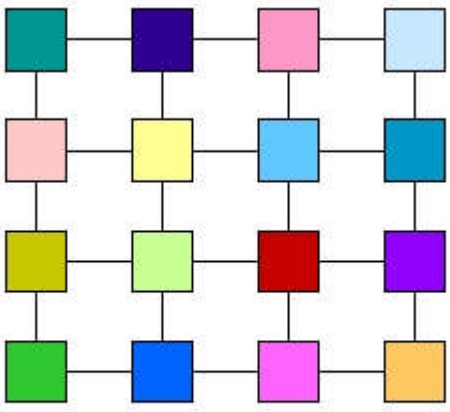
Multicomputadoras con Conmutador

Cada cpu tiene *acceso directo y exclusivo a su propia memoria particular [25, Tanenbaum].*

Existen *diversas topologías,* las más comunes son la **retícula** y el **hipercubo**.

Las *principales características de las retículas* son:

- Son fáciles de comprender.
- Se basan en las tarjetas de circuitos impresos.
- Se adecúan a problemas con una naturaleza bidimensional inherente (teoría de gráficas, visión artificial, etc.) (ver Figura 7.6 [25, Tanenbaum]).



Figura~7.6: Retícula.

Las *principales características del hipercubo* son:

- Es un cubo *"n"* - dimensional.
- En un *hipercubo de dimensión 4:*

    - Se puede considerar como dos cubos ordinarios, cada uno de ellos con 8 vértices y 12 aristas.
    - Cada vértice es un cubo.
    - Cada arista es una conexión entre 2 cpu.
    - Se conectan los vértices correspondientes de cada uno de los cubos.
- En un *hipercubo de dimensión 5:*
    - Se deberían añadir dos cubos conectados entre sí y conectar las aristas correspondientes en las dos mitades, y así sucesivamente.
- En un *hipercubo de "n" dimensiones:*
    - Cada cpu tiene *"n"* conexiones con otras cpu.
    - La complejidad del cableado aumenta en proporción logarítmica con el tamaño.
    - Solo se conectan los procesadores vecinos más cercanos:
        - Muchos mensajes deben realizar varios saltos antes de llegar a su destino.
        - La trayectoria más grande crece en forma logarítmica con el tamaño:
            - En la retícula crece como la raíz cuadrada del número de cpu.
    - Con la tecnología actual ya se pueden producir hipercubos de 16.384 cpu (ver Figura 7.7 [25, Tanenbaum]).

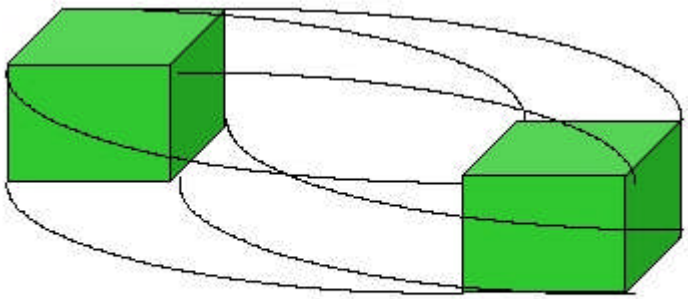


Figura 7.7: Hipercubo de dimensión 4.

Conceptos de Software

La importancia del software supera frecuentemente a la del hardware [25, Tanenbaum].

La imagen que un sistema presenta queda determinada en gran medida por el software del S. O. y no por el hardware.

Los *S. O.* no se pueden encasillar fácilmente, como el hardware, pero se los puede *clasificar en dos tipos:*

- *Débilmente acoplados.*
- *Fuertemente acoplados.*

El software *débilmente acoplado* de un sistema distribuido:

- Permite que las *máquinas y usuarios sean independientes entre sí* en lo fundamental.
- Facilita que interactúen en cierto grado cuando sea necesario.
- Los equipos individuales se distinguen fácilmente.

Combinando los distintos tipos de *hardware distribuido con software distribuido* se logran distintas soluciones:

- No todas interesan desde el punto de vista funcional del usuario:
  - Ej.: un multiprocesador es un multiprocesador:
    - No importa si utiliza un bus con cachés monitores o una red omega.

Sistemas Operativos de Redes

Una posibilidad es el *software débilmente acoplado en hardware débilmente acoplado [25, Tanenbaum]:*

- Es una solución muy utilizada.
- Ej.: una red de estaciones de trabajo conectadas mediante una LAN.

