PROGRAMA EDUCATIVO:
INGENIERÍA EN COMPUTACIÓN

UNIDAD DE APRENDIZAJE:
TIPOS DE SISTEMAS OPERATIVOS

ÍNDICE

Tipo de Sistemas Operativos, conceptos básicos y estructura de los mismos.

1.1 Tipos de sistemas operativos

¿Qué es un sistema operativo?

Un Sistema Operativo (SO) es el software básico de una computadora que provee una interfaz entre el resto de programas, los dispositivos hardware y el usuario. Las funciones básicas del Sistema Operativo son administrar los recursos de la máquina, coordinar el hardware y organizar archivos y directorios en dispositivos de almacenamiento, así como las acciones de entrada/salida. Los componentes básicos del sistema operativo son:

- Núcleo o kernel: que es la parte básica del S.O. de función medular y tiene la capacidad de traducir las órdenes que introduzca el usuario.
    - Interprete de comandos (Shell): es el programa que recibe todo lo que se escribe en la terminal y convierte en instrucciones para el S.O.
    - Promt: es un indicador que muestra el intérprete para anunciar que espera una orden del usuario en cierta localidad.

Sistemas operativos de mainframe: Una mainframe con 1000 discos y millones de gigabytes de datos no es poco común.

Los sistemas operativos para las mainframes están profundamente orientados hacia el procesamiento de muchos trabajos a la vez, de los cuales la mayor parte requiere muchas operaciones de E/S. Por lo general ofrecen tres tipos de servicios: procesamiento por lotes, procesamiento de transacciones y tiempo compartido.

Sistemas operativos de servidores.

Se ejecutan en servidores, que son computadoras personales muy grandes, estaciones de trabajo o incluso mainframes. Dan servicio a varios usuarios a la vez a través de una red y les permiten compartir los recursos de hardware y de software. Los servidores pueden proporcionar servicio de impresión, de archivos o Web.

Sistemas operativos de multiprocesadores

Es la conexión de varias CPU en un solo sistema. Dependiendo de la exactitud con la que se conecten y de lo que se comparta, estos sistemas se conocen como computadoras en paralelo, multicomputadoras o multiprocesadores. Nota: esto lo dice el libro Sistemas Operativos de Tanembaum mas sin embargo a lo largo del curso pudimos concluir que las multicomputadoras y los multiprocesadores son dos términos totalmente diferentes con características específicas.

Sistemas operativos de computadoras personales.

Su trabajo es proporcionar buen soporte para un solo usuario. Se utilizan ampliamente para el procesamiento de texto, las hojas de cálculo y el acceso a Internet.

Sistemas operativos integrados.

Los sistemas integrados (*embedded*), que también se conocen como incrustados o embebidos, operan en las computadoras que controlan dispositivos que no se consideran generalmente como computadoras, ya que no aceptan software instalado por el usuario. Algunos ejemplos comunes son los hornos de microondas, las televisiones, los autos, los grabadores de DVDs, los teléfonos celulares y los reproductores de MP3. La propiedad principal que diferencia a los sistemas integrados de los dispositivos de bolsillo es la certeza de que nunca se podrá ejecutar software que no sea confiable. No se pueden descargar nuevas aplicaciones en el horno de microondas; todo el software se encuentra en ROM. Esto significa que no hay necesidad de protección en las aplicaciones, lo cual conlleva a cierta simplificación. Los sistemas como QNX y VxWorks son populares en este dominio.

Sistemas operativos en tiempo real.

Otro tipo de sistema operativo es el sistema en tiempo real. Estos sistemas se caracterizan por tener el tiempo como un parámetro clave. Por ejemplo, en los sistemas de control de procesos industriales, las computadoras en tiempo real tienen que recolectar datos acerca del proceso de producción y utilizarlos para controlar las máquinas en la fábrica.

A menudo hay tiempos de entrega estrictos que se deben cumplir. Por ejemplo, si un auto se desplaza sobre una línea de ensamblaje, deben llevarse a cabo ciertas acciones en determinados instantes. Si un robot soldador realiza su trabajo de soldadura antes o después de tiempo, el auto se arruinará. Si la acción *debe* ocurrir sin excepción en cierto momento (o dentro de cierto rango), tenemos un **sistema en tiempo real duro**. Muchos de estos sistemas se encuentran en el control de procesos industriales, en aeronáutica, en la milicia y en áreas de aplicación similares. Estos sistemas deben proveer garantías absolutas de que cierta acción ocurrirá en un instante determinado.

Nota: En la información anterior se muestran los tipos de sistemas operativos en la actualidad más sin embargo existe una gran variedad de sistemas operativos según la implementación y el objetivo de su uso a grandes rasgos tenemos la siguiente clasificación:

- Por su estructura:
  - Monolíticos: Solo contienen un kernel.
  - Estructurados: Contienen una estructura lógica en el sistema que es específica para un fin.
- Por usuario:
  - Monousuario: Un Shell para un solo usuario.
  - Multiusuario: Un Shell para cada usuario.
- Por tareas:
  - Monotareas: Solo un Kernel, solo puede ejecutar tareas de una en una.
  - Multitareas: En este contexto existe el manejo de múltiples tareas o procesos al mismo tiempo.
    - ✓ OJO: las multitareas se realizan siempre y cuando se tengan los suficientes RECURSO para lograrlo (RAM, memoria, memoria virtual, procesador… entre otras).

1.2 Estructura de los sistemas operativos.

Se considera la organización interna de los S.O. y conforme a ella se le clasifica de la siguiente manera, destacándose sus principales características:
(Martínez)

Sistemas monolíticos.

En este diseño, que hasta ahora se considera como la organización más común, todo el sistema operativo se ejecuta como un solo programa en modo kernel. El sistema operativo se escribe como una colección de procedimientos, enlazados entre sí en un solo programa binario ejecutable extenso. Cuando se utiliza esta técnica, cada procedimiento en el sistema tiene la libertad de llamar a cualquier otro, si éste proporciona cierto cómputo útil que el primero necesita. Al tener miles de procedimientos que se pueden llamar entre sí sin restricción, con frecuencia se produce un sistema poco manejable y difícil de comprender.

Para construir el programa objeto actual del sistema operativo cuando se utiliza este diseño, primero se compilan todos los procedimientos individuales (o los archivos que contienen los procedimientos) y luego se vinculan en conjunto para formar un solo archivo ejecutable, usando el enlazador del sistema. En términos de ocultamiento de información, en esencia no hay nada: todos los procedimientos son visibles para cualquier otro procedimiento (en contraste a una estructura que contenga módulos o paquetes, en donde la mayor parte de la información se oculta dentro de módulos y sólo los puntos de entrada designados de manera oficial se pueden llamar desde el exterior del módulo).

Sistemas de capas.

Se organizar el sistema operativo como una jerarquía de capas, cada una construida encima de la que tiene abajo. El primer sistema construido de esta forma fue el sistema THE, construido en Technische Hogeschool Eindhoven en Holanda por E. W. Dijkstra (1968) y sus estudiantes. El sistema THE era un sistema simple de procesamiento por lotes para una computadora holandesa, la Electrologica X8, que tenía 32K de palabras de 27 bits (los bits eran costosos en aquel entonces).

Capa	Función
5	El operador
4	Programas de usuario
3	Administración de la entrada/salida
2	Comunicación operador-proceso
1	Administración de memoria y tambor
0	Asignación del procesador y multiprogramación

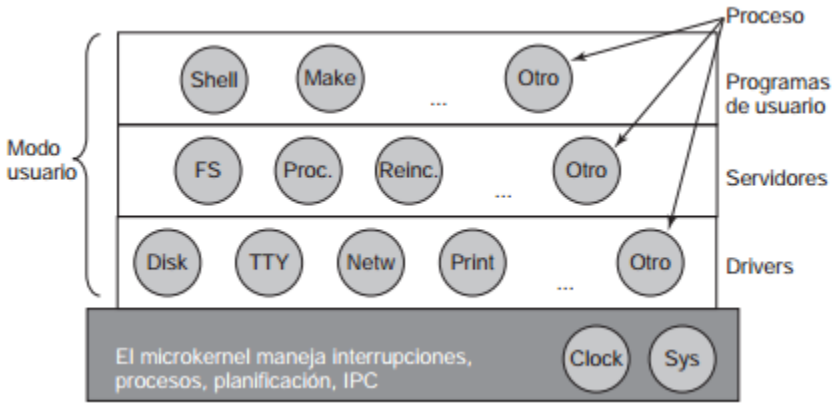
La capa 2 se encargaba de la comunicación entre cada proceso y la consola del operador (es decir, el usuario). Encima de esta capa, cada proceso tenía en efecto su propia consola de operador. La capa 3 se encargaba de administrar los dispositivos de E/S y de guardar en búferes los flujos de información dirigidos para y desde ellos. Encima de la capa 3, cada proceso podía trabajar con los dispositivos abstractos de E/S con excelentes propiedades, en vez de los dispositivos reales con muchas peculiaridades. La capa 4 era en donde se encontraban los programas de usuario.

No tenían que preocuparse por la administración de los procesos, la memoria, la consola o la E/S. El proceso operador del sistema se encontraba en el nivel 5. Una mayor generalización del concepto de capas estaba presente en el sistema MULTICS. En vez de capa, MULTICS se describió como una serie de anillos concéntricos, en donde los interiores tenían más privilegios que los exteriores (que en efecto viene siendo lo mismo). Cuando un procedimiento en un anillo exterior quería llamar a un procedimiento en un anillo interior, tenía que hacer el equivalente de una llamada al sistema; es decir, una instrucción TRAP cuyos parámetros se comprobara cuidadosamente que fueran válidos antes de permitir que continuara la llamada. Aunque todo el sistema operativo era parte del espacio de direcciones de cada proceso de usuario en MULTICS, el hardware hizo posible que se designaran procedimientos individuales (en realidad, segmentos de memoria) como protegidos contra lectura, escritura o ejecución.

Mientras que en realidad el esquema de capas de THE era sólo una ayuda de diseño, debido a que todas las partes del sistema estaban enlazadas entre sí en un solo programa ejecutable, en MULTICS el mecanismo de los anillos estaba muy presente en tiempo de ejecución y el hardware se encargaba de implementarlo. La ventaja del mecanismo de los anillos es que se puede extender fácilmente para estructurar los subsistemas de usuario.
(Edicion, 2009)

Microkernels.

La idea básica detrás del diseño de microkernel es lograr una alta confiabilidad al dividir el sistema operativo en módulos pequeños y bien definidos, sólo uno de los cuales (el microkernel) se ejecuta en modo kernel y el resto se ejecuta como procesos de usuario ordinarios, sin poder relativamente. En especial, al ejecutar cada driver de dispositivo y sistema de archivos como un proceso de usuario separado, un error en alguno de estos procesos puede hacer que falle ese componente, pero no puede hacer que falle todo el sistema. Así, un error en el driver del dispositivo de audio hará que el sonido sea confuso o se detenga, pero la computadora no fallará. En contraste, en un sistema monolítico con todos los drivers en el kernel, un driver de audio con errores puede hacer fácilmente referencia a una dirección de memoria inválida y llevar a todo el sistema a un alto rotundo en un instante.



Estructura del sistema MINIX3

Una idea que está en parte relacionada con tener un kernel mínimo es colocar el mecanismo para hacer algo en el kernel, pero no la directiva. Para aclarar mejor este punto, considere la planificación de los proceso. Un algoritmo de planificación relativamente simple sería asignar una prioridad a cada proceso y después hacer que el kernel ejecute el proceso de mayor prioridad que sea ejecutable. El mecanismo, en el kernel, es buscar el proceso de mayor prioridad y ejecutarlo. La directiva, asignar prioridades a los procesos, puede realizarse mediante los procesos en modo usuario. De esta forma, la directiva y el mecanismo se pueden desacoplar y el kernel puede reducir su tamaño.

Modelo cliente-servidor.

Una ligera variación de la idea del microkernel es diferenciar dos clases de procesos: los servidores, cada uno de los cuales proporciona cierto servicio, y los clientes, que utilizan estos servicios. Este modelo se conoce como cliente-servidor. A menudo la capa inferior es un microkernel, pero eso no es requerido. La esencia es la presencia de procesos cliente y procesos servidor. La comunicación entre clientes y servidores se lleva a cabo comúnmente mediante el paso de mensajes. Para obtener un servicio, un proceso cliente construye un mensaje indicando lo que desea y lo envía al servicio apropiado.

Después el servicio hace el trabajo y envía de vuelta la respuesta. Si el cliente y el servidor se ejecutan en el mismo equipo se pueden hacer ciertas optimizaciones, pero en concepto estamos hablando sobre el paso de mensajes. Como los clientes se comunican con los servidores mediante el envío de mensajes, no necesitan saber si los mensajes se manejan en forma local en sus propios equipos o si se envían a través de una red a servidores en un equipo remoto. En cuanto a lo que al cliente concierne, lo mismo ocurre en ambos casos: se envían las peticiones y se regresan las respuestas. Por ende, el modelo cliente-servidor es una abstracción que se puede utilizar para un solo equipo o para una red de equipos.

Cada vez hay más sistemas que involucran a los usuarios en sus computadoras domésticas como clientes y equipos más grandes que operan en algún otro lado como servidores. De hecho, la mayor parte de la Web opera de esta forma. Una computadora envía una petición de una página Web al servidor y la página Web se envía de vuelta. Éste es un uso común del modelo cliente-servidor en una red.

1.3 Sistema operativo de red.

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. Puede o no tener un disco duro. En definitiva, tiene su propio sistema operativo. Lo normal es que todos los comandos se ejecuten en forma local, justo en la estación de trabajo.

Aunque es mejor que nada, esta forma de comunicación es primitiva en extremo y ha provocado que los diseñadores de sistemas busquen formas más convenientes de comunicación y distribución de la información. Un método 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 al sistema de archivos. Los servidores de archivo aceptan solicitudes para la lectura y escritura de archivos por parte de los programas usuarios que se ejecutan en las otras máquinas (no servidoras), llamadas **clientes.** Cada una de las solicitudes que llegue se examina, se ejecuta y la respuesta se envía de regreso. 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.**

Asignación de procesadores.