Cada usuario tiene una *estación de trabajo para su uso exclusivo:*

- Tiene su propio S. O.
- La mayoría de los requerimientos se resuelven localmente.
- Es posible que un usuario se conecte de manera remota con otra estación de trabajo:
  - Mediante un comando de *"login remoto".*
  - Se convierte la propia *estación de trabajo* del usuario en una *terminal remota* enlazada con la máquina remota.
  - Los comandos se envían a la máquina remota.
  - La salida de la máquina remota se exhibe en la pantalla local.
- Para alternar con otra máquina remota, primero hay que desconectarse de la primera:
  - En cualquier instante solo se puede utilizar una máquina.
- Las redes también disponen de un comando de copiado remoto de archivos de una máquina a otra:
  - Requiere que el usuario conozca:
    - La posición de todos los archivos.
    - El sitio donde se ejecutan todos los comandos.

Una *mejor solución* consiste en un *sistema de archivos global compartido*, accesible desde *todas* las estaciones de trabajo:

- Una o varias máquinas soportan al **sistema de archivos:**
  - Son los **"servidores de archivos".**

Los *"servidores de archivos":*

- Aceptan solicitudes de los programas de usuarios:
  - Los *programas* se ejecutan en las máquinas no servidoras, llamadas **"clientes".**
  - Las solicitudes se examinan, se ejecutan y la respuesta se envía de regreso.
- Generalmente tienen un *sistema jerárquico de archivos.*

Las estaciones de trabajo pueden *importar* o *montar* estos sistemas de archivos:

- Se incrementan sus *sistemas de archivos locales.*
- Se pueden montar los servidores en lugares diferentes de sus respectivos sistemas de archivos:
  - Las rutas de acceso a un determinado archivo pueden ser *diferentes* para las distintas estaciones.
  - Los distintos clientes tienen un *punto de vista distinto* del sistema de archivos.
  - El nombre de un archivo depende:
    - Del lugar desde el cual se tiene acceso a él.
    - De la configuración del sistema de archivos.

El S. O. de este tipo de ambiente debe:

- Controlar las estaciones de trabajo en lo individual.
- Controlar a los servidores de archivo.
- Encargarse de la comunicación entre los servidores.

Todas las máquinas pueden ejecutar el mismo S. O., pero esto no es necesario.

Si los clientes y los servidores ejecutan diversos S. O., como mínimo deben *coincidir en el formato y significado de todos los mensajes* que podrían intercambiar.

Esquemas como este se denominan **"sistema operativo de red":**

- Cada máquina tiene un alto grado de *autonomía.*
- Existen *pocos requisitos* a lo largo de todo el sistema.

NFS: Network File System

Es uno de los más conocidos y aceptado como **sistema operativo de red** [25, Tanenbaum].

Fue un desarrollo de Sun Microsystems, soportado también por distintos fabricantes:

- Surgió para UNIX pero se amplió a otros S. O. (ej.: MS - DOS).

- Soporta *sistemas heterogéneos,* por ej.: clientes de MS - DOS que hagan uso de servidores UNIX.
- Los equipos pueden ser también de *hardware heterogéneo.*

Los aspectos más interesantes son los relacionados con:

- *La arquitectura.*
- *El protocolo.*
- *La implantación.*

La Arquitectura de NFS

La idea fundamental es permitir que una colección arbitraria de clientes y servidores compartan un sistema de archivos común.

Generalmente todos los clientes y servidores están en la misma LAN, pero esto no es necesario; por ello se puede ejecutar NFS en una *WAN ("red de área amplia").*

NFS permite que *cada máquina* sea *un cliente y un servidor* al mismo tiempo.

Cada servidor de NFS *exporta uno o varios de sus directorios* (y subdirectorios dependientes) para el acceso por parte de clientes remotos.

Los clientes tienen acceso a los directorios exportados mediante el *montaje:*

- Cuando un cliente monta un directorio (remoto), este se convierte en parte de su jerarquía de directorios.

Un cliente sin disco puede montar un archivo remoto en su directorio raíz; esto produce un *sistema de archivos* soportado en su totalidad en un *servidor remoto.*

Las estaciones de trabajo que no poseen discos locales pueden montar directorios remotos en donde lo deseen, en la parte superior de su jerarquía de directorios local; esto produce un *sistema de archivos* que es *en parte local y en parte remoto.*

Si dos o más clientes *montan el mismo directorio* al mismo tiempo:

- Se pueden comunicar al compartir archivos en sus directorios comunes.
- No hay que hacer nada especial para lograr compartir los archivos.

Los *archivos compartidos* figuran en la jerarquía de directorios de varias máquinas y se los puede leer o escribir de la manera usual.

Protocolos de NFS

Uno de los *objetivos de NFS* es:

- Soportar un *sistema heterogéneo* en donde los clientes y servidores podrían ejecutar *distintos S. O. en hardware diverso*, por ello es esencial que la *interfaz* entre los clientes y los servidores esté bien definida.

NFS logra este objetivo definiendo dos **"protocolos cliente - servidor":**

- Un **"protocolo"** es un conjunto de:
  - *Solicitudes* que envían los clientes a los servidores.
  - *Respuestas* que envían los servidores de regreso a los clientes.

Un **"protocolo de NFS"** maneja el *montaje.*

Un *cliente* puede:

- Enviar el nombre de una ruta de acceso a un servidor.
- Solicitar el permiso para montar ese directorio en alguna parte de su jerarquía de directorios.

Si el nombre de la ruta de acceso es *válido* y el directorio especificado ha sido *exportado:*

- El servidor regresa un *"asa de archivo" (file handle)* al cliente:
  - Contiene campos que identifican:
    - De manera única el tipo de sistema de archivos, el disco, el número de nodo-i del directorio.
    - La información relativa a la seguridad.
  - Es utilizada en llamadas posteriores para la lectura o escritura de archivos en el directorio montado.

Algunos S. O. soportan la alternativa del *"automontaje":*

- Permite que un conjunto de directorios remotos quede *asociado* con un directorio local.
- Ninguno de los directorios remotos se monta durante el arranque del cliente.
- La primera vez que se abra un archivo remoto, el S. O. envía un mensaje a los servidores:
  - Los servidores responden y se monta su directorio.
- Las principales *ventajas sobre el montaje estático* son:
  - Se evita el trabajo de contactar servidores y montar directorios que *no son requeridos de inmediato.*
  - Si el cliente puede utilizar varios servidores en paralelo, se puede tener:
    - Cierta tolerancia a fallas.
    - Mejorar el rendimiento.

NFS no da soporte a la *duplicación* de archivos o directorios.

Otro **"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.
- Tener acceso a los atributos de archivo, tales como su modo, tamaño y fecha de la última modificación.

NFS soporta **"servidores sin estado":**

- *No mantienen la información de estado relativa a los archivos abiertos.*
- Si un servidor falla y arranca rápidamente, no se pierde información acerca de los archivos abiertos y los programas cliente no fallan.

El *"sistema de archivos remotos" (RFS)* del Sistema V de UNIX no funciona así, sino que:

- El servidor lleva un registro del hecho que cierto archivo está abierto y la posición actual del lector.
- Si un servidor falla y vuelve a arrancar rápidamente:
    - Se pierden todas las conexiones abiertas.
    - Los programas cliente fallan.

En un *"servidor sin estado"*, como NFS:

- Los bloqueos no tienen que asociarse con los archivos abiertos y el servidor no sabe cuáles archivos están abiertos.
- Se necesita un *mecanismo adicional independiente* para controlar el *bloqueo.*

NFS utiliza el *esquema de protección de UNIX*, con los bits *"rwx"* para el propietario, grupo y otros.

Se puede utilizar la criptografía de claves públicas para dar validez al cliente y el servidor en cada solicitud y respuesta; el cliente malicioso no puede personificar a otro cliente, ya que no conoce su clave secreta.

Las claves utilizadas para la autentificación, así como otra información, están contenidas en el **NIS:**

- *"Network Information Service":* Servicio de Información de la Red.
- Almacena parejas (clave, valor).
- Cuando se proporciona una clave, regresa el valor correspondiente.
- Almacena la asociación de:
    - Los nombres de los usuarios con las contraseñas (cifradas).
    - Los nombres de las máquinas con las direcciones en la red y otros elementos.

Implantación de NFS

La *implantación del código* del cliente y el servidor es *independiente de los protocolos NFS.*

Una implementación que suele tomarse como referencia es la de Sun, que consta de tres capas (ver Figura 7.8 [25, Tanenbaum]).