Por definición, un sistema distribuido consta de varios procesadores. Éstos se pueden organizar como colección de estaciones de trabajo personales, una pila pública de procesadores o alguna forma híbrida. En todos los casos, se necesita cierto algoritmo para decidir cuál proceso hay que ejecutar y en qué máquina.

-No migratorios: Cuando un proceso es creado, se decide dónde colocarlo. Una vez colocado en una máquina, el proceso permanecerá ahí hasta que termine su ejecución.
-Migratorios: Un proceso se puede trasladar aunque haya iniciado su ejecución.

Este tipo de categorías permiten un mejor balanceo de carga.

Algoritmos de asignación de procesadores.

Algoritmos Deterministas vs. Heurísticos
Algoritmos Centralizados vs. Distribuidos
Algoritmos Óptimos vs. Subóptimos
Algoritmos Locales vs. Globales
Algoritmos Iniciados por el emisor vs. Iniciados por el receptor

- Deterministas: se conoce el comportamiento de los procesos.
- Heurísticos: cuando la carga es impredecible.
- Centralizados: tener toda la información en un solo lugar permite tomar una mejor decisión, pero es menos robusta y coloca una carga pesada en la máquina central.
- Distribuidos: carecen de alternativas.
- Óptimos: más caros, más información y mas proceso.

Aspectos de implantación.

Medición de la carga

Los algoritmos suponen que cada máquina conoce su carga.
Algunos algoritmos cuentan el número de procesos para determinar su carga.
Una mejora al criterio anterior es contar solo los procesos en ejecución o listos.
Otra forma de medir la carga es determinar la fracción de tiempo que el CPU está ocupado.

Costo excesivo

Muchos algoritmos ignoran el costo de recolectar medidas y desplazar los procesos.
Al desplazar un proceso se debe verificar si hay ganancia en hacerlo. Para esto hay que tomar en cuenta el uso de memoria, tiempo de CPU y ancho de banda de la red.

Complejidad

Se supone que al crear un nuevo proceso en una máquina, ésta se sobrecargará, por lo tanto hay que desplazar el proceso a otro lugar.

Envío de procesos.

- Se elige una máquina al azar y se envía el proceso, si la máquina esta subcargada entonces aceptará el proceso, en caso contrario se repite la operación hasta que alguien acepte el proceso o se exceda un contador de tiempo.
- Se elige una máquina al azar y se le pregunta si esta subcargada, si su respuesta es afirmativa entonces la máquina origen le envía el proceso, sino se repite la operación con otra máquina elegida al azar y así hasta encontrar una máquina adecuada o se exceda el número de pruebas, en cuyo caso permanecerá en el sitio de su creación.

1.4 Sistemas Operativos Distribuidos.

Los SO distribuidos desempeñan las mismas funciones que un SO normal, pero con una diferencia que es trabajar en un entorno distribuido, esto quiere decir que su misión principal es facilitar el acceso y la gestión de los recursos distribuidos en la red. En este tipo de sistemas los usuarios pueden acceder a recursos remotos de la misa manera en que lo hacen para los recursos locales, permiten distribuir trabajos, tareas o procesos entre un conjunto de procesadores (computadoras) desde un equipo o varios, lo cual es transparente para el usuario. Este tipo de sistemas tienen que ser muy confiables y estables y sobretodo tener la red en buen funcionamiento ya que sin ella no se podría trabajar en dicha distribución. (Sergio, 2015)

En si un SO Distribuido es un conjunto de procesadores interconectados por redes. Dos tipos de esquemas básicos de los SO Distribuidos:

- Fuertemente acoplado: es en donde se comparte una memoria de reloj global, cuyos tiempos de acceso son similares para todos los procesadores.

- Débilmente acoplado: los procesadores no comparten ni memoria ni reloj, ya que cada uno tiene su propia memoria local.(FalcomHive, 2015)

Características.

- **Transparencia**

En este sistema el acceso a los archivos remotos debe realizarse de la misa manera, donde el acceso a los archivos se realice mediante una conexión en la red con un servidor.

- Transparencia de localización: en donde los usuarios no pueden indicar la localización de los recursos de hardware y software como las impresoras, archivos y bases de datos.

- Transparencia de migración: esto quiere decir que los recursos deben moverse de una posición a otra sin tener que cambiar sus nombres.

- Transparencia de replica: ya que una réplica automática es imposible, e debe tener una consideración de n servidores conectados de manera lógica en anillo.

- Transparencia con respecto a la concurrencia: los usuarios no notaran la existencia de oros usuarios.

- Transparencia con respecto al paralelismo: un ejemplo es cuando se solicita la impresión de un documento, con frecuencia se prefiere que la salida aparezca como impresora local, o una que este a metros de distancia aunque esta impresora sea más rápida y tenga más funciones y este inactiva por el momento.

Flexibilidad.

Es importante que el sistema sea flexible ya que esta proporciona los servicios a los clientes.

Que cada nodo trabaje como debe de trabajar.

Núcleo monolítico: cada máquina debe ejecutar sus tareas como un núcleo tradicional que proporciona la mayoría de los servicios.

Micro núcleo: el núcleo realiza la mejor ejecución posible y el grueso de los servicios del SO se obtienen a partir de servidores a nivel de usuarios.

Confiabilidad.

Si una computadora falla en el sistema otra máquina se debe encargar de su trabajo.

Desempeño.

Con la utilización de las métricas de desempeño debe tener un buen tiempo de respuesta y rendimiento.

Escalabilidad.

Donde se puedan ir agregando nodos (computadoras) sin causar problemas los usuarios.(Tenenmaum, 1996)

3.1 Protocolos por capas.

Capas del modelo OSI.

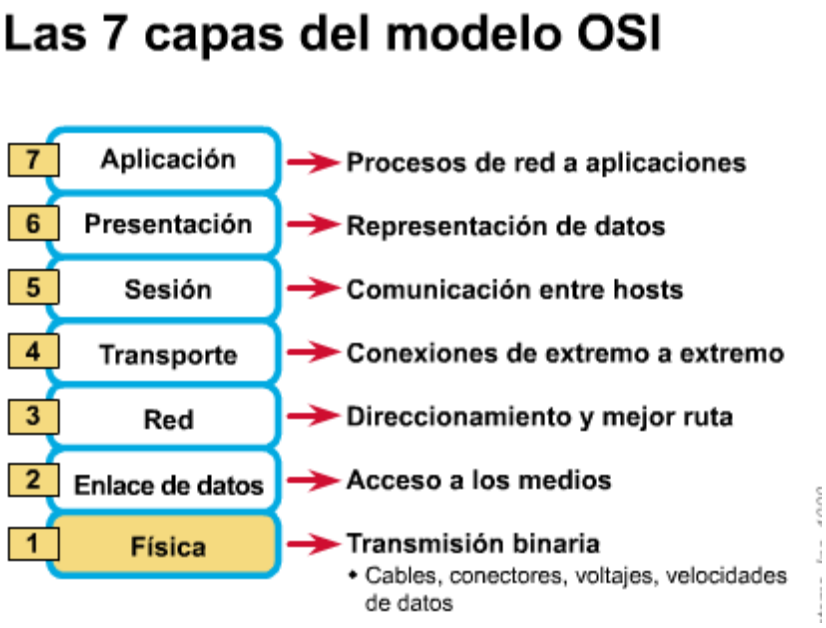


Imagen 1

Capa 1: La capa física.

La capa física define las especificaciones eléctricas, mecánicas, de procedimiento y funcionales para activar, mantener y desactivar el enlace físico entre sistemas finales. Las características tales como niveles de voltaje, temporización de cambios de voltaje, velocidad de datos físicos, distancias de transmisión máximas, conectores físicos y otros atributos similares son definidas por las especificaciones de la capa física. Si desea recordar la Capa 1 en la menor cantidad de palabras posible, piense en señales y medios. (CLAROS, I 2015)

En la clase de Sistemas Operativos Distribuidos adquirimos los siguientes conocimientos sobre la capa 1 del modelo OSI que se refiere a la capa (Física).

- Se transmite mediante 0 y 1 (Código Binario).
- Cableado: UTP categoría 5, velocidad de 100 Mbps. (Medio de transmisión), se utiliza para Ethernet 10BASE-T.
- Cable coaxial, Cable de par trenzado.
- Conectores
- Voltajes
- Velocidades de Datos.

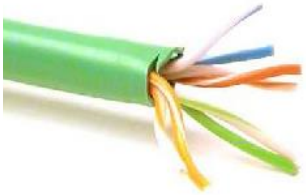


Figura 2

Capa 2: La capa de enlace de datos.

La capa de enlace de datos proporciona tránsito de datos confiable a través de un enlace físico. Al hacerlo, la capa de enlace de datos se ocupa del direccionamiento físico (comparado con el lógico), la topología de red, el acceso a la red, la notificación de errores, entrega ordenada de tramas y control de flujo. Si desea recordar la Capa 2 en la menor cantidad de palabras posible, piense en tramas y control de acceso al medio. (CLAROS, I 2015)

En la clase de Sistemas Operativos Distribuidos adquirimos los siguientes conocimientos sobre la capa 2 (Enlace).

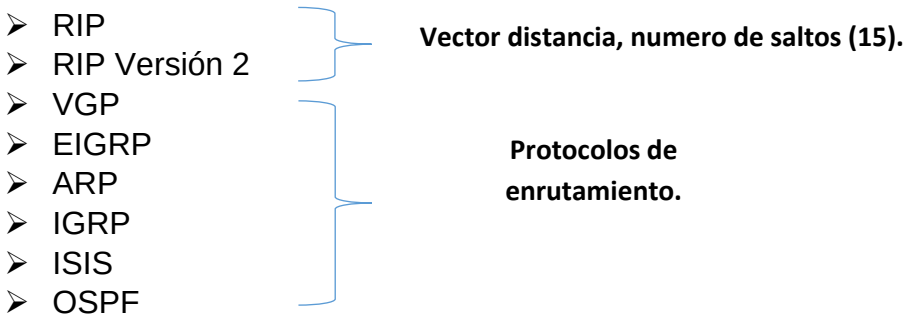
- Trama
- Acceso a los medios.
- HDLC
- Tarjetas de Red
- DIR.
- MAC
- Normas (802.3,802.5, 802.11,).
- Tabla de conmutación.
- Switches.
- STP
- VTP
- Token-Testigo.
- Velan Tronking Protocol→ VTP

- Statik Tronking Protocol→ STP → (Evita Bucles).

Capa 3: La capa de red.

La capa de red es una capa compleja que proporciona conectividad y selección de ruta entre dos sistemas de hosts que pueden estar ubicados en redes geográficamente distintas. Si desea recordar la Capa 3 en la menor cantidad de palabras posible, piense en selección de ruta, direccionamiento y enrutamiento (CLAROS, I 2015). En la clase de Sistemas Operativos Distribuidos adquirimos los siguientes conocimientos sobre la capa 3 (Red).

- Listas de acceso.
- Direccionamiento y mejor ruta.
- IP (Protocol de Internet).
- Enrutamiento
- Gateway
- Routers
- Tablas de enrutamiento.
- Segmentación
- Bridge (Router con)



Protocolos de enrutamiento: Se encargan de alcanzar la ruta más corta.

Capa 4: La capa de transporte.

Es la encargada de controlar el flujo de datos entre los nodos que establecen una comunicación. La capa de transporte segmenta los datos originados en el host emisor y los re ensamblan en una corriente de datos dentro del sistema del host receptor.

El límite entre la capa de transporte y la capa de sesión puede imaginarse como el límite entre los protocolos de aplicación y los protocolos de flujo de datos. Mientras que las capas de aplicación, presentación y sesión están relacionadas con asuntos de aplicaciones, las cuatro capas inferiores se encargan del transporte de datos.

También la capa de transporte intenta suministrar un servicio de transporte de datos que aísla las capas superiores de los detalles de implementación del transporte. Específicamente, temas como la confiabilidad del transporte entre dos hosts es responsabilidad de la capa de transporte. Al proporcionar un servicio de comunicaciones, la capa de transporte establece, mantiene y termina adecuadamente los circuitos virtuales. Al proporcionar un servicio confiable, se utilizan dispositivos de detección y recuperación de errores de transporte. Si desea recordar a la Capa 4 en la menor cantidad de palabras posible, piense en calidad de servicio y confiabilidad. (CLAROS, I 2015)

En la clase de Sistemas Operativos Distribuidos adquirimos los siguientes conocimientos sobre la capa 4 (Transporte).

- Segmentos
- Conexión de extremo a extremo.
- TCP, UDP (No orientado a conexión se solicita que los equipos se hablen).
    - Todos los protocolos de capa 4 hacen uso de TCP y UDP.
    - UDP: No pide retransmisión de cambio.
- HTTP: Puerto (80).
- FTP: Puerto (20, 21).
- TELNET: Puerto (22).
- SSH: Puerto (23).
- DNS: Puerto (53).
- Puertos

Capa 5: La capa de sesión.

Es la encargada de establecer, finalizar, administra el enlace de comunicación o sesión, entre las computadoras emisoras y receptoras (belarmino, 2015). La capa de sesión proporciona sus servicios a la capa de presentación. También sincroniza el diálogo entre las capas de presentación de los dos hosts y administra su intercambio de datos.

Además de regular la sesión, la capa de sesión ofrece disposiciones para una eficiente transferencia de datos, clase de servicio y un registro de excepciones acerca de los problemas de la capa de sesión, presentación y aplicación (CLAROS, I 2015). Los protocolos que operan en la capa de sesión pueden proporcionar dos tipos distintos de enfoques para que los datos vayan del emisor al receptor: La comunicación orientada a la conexión y la comunicación sin conexión.

En la clase de Sistemas Operativos Distribuidos adquirimos los siguientes conocimientos sobre la capa 5 (Sesión).

La comunicación se hace entre una estación de trabajo y un servidor, por medio del envio de una petición y accediendo a ella, utilizando también TCP/IP Para la comunicación, no importa si son diferentes Sistemas Operativos.

- Recuperación
- Comunicación entre hosts.
- Control de dialogo.
- Agrupamiento
- Half Duplex
- Full Duplex
- RPC (Remote Procedure Call)

Capa 6: La capa de presentación.