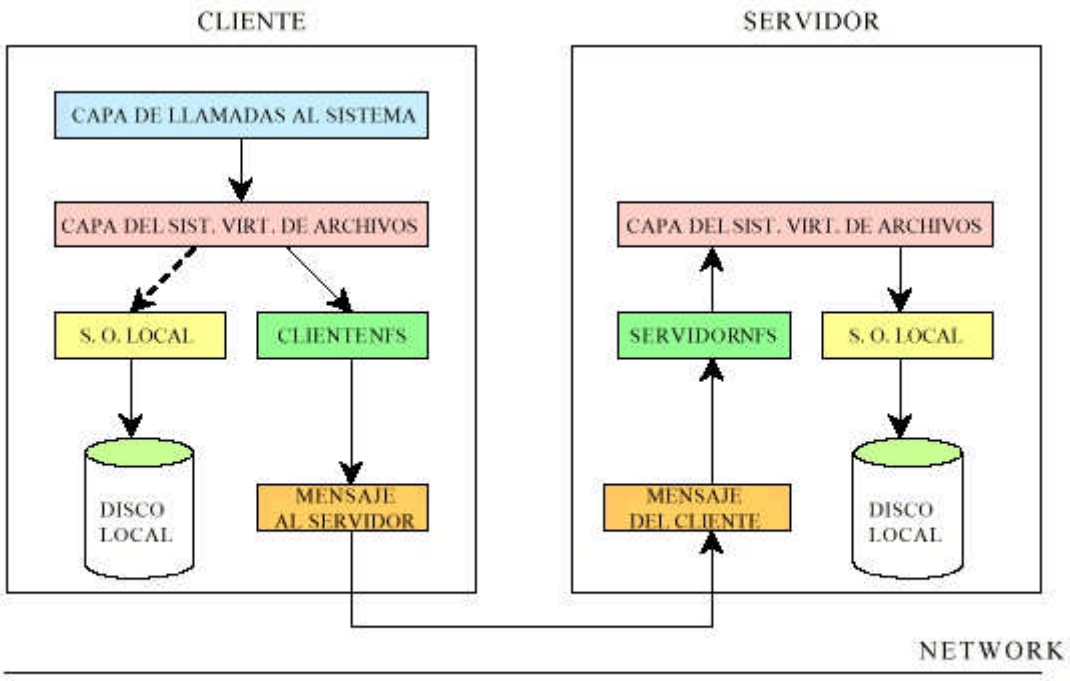


Figura 7.8: Estructura de capas de NFS.

La *capa superior* es la de *llamadas al sistema:*

- Maneja las llamadas del tipo *open, read y close.*
- Analiza la llamada y verifica los parámetros.
- Llama a la *segunda capa:* capa del *sistema virtual de archivos: virtual file system: VFS.*

La *capa VFS* mantiene una *tabla con una entrada por cada archivo abierto* que es análoga a la tabla de nodos-i para los archivos abiertos en UNIX.

La capa VFS tiene una entrada por cada archivo abierto:

- Se la llama *nodo-v (nodo-i virtual).*
- Los nodos-v se utilizan para indicar si el archivo es *local o remoto.*
- Para los archivos remotos, poseen la información suficiente como para tener acceso a ellos.

Para *montar un sistema remoto de archivos,* el administrador del sistema llama al programa *mount :*

- Utiliza la información del directorio remoto, el directorio local donde será montado y otros datos adicionales.
- Con el nombre del directorio remoto por montar se descubre el nombre de la máquina donde se localiza dicho directorio.

- Se verifica si el directorio existe y si está disponible para su montaje remoto.

El núcleo:

- Construye un *nodo-v* para el directorio remoto.
- Pide el código del cliente NFS para crear un *nodo-r* (nodo-i remoto) en sus tablas internas.

El nodo-v apunta al nodo-r.

Cada *nodo-v* de la capa VFS contendrá en última instancia un *apuntador a un nodo-i* en el S. O. local.

Es posible ver desde el *nodo-v* si un archivo o directorio es local o remoto y, si es remoto, encontrar su asa de archivo.

Todo archivo o directorio abierto tiene un *nodo-v* que apunta a un *nodo-r* o a un *nodo-i.*

Por razones de eficiencia las transferencias entre cliente y servidor se hacen en bloques grandes, generalmente de 8k:

- Luego de haber recibido la capa VFS del cliente el bloque necesario, emite la solicitud del siguiente bloque; esto se denomina *lectura adelantada (read ahead).*

Un criterio similar se sigue con la *escritura:*

- Antes de ser enviados al servidor los datos se acumulan en forma local:
  - Hasta completar cierta cantidad de bytes, o
  - Hasta que se cierra el archivo.