La capa de presentación garantiza que la información que envía la capa de aplicación de un sistema pueda ser leída por la capa de aplicación de otro. De ser necesario, la capa de presentación traduce entre varios formatos de datos utilizando un formato común. Si desea recordar la Capa 6 en la menor cantidad de palabras posible, piense en un formato de datos común. (CLAROS, I 2015)

En la clase de Sistemas Operativos Distribuidos adquirimos los siguientes conocimientos sobre la capa 6 (Presentación).

- Representación de Datos.
- Esta capa toma los paquetes de la capa de aplicación y los convierte a un formato genérico que pueden leer todas las computadoras.
- Verificar el tipo de archivo que sea el mismo que se envió para saber si llego el mensaje a su destino.
- Transmisión de datos por la Red.
- Se encarga de cifrar los datos, también los comprime para reducir su tamaño.

Capa 7: La capa de aplicación.

Es la capa del modelo OSI más cercana al usuario; suministra servicios de red a las aplicaciones del usuario. Difiere de las demás capas debido a que no proporciona servicios a ninguna otra capa OSI, sino solamente a aplicaciones que se encuentran fuera del modelo OSI. Algunos ejemplos de aplicaciones son los programas de hojas de cálculo, de procesamiento de texto y los de las terminales bancarias. La capa de aplicación establece la disponibilidad de los potenciales socios de comunicación, sincroniza y establece acuerdos sobre los procedimientos de recuperación de errores y control de la integridad de los datos. (CLAROS, I 2015)

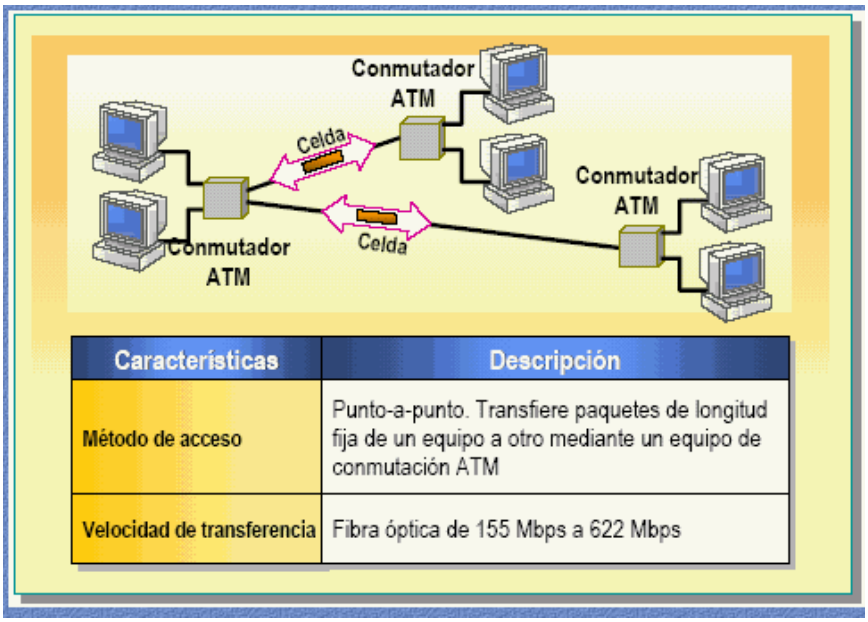
En la clase de Sistemas Operativos Distribuidos adquirimos los siguientes conocimientos sobre la capa 7 (Aplicación).

- Ofrece servicios de red.
- FTP
- HTTP
- HTTPS
- SMTP
- TKTP
- SNMP
- TELNET
- SSH
- Redes conmutadas por circuitos.
- Corta distancia.
- Puerto 25
- ATM
- Frema relay

3.2 Redes con modo de transferencia asíncrona.

- El modo de transferencia asíncrona (Asynchronous transfer mode, ATM) es una red de conmutación de paquetes que envía paquetes (CELDAS ATM) de longitud fija a través de LANs o WANs, en lugar de paquetes de longitud variable utilizados en otras tecnologías.
- Los paquetes de longitud fija, o celdas, son paquetes de datos que contienen únicamente información básica de la ruta, permitiendo a los dispositivos de conmutación enrutar el paquete rápidamente.

La comunicación tiene lugar sobre un sistema punto-a-punto que proporciona una ruta de datos virtual y permanente entre cada estación. (Modo de Transferencia Asíncrona ATM, 2015)

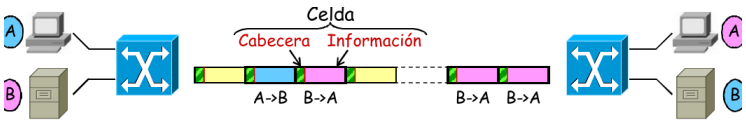


Características	Descripción
Método de acceso	Punto-a-punto. Transfiere paquetes de longitud fija de un equipo a otro mediante un equipo de conmutación ATM
Velocidad de transferencia	Fibra óptica de 155 Mbps a 622 Mbps

- ATM es una tecnología de telecomunicación desarrollada para hacer frente a la gran demanda de capacidad de transmisión para servicios y aplicaciones.

- El modo de transferencia asíncrona (Asynchronous transfer mode, ATM) es una red de conmutación de paquetes que envía paquetes (celdas ATM) de longitud fija a través de LAN's o WAN's, en lugar de paquetes de longitud variable utilizados en otras tecnologías.

- El modo de transferencia asíncrono (ATM), es una tecnología de conmutación que usa pequeñas celdas de tamaño fijo por medio de una red LAN o WAN. Los paquetes o celdas, son paquetes de datos que contienen únicamente información básica de la ruta, permitiendo a los dispositivos de conmutación enrutar el paquete rápidamente. La comunicación tiene lugar sobre un sistema punto-a-punto que proporciona una ruta de datos virtual y permanente entre cada estación. (Laura, 2015)



- Celdas con cabecera y carga útil (información)
- Cada conexión se identifica por la cabecera (cto. virtual)
- El acceso al canal es asincrónico o estadístico (en función del volumen de información)

Problema:

- Control de tráfico: garantizar capacidad y retardos para

Es cierto que las redes de datos, como X.25, existieron durante años, pero eran hijastros y con frecuencia tenían velocidad de 56 o 64 Kb por segundo. Sistemas como Internet fueron considerados como curiosidades académicas, similares a una vaca con dos cabezas en una exhibición de un circo. En la transmisión de la voz analógica era donde estaba la acción (y el dinero).

Cuando las compañías telefónicas decidieron construir redes para el siglo xxI, se enfrentaron a un dilema: el tráfico de las voces es suave y necesita un ancho de banda bajo, pero constante, mientras que el tráfico de datos es explosivo, no necesita por lo general, un ancho de banda (cuando no hay tráfico), pero a veces necesita gran cantidad de recursos durante periodos muy breves. Ni la conmutación de circuitos tradicional (utilizada en la red telefónica de conmutación pública) ni la conmutación de paquetes (utilizada en Internet) eran adecuadas para ambos tipos de tráfico.

El modelo ATM consiste en que un emisor establece primero una conexión (es decir, un circuito virtual) con el (o los) receptor(es). Durante el establecimiento de la conexión, se determina una ruta desde el emisor hasta el (los) receptor(es) y se guarda la información del ruteo en los computadores a lo largo del camino. Mediante esta conexión se pueden enviar los paquetes, pero el hardware los separa en pequeñas unidades de tamaño fijo llamadas celdas. Las celdas de un circuito virtual dado siguen todas las mismas rutas guardadas en los conmutadores. Cuando ya no se necesita la conexión, ésta se libera y se purga la información de ruteo de los conmutadores.