Otra técnica utilizada para *mejorar el rendimiento* es el *ocultamiento* o *caching:*

- Los *servidores* ocultan los datos para evitar el acceso al disco.
- Esto es invisible para los clientes.

Los *clientes* mantienen dos cachés:

- Uno para los atributos de archivo (nodos-i).
- Otro para los datos del archivo.

Cuando se necesita un nodo-i o un bloque del archivo:

- Primero se verifica si la solicitud se puede satisfacer mediante el caché del cliente, con esto se evita el tráfico en la red.

Un *problema importante del caching* es que *el caché no es coherente*; ej.:

- Dos clientes ocultan el mismo bloque del archivo.
- Uno de ellos lo modifica.

- Cuando el otro lee el bloque, obtiene el valor antiguo.
- Para mitigar este problema, la implantación de NFS:
    - Asocia a cada bloque caché un temporizador (timer).
    - Cuando el timer expira, la entrada se descarta.
    - Generalmente los tiempos son de:
        - 3 segundos para bloques de datos.
        - 30 segundos para bloques de directorio.
- Al abrir un archivo con caché se envía un mensaje al servidor para revisar la hora de la última modificación.
- Se determina si la copia del caché es válida o debe descartarse, utilizando una nueva copia del servidor.
- El temporizador del caché expira cada 30 segundos y todos los bloques modificados en el caché se envían al servidor.

Resumiendo:

- *NFS solo trata el sistema de archivos.*
- *NFS no hace referencia a otros aspectos, como la ejecución de un proceso.*
- *NFS se ha difundido ampliamente, a pesar de todo.*

Sistemas Realmente Distribuidos

NFS es un ejemplo de software débilmente acoplado en hardware débilmente acoplado [25, Tanenbaum]:

- Cada computadora puede ejecutar su propio S. O.
- Solo se dispone de un sistema compartido de archivos.
- El tráfico cliente - servidor debe obedecer los protocolos NFS.

Las multicomputadoras son un ejemplo de software fuertemente acoplado en hardware débilmente acoplado:

- Crean la *ilusión* de que toda la red de computadoras es *un solo sistema de tiempo compartido*, en vez de una colección de máquinas diversas.

Un sistema distribuido es aquel que se ejecuta en una colección de máquinas sin memoria compartida, pero que aparece ante sus usuarios como una sola computadora:

- A esta propiedad se la conoce como la **imagen de un único sistema.**

También se define un sistema distribuido como aquel que se ejecuta en una colección de máquinas enlazadas mediante una red pero que actúan como un ***uniprocesador virtual.***

Algunas de las *características* de los sistemas distribuidos son las siguientes:

- Debe existir un *mecanismo de comunicación global* entre los procesos:

    - Cualquier proceso debe poder comunicarse (intercambiar información) con cualquier otro.
- No tiene que haber:
    - Distintos mecanismos en distintas máquinas.
    - Distintos mecanismos para la comunicación local o la comunicación remota.
- Debe existir un *esquema global de protección.*
- La *administración de procesos debe ser la misma* en todas parte.
- Se debe tener una *misma interfaz de llamadas al sistema* en todas partes:
    - Es normal que se ejecuten núcleos idénticos en todas las cpu del sistema.
- Es necesario un *sistema global de archivos.*

Sistemas de Multiprocesador con Tiempo Compartido

Corresponde a software fuertemente acoplado en hardware fuertemente acoplado [25, Tanenbaum].

Los ejemplos más comunes de propósito general son los *multiprocesadores:*

- Operan *como un sistema de tiempo compartido*, pero con *varias cpu* en vez de una sola.
- Externamente un multiprocesador con 32 cpu de 3 mips actúa de manera muy parecida a una sola cpu de 96 mips; 1 mips: 1.000.000 de instrucciones por segundo.
- Se corresponde con la *imagen de un único sistema.*

La característica clave es la existencia de una sola cola para ejecución (Ver Figura 7.9 [25, Tanenbaum]):