3.3 El modelo cliente-servidor.

Servidores ofrecen → servicios → usuarios (clientes).

Están presentes un conjunto de procesos servidores, cada uno actuando como un gestor de recursos para una colección de recursos de un tipo, y una colección de procesos clientes, cada uno llevando a cabo una tarea que requiere acceso a algunos recursos hardware y software compartidos. Los gestores de recursos a su vez podrán necesitar acceder a recursos compartidos manejados por otros procesos, así que algunos procesos son ambos clientes y servidores.

En el modelo, cliente-servidor, todos los recursos compartidos son mantenidos y manejados por los procesos servidores. Los procesos clientes realizan peticiones a los servidores cuando necesitan acceder a algún recurso. Si la petición es válida, entonces el servidor lleva a cabo la acción requerida y envía una respuesta al proceso cliente. (Rojo, 2015). El modelo OSI es una solución elegante y realmente aplicable en muchos casos pero tiene problemas:

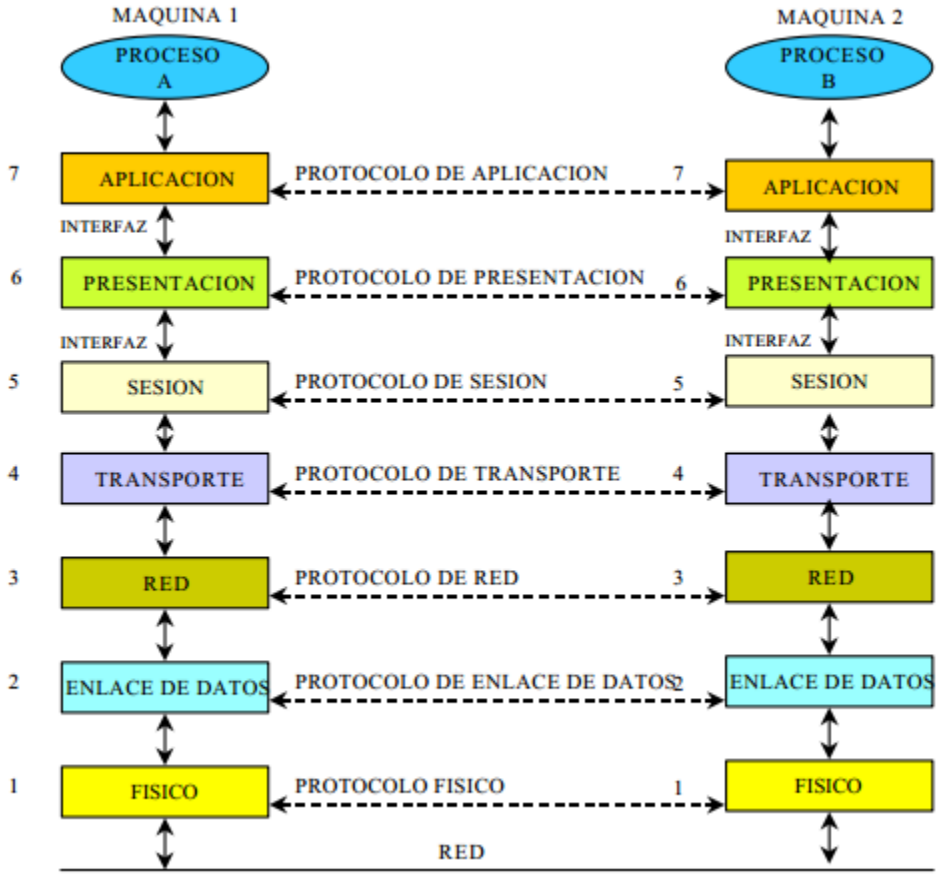
- La existencia de los encabezados genera un costo adicional en la trasmisión
- Cada envió de mensaje genera :
    - proceso en media docena de capas
    - Preparación y agregado de encabezados en el camino hacia “abajo”.
    - Eliminación y examen de encabezados en el camino hacia “arriba”.

Con enlaces del orden de decenas de miles de bits /segundo y CPU poderosas

- La carga de procesamiento de los protocolos no es significativa
- El factor limitante es la capacidad de las lineas

Con problemas del orden de millones de bits / segundo y computadoras personales:

- La carga de procesamiento de los protocolos si es frecuentemente significativa
- El factor limitante no es la capacidad de líneas.



Muestra Capas, Interfaces y protocolos de modelo OSI

(Martínez, 2015)

Direccionamiento.

→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, el núcleo sabrá qué hacer con el mensaje recibido (dárselo al único proceso en ejecución)

Si existen varios procesos en ejecución 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

→ Otro tipo de sistema de direccionamiento envía mensajes a los procesos en vez de a las máquinas. Aunque este método elimina toda ambigüedad acerca de quién es el verdadero receptor, presenta el problema de cómo identificar los procesos

→Otro método consiste en asignarle a cada proceso una dirección que no contenga un número de máquina. La forma de lograr esto es mediante un asignado centralizado de direcciones a los procesos que mantenga tan sólo un contador. Al recibir una solicitud de dirección, el asignador regresa el valor actual del contador y lo incrementa en uno. La desventaja de este esquema es que los componentes centralizados no se pueden extender a los grandes sistemas, por lo cual hay que evitarlo.

En resumen, se tienen los métodos siguientes para el direccionamiento de los procesos:

1. Integrar machine.nurnber al código del cliente.

2. Dejar que los procesos elijan direcciones al azar; se localizan mediante transmisiones.

3. Colocar los nombres en ASCII de los servidores en los clientes; buscarlos al tiempo de la ejecución.

Primitivas con bloqueo vs. Sin bloqueo.

Primitivas con bloqueo (a veces llamadas primitivas síncronas). Es un proceso llama a send. Eso especifica un destino v un buffer dónde enviar ese destino. Mientras se envía, el mensaje, el proceso emisor se bloquea (es decir, se suspende). La instrucción que sigue a la llamada a send no se ejecuta sino hasta que el mensaje

Las **primitivas sin bloqueo** (a veces llamadas primitivas asíncronas). 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 emisor puede continuar su cómputo en forma paralela con la transmisión del mensaje

Primitivas almacenadas en buffer vs. No almacenadas.

Las **primitivas descritas** hasta ahora son en esencia **primitivas no almacenadas.** Esto significa que una dirección se refiere a un proceso específico, Una llamada a la recepción de la dirección indica al núcleo de la máquina donde se ejecuta ésta que el proceso que llamó escucha a la dirección addr y que está preparada para recibir un mensaje enviado a esa dirección.

La técnica **primitiva con almacenamiento en buffers** trata de un proceso interesado en recibir mensajes le indica al núcleo que cree un buzón para él y especifica una dirección en la cual busca los paquetes de la red, todos los mensajes que lleguen con esa dirección se colocan en el buzón. La llamada a receive elimina ahora un mensaje del buzón o se bloquea.

Primitivas confiables vs. No confiables.

Cuando un 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 haya sido entregado. El mensaje podría haberse perdido.