- 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.*

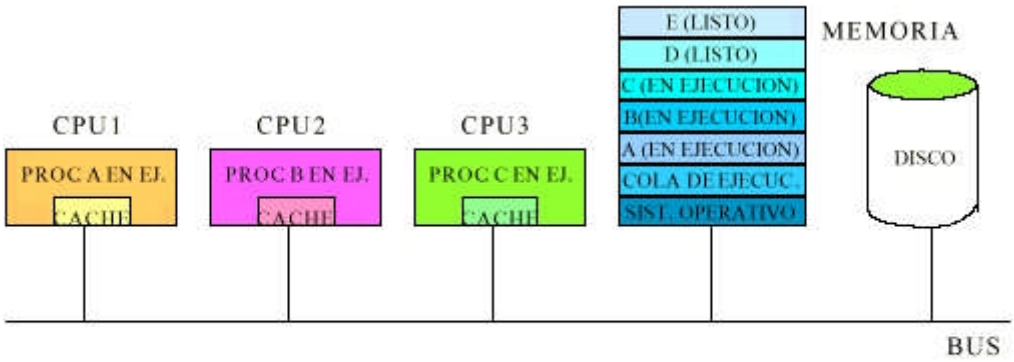


Figura 7.9: Un multiprocesador con una sola cola de ejecución.

Los programas de los procesos están en la memoria compartida, también el S. O.

El *planificador (de procesos)* del S. O. se ejecuta como una *"región crítica"*, con ello se evita que dos cpu elijan el mismo proceso para su ejecución inmediata.

Cuando un *proceso se asigna a un procesador:*

- Encuentra que el caché del procesador está ocupado por palabras de memoria que pertenecen a aquella parte de la memoria compartida que contiene al programa del proceso anterior.
- Luego de un breve lapso se habrán reemplazado por el código y los datos del programa del proceso asignado a ese procesador.

Ninguna cpu tiene memoria local, es decir que todos los programas se almacenan en la *memoria global compartida.*

Si todas las cpu están inactivas en espera de e / s y un proceso está listo para su ejecución:

- Es conveniente asignarlo a la cpu que se utilizó por última vez (para ese proceso):
    - La hipótesis es que ningún otro proceso utilizó esa cpu desde entonces *(hipótesis de Vaswani y Zahorjan).*

Si un proceso se bloquea en espera de e / s en un multiprocesador, el S. O. puede:

- Suspenderlo.
- Dejarlo en *"espera ocupada":*
    - Es aplicable cuando la mayoría de la e / s se realiza en *menos tiempo* del que tarda un cambio entre los procesos.
    - El proceso conserva su procesador por algunos milisegundos en espera de que la e / s finalice:
        - Si se agota el tiempo de espera y no ha finalizado la e / s, se realiza una conmutación de procesos.

Generalmente se dispondrá de un *sistema de archivos tradicional, con un único caché:*

- Globalmente considerado es similar al sistema de archivos de un único procesador.

Aspectos del Diseño

La comparación de las tres *principales formas* de organizar *"n"* cpu se puede resumir en la Tabla 7.2 [25, Tanenbaum].

Elemento	S. O. de red	S. O. distribuido	S. O. de multiprocesador
¿Se ve como un uniprocesador virtual?	No	Sí	Sí
¿Todas tienen que ejecutar el mismo S. O.?	No	Sí	Sí
¿Cuántas copias del S. O. existen?	n	n	1
¿Cómo se logra la comunicación?	Archivos compartidos	Mensajes	Memoria compartida
¿Se requiere un acuerdo en los protocolos de la red?	Sí	Sí	No
¿Existe una única cola de ejecución?	No	No	Sí
¿Existe una semántica bien definida para los archivos compartidos?	Por lo general No	Sí	Sí

Tabla 7.2: Comparación de tres formas distintas de organizar "n" cpu.

Los *aspectos claves en el diseño de S. O. distribuidos* son:

- *Transparencia.*
- *Flexibilidad.*
- *Confiabilidad.*
- *Desempeño.*
- *Escalabilidad.*

Transparencia

Un aspecto muy importante es la *forma* de lograr la *imagen de un único sistema* *[25, Tanenbaum]*.

Los usuarios *deben percibir* que la colección de máquinas conectadas son *un sistema de tiempo compartido de un solo procesador:*

- Un sistema que logre este objetivo se dice que es *transparente.*

Desde el *punto de vista de los usuarios*, la transparencia se logra cuando:

- Sus pedidos se satisfacen con ejecuciones en paralelo en distintas máquinas.
- Se utilizan una variedad de servidores de archivos.
- El usuario no necesita saberlo ni notarlo.

La transparencia desde el *punto de vista de los programas* significa diseñar la interfaz de llamadas al sistema de modo que no sea visible la existencia de varios procesadores.

No es transparente un sistema donde el *acceso a los archivos remotos* se realice mediante:

- El establecimiento explícito de una conexión en la red con un servidor remoto.
- El envío posterior de mensajes, donde el acceso a los servicios remotos será distinto al acceso a los servicios locales.

Existen distintos *tipos de transparencia* en un sistema distribuido:

- *De localización:* los usuarios no pueden indicar la localización de los recursos.
- *De migración:* los recursos se pueden mover a voluntad sin cambiar sus nombres.
- *De réplica:* los usuarios no pueden indicar el número de copias existentes.
- *De concurrencia:* varios usuarios pueden compartir recursos de manera automática.
- *De paralelismo:* las actividades pueden ocurrir en paralelo sin el conocimiento de los usuarios.

Flexibilidad

La flexibilidad es de *fundamental importancia [25, Tanenbaum]*.

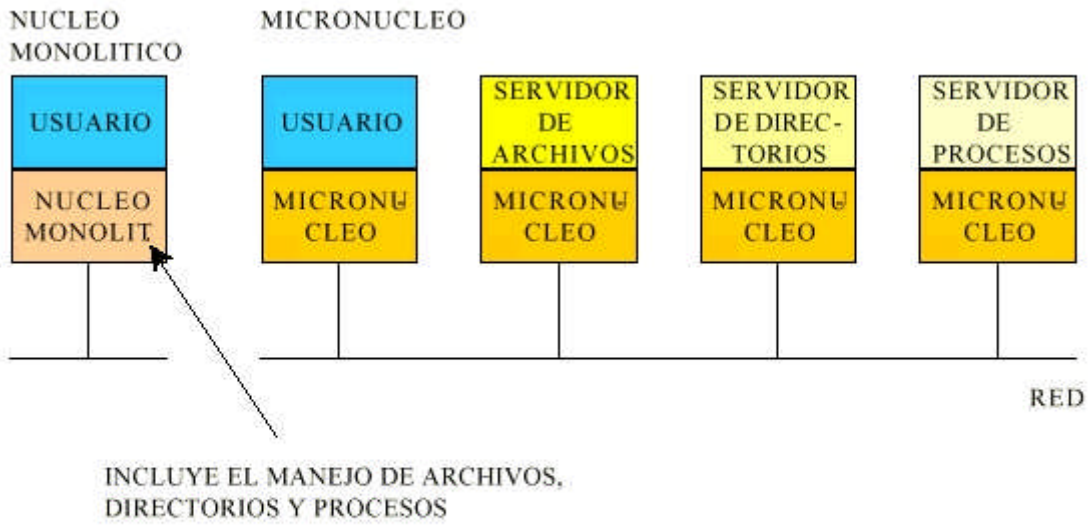


Figura 7.10: Esquema de núcleo monolítico y de micronúcleo.

Existen dos *escuelas de pensamiento* en cuanto a la *estructura de los sistemas distribuidos* (ver Figura 7.10 [25, Tanenbaum]):

- *Núcleo monolítico:*
  - Cada máquina debe ejecutar un núcleo tradicional que proporcione la mayoría de los servicios.
- *Micronúcleo (microkernel):*
  - El núcleo debe proporcionar lo menos posible.

    - El grueso de los servicios del S. O. se debe obtener a partir de los servidores al nivel usuario.

El *núcleo monolítico* es el S. O. centralizado aumentado con:

- Capacidades de red.
- Integración de servicios remotos.

Con *núcleo monolítico:*

- La mayoría de las llamadas al sistema se realizan mediante señalamiento al núcleo:
    - El núcleo realiza el trabajo.
    - El núcleo regresa el resultado al proceso del usuario.
- La mayoría de las máquinas tiene discos y administra sus propios sistemas locales de archivos.

El *micronúcleo es más flexible* y proporciona solo cuatro servicios mínimos:

- Un mecanismo de comunicación entre procesos.
- Cierta administración de la memoria.
- Una cantidad limitada de planificación y administración de procesos de bajo nivel.
- Entrada / salida de bajo nivel.

Contrariamente al núcleo monolítico, el *micronúcleo no proporciona* el sistema de archivos, el sistema de directorios, toda la administración de procesos o gran parte del manejo de las llamadas al sistema.

El objetivo es mantener el micronúcleo pequeño.

Todos los demás servicios del S. O. se implementan generalmente como servidores a nivel usuario:

- Para obtener un servicio:
    - El usuario envía un mensaje al servidor apropiado.
    - El servidor realiza el trabajo y regresa el resultado.