Existen tres distintos enfoques de este problema:

- Uno consiste en volver a definir la semántica de send para hacerla no confiable. El sistema no da garantía alguna acerca de la entrega de los mensajes. La implantación de una comunicación confiable se deja por completo en manos de los usuarios
- 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 al proceso usuario (cliente). El reconocimiento va de un núcleo al otro; ni el cliente ni el servidor ven alguna vez un reconocimiento.
- El tercer método aprovecha el hecho de que la comunicación cliente-servidor se estructura como 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.

Implantación del modelo cliente-servidor.

Los detalles de implantación de la transferencia de mensajes dependen en cierta medida de las opciones elegidas, es posible hacer algunos comentarios generales acerca de la implantación, protocolos y software.

Cuatro diseños de la implementación del servidor

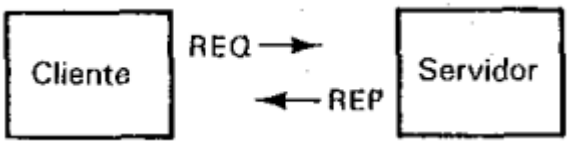
Elemento	Opción 1	Opción 2	Opción 3
Direcciona-miento	Número de máquina	Direcciones ralas de procesos	Búsqueda de nombres en ASCII por medio del servidor
Bloqueo	Primitivas con bloqueo	Sin bloqueo, con copia al núcleo	Sin bloqueo, con interrupciones
Almacena-miento en buffers	No usar el almacenamiento en buffers, descartar los mensajes inesperados	Sin almacenamiento en buffers, mantenimiento temporal de los mensajes inesperados	Buzones
Confiabilidad	No confiable	Solicitud-Reconocimiento-Respuesta-Reconocimiento	Solicitud-Respuesta-Reconocimiento

La elección requiere en gran medida de la tasa de pérdidas de la red que se utilice. Otro aspecto interesante es el protocolo subyacente utilizado en la comunicación cliente-servidor. La siguiente muestra seis tipos de paquetes que se utilizan comúnmente para la implantación de los protocolos cliente-servidor.

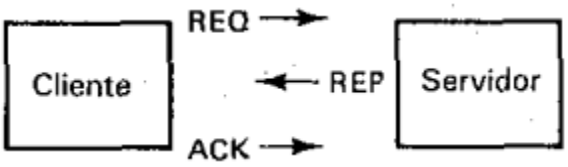
Código	Tipo de paquete	De	A	Descripción
REQ	Solicitud	Cliente	Servidor	El cliente desea servicio
REP	Respuesta	Servidor	Cliente	Respuesta del servidor al cliente
ACK	Reconocimiento	Cual-quiera	Algún otro	El paquete anterior que ha llegado
AYA	¿Estás vivo?	Cliente	Servidor	Verifica si el servidor se ha descompuesto
IAA	Estoy vivo	Servidor	Cliente	El servidor no se ha descompuesto
TA	Intenta de nuevo	Servidor	Cliente	El servidor no tiene espacio
AU	Dirección desconocida	Servidor	Cliente	Ningún proceso utiliza esta dirección

(Tanenbaum, 2009)

El paquete REQ, utilizado para enviar un mensaje de solicitud de un cliente a un servidor. (Para hacer más sencilla esta exposición, el resto de esta sección supondremos que cada mensaje cabe dentro de un paquete). El paquete REP, que regresa los resultados del servidor al cliente. El paquete ACK, es cual se utiliza en protocolos confiables para confirmar la recepción correcta de un paquete anterior.



Ejemplos de intercambios de protocolos.



3.4 Llamadas a procedimientos remotos.

(RPC- Remote Procedure Call).

- Birrel y Nelson abordaron este tema en 1984.
- El programador no se preocupa por la transferencia de mensajes.
- Mediante RPC se puede invocar la ejecución de un procedimiento en la máquina A que no está en A, sino en B.
- Cuando un proceso es invocado, la dirección que tiene el IP es almacenada en la pila, y el registro apuntador será cargado con la dirección del proceso invocado, cuando el procedimiento invocado termina, entonces el registro IP es cargado con la dirección que está en el tope de la pila y sigue la ejecución del programa.
- El objetivo de un RPC es que se parezca lo más posible a una llamada local (transparencia).
- Cuando un servidor se ejecuta por primera vez, lo primero que hace es exportar su interfaz: nombre de los servicios y sus parámetros, especificando si son de entrada o de salida, su versión y su identificador único.
- Un proceso llamado conector es quien recibe la interfaz y realiza el registro.

- La versión de un servicio es importante, pues indica si un servidor ha sido actualizado o no. Así el conector contendrá las últimas versiones de los servidores.

- Este método de exportación e importación es altamente flexible.

- El conector se puede convertir en un cuello de botella, además un proceso puede tener un tiempo de vida corto y el tiempo entre importar y exportar interfaces puede ser significativo.

- **Consideremos una llamada como:**

  count = read(fd, buf, nbytes);

- En donde fd es un entero, bufes un arreglo de caracteres y nbytes es otro entero. Si la llamada se hace desde el programa principal, la pila se verá como en la figura 2-17(a) antes de la llamada. Para hacer ésta, quien la hace coloca los parámetros en la pila, en orden, el último en el primer lugar, como se muestra en la figura 2-17(b). (La razón de que los compiladores de C introduzcan los datos en orden inverso tiene que ver con prinlf; al hacerlo así,printf siempre localiza su primer parámetro, la cadena de formato.) Después de que read termina su ejecución, coloca el valor de regreso en un registro, elimina la dirección y transfiere de nuevo el coiitrol a quien hizo la llamada. Este último elimina entonces los parámetros de la pila y regresa a su estado original, como se ve en la figura2- 17(c).

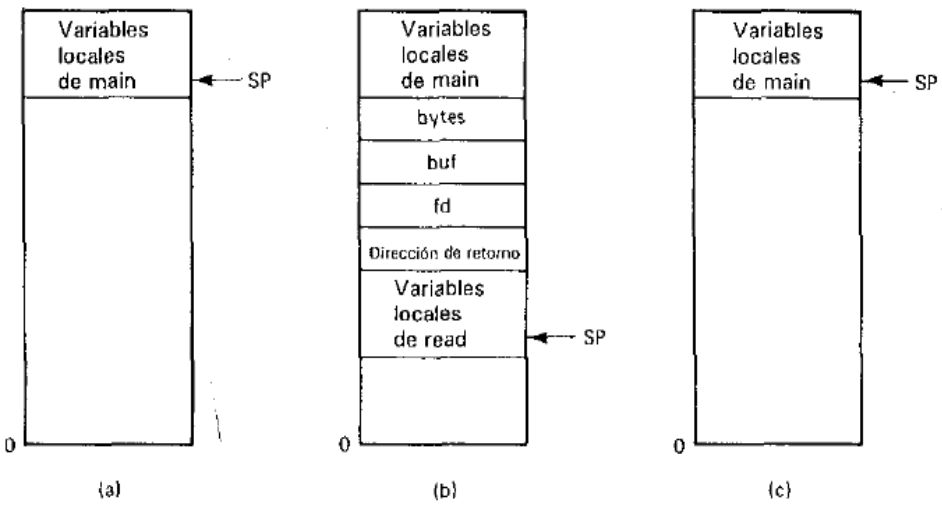


Figura 2-17. (a) La pila antes de la llamada a *read*. (b) La pila mientras el procedimiento llamado está activo. (c) La pila después del regreso a quien hizo la llamada.