Una *importante ventaja* de este método es su *alta modularidad:*

- Existe una interfaz bien definida con cada servicio (conjunto de mensajes que comprende el servidor).
- Cada servicio es igual de accesible para todos los clientes, independientemente de la posición.
- Es fácil implantar, instalar y depurar nuevos servicios, sin necesidad de detener el sistema totalmente.

Confiabilidad

Un importante *objetivo de los sistemas distribuidos* es que *si una máquina falla, alguna otra debe encargarse del trabajo [25, Tanenbaum]*.

La *confiabilidad global teórica* del sistema podría ser el *"or"* booleano de la *confiabilidad de los componentes*; ejemplo:

- Se dispone de 5 servidores de archivos, cada uno con una probabilidad de 0,95 de funcionar en un instante dado.
- La probabilidad de falla simultánea de los 5 es $(0,05)^5 = 0,000006$.
- La probabilidad de que al menos uno esté disponible es 0,999994.

La *confiabilidad práctica* se ve disminuida ya que muchas veces se requiere que ciertos servidores estén en servicio simultáneamente para que el todo funcione, debido a ello algunos sistemas tienen una disponibilidad más relacionada con el *"and"* booleano de las componentes que con el *"or"* booleano.

Un aspecto de la confiabilidad es la *disponibilidad*, que se refiere a la *fracción de tiempo en que se puede utilizar el sistema.*

La disponibilidad se mejora mediante:

- Un diseño que no exija el funcionamiento simultáneo de un número sustancial de componentes críticos.
- La *redundancia*, es decir la duplicidad de componentes clave del hardware y del software.

Los datos no deben perderse o mezclarse y si los archivos se almacenan de manera redundante en varios servidores, *todas las copias deben ser consistentes.*

Otro aspecto de la confiabilidad general es la *seguridad*, lo que significa que los archivos y otros recursos deben ser protegidos contra el uso no autorizado.

Un aspecto también relacionado con la confiabilidad es la *tolerancia a fallas,* según la cual las fallas se deben ocultar brindando una *recuperación transparente para el usuario,* aunque haya cierta degradación de la performance.

Desempeño

Cuando se ejecuta una aplicación en un sistema distribuido *no debe parecer peor que su ejecución en un único procesador,* pero esto es difícil de lograr [25, Tanenbaum].

Algunas *métricas del desempeño* son:

- Tiempo de respuesta.
- Rendimiento (número de trabajos por hora).
- Uso del sistema y cantidad consumida de la capacidad de la red.

El problema se complica por el hecho de que la comunicación entre equipos es lenta comparada con:

- La velocidad de proceso.
- La velocidad de la comunicación dentro de un mismo procesador.

Se requiere el uso de protocolos de comunicaciones en los extremos (procesadores) que intervienen en la comunicación, con lo que se incrementa el consumo de ciclos de procesador.

Para *optimizar el desempeño* frecuentemente hay que:

- Minimizar el número de mensajes:
  - La dificultad es que la mejor forma de mejorar el desempeño es tener muchas actividades en ejecución paralela en distintos procesadores, pero esto requiere el envío de muchos mensajes.
- Centralizar el trabajo en una sola máquina:
  - Resulta poco apropiado para un sistema distribuido.

También se debe prestar atención al *tamaño de grano* de todos los cálculos:

- *Paralelismo de grano fino:*
  - Corresponde a trabajos con un gran número de pequeños cálculos y mucha interacción con otros trabajos, debido a ello requieren mucha comunicación que puede afectar el desempeño.
- *Paralelismo de grano grueso:*
  - Corresponde a trabajos con grandes cálculos, poca interacción y pocos datos, por lo tanto requieren poca comunicación y no afectan la performance.

Escalabilidad

La tendencia indica que el tamaño de los sistemas distribuidos es *hacia cientos de miles y aun decenas de millones de usuarios conectados [25, Tanenbaum].*

Existen *cuellos de botella potenciales* que se debe intentar evitar en los *sistemas distribuidos* de gran escala:

- *Componentes centralizados:*
  - Ej.: un solo servidor de correo para todos los usuarios.
- *Tablas centralizadas:*
  - Ej.: un único directorio telefónico en línea.
- *Algoritmos centralizados:*
  - Ej.: realización de un ruteo con base en la información completa.

Se deben *utilizar algoritmos descentralizados* con las siguientes características:

- Ninguna máquina tiene la información completa acerca del estado del sistema.
- Las máquinas toman decisiones solo en base a la información disponible de manera local.
- El fallo de una máquina no arruina el algoritmo.
- No existe una hipótesis implícita de la existencia de un reloj global.