El cliente no puede localizar al servidor.

- Debido a que el servidor cambió de dirección y el cliente no se actualizó.
- El error provoca una excepción.
- El uso de mecanismos de recuperación, destruye la transparencia.

Pérdida de mensajes de respuesta.

- Se puede utilizar un cronómetro, si no llega respuesta en un periodo razonable, se envía la solicitud nuevamente. ¿Pero qué se perdió, la solicitud, la respuesta o el servidor es lento?
- Una posible solución es asociar un número secuencial a cada solicitud. Entonces el núcleo del servidor debe mantener un registro del número secuencial más reciente, así podrá notar la diferencia entre una solicitud original y las retransmisiones.
- Como protección adicional se podría colocar un bit en el encabezado del mensaje para distinguir las solicitudes originales de las retransmisiones.

Falla del servidor.

- Puede suceder que:

  -El servidor falla después de responder

  -El servidor falla antes de responder

- Existen tres opciones en torno a lo que se debe hacer en estos casos:
- Semántica al menos una vez. Seguir intentando hasta conseguir una respuesta (el servidor puede volver a arrancar o se reconecta un nuevo servidor). Garantiza que la RPC se realiza al menos una vez.
- Semántica a lo más una vez. Se da por vencido inmediatamente e informa de la falla. La RPC se realiza a lo más una vez o ni una sola vez.
- No garantizar nada.

Fallas del cliente.

- Cuando un cliente falla después de enviar una solicitud sin que haya recibido respuesta, existirá una labor de cómputo de la cual nadie espera respuesta.
- A esta labor de cómputo no deseada se le llama huérfano.
- Problemas que puede provocar un huérfano.
  - Desperdiciar ciclos de CPU
  - Bloquear archivos o capturar recursos valiosos
  - Crear confusión (si el cliente vuelve a arrancar y realiza de nuevo la RPC, la respuesta puede ser inmediata y entonces no sabrá lo que paso)

3.5 Comunicación en grupo.

Una hipótesis subyacente e intrítiseca de RPC es que la comunicación sólo es entre dos partes, el cliente y el servidor. A veces, existen ciertas circunstancias en las que la comunicación es entre varios procesos y no solamente dos. Por ejemplo, consideremos un grupo de servidores de archivos que cooperan para ofrecer un servicio de archivos tolerante de fallas. En tal sistema, sería reconocible que un cliente envíe el mensaje a todos los servidores, para garantizar que la solicitud se lleve a cabo, aún en el caso en que uno de ellos falle. RPC no puede controlar la comunicación de un servidor con muchos receptores, a internos que realice RPC con cada uno en forma individual. En esta sección analizaremos las alternativas de comunicación en las que un mensaje se puede enviar a varios receptores en una operación.

Los grupos son dinámicos:

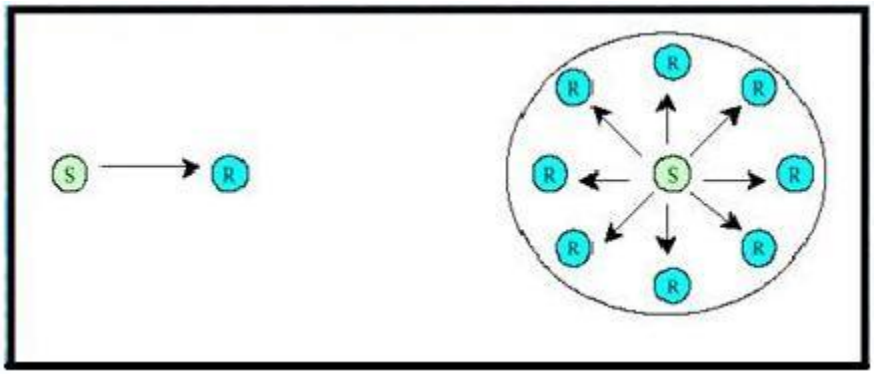
- Se pueden crear y destruir.
- Un proceso se puede unir a un grupo o dejar a otro
- Un proceso puede ser miembro de varios grupos a la vez.

La implantación de la comunicación en grupo depende en gran medida del hardware. En ciertas redes es posible crear una dirección especial de red a la que pueden escuchar varias máquinas. Cuando se envía un mensaje a una de esas direcciones se lo entrega automáticamente a todas las máquinas que escuchan a esa dirección.

- Esta técnica se denomina multitransmisión.
- Cada grupo debe tener una dirección de multitransmisión distinta.

Las redes que no soportan multitransmisión operan con transmisión simple:
Significa que los paquetes que tienen cierta dirección se entregan a todas las máquinas. Se puede utilizar para implantar los grupos, pero es menos eficiente que la multitransmisión. Cada máquina debe verificar, mediante su software, si el paquete va dirigido a ella, En caso negativo se descarta, pero para analizarlo se generó una Interrupción y se dedicó ciclos de CPU. Otra solución es implantar la comunicación en grupo mediante la transmisión por parte del emisor de paquetes individuales a cada uno de los miembros del grupo; en vez de un paquete se precisan “n” paquetes. Es menos eficiente que las soluciones anteriores. Es una solución válida particularmente con grupos pequeños.El envío de un mensaje de un emisor a un único receptor se llama unitransmisión. (NANCY, 2015)

Un grupo es una colección de procesos que actúan juntos en cierto sistema o alguna forma determinada por el usuario. La propiedad fundamental de todos los grupos es que cuando un mensaje se envía al propio grupo, todos los miembros de éste lo reciben. Es una forma de comunicación uno-muchos (emisor y muchos receptores) y contrasta con la comunicación puntual (Tanenbaum)



Los grupos son algo parecido a las organizaciones sociales. La finalidad de presentar los grupos es permitir a los procesos como una abstracción. Así, un proceso puede enviar un mensaje a un grupo de servidores sin tener que conocer su número o su localización, que puede cambiar de una llamada a la siguiente. La comunicación de grupo depende en gran medida del hardware .En ciertas redes, es posible crear una dirección especial de red (por ejemplo, indicada al hacer que uno de los bits de orden superior tome el valor 1) a la que pueden escuchar varias máquinas; cuando se envía un mensaje a una de estas direcciones, se entrega de manera automática a todas las máquinas que escuchan a esa dirección.

Esta técnica se llama multitransmisión y las redes que no tienen multitransmisión siguen teniendo transmisión simple. Para un grupo con *n* miembros, se necesitan *n* paquetes, en vez de un paquete en el caso de la multitransmisión o la transmisión simple. El envío de un mensaje de un emisor a un receptor se llama unitransmisión.

Broadcast o Difusion forzada.

Transmisión de un paquete que será recibido por todos los dispositivos en una red.

Multicast.

Consiste en la entrega de paquetes a través de una red a varios destinos de forma simultánea evitando al máximo el duplicar los paquetes, esto es, se duplican paquetes exclusivamente cuando se bifurca el camino a los diferentes destinos finales.

Unicast o Pointcast.

Un nodo emite y otro recibe, solo escucha aquel a quien se dirigió el mensaje Una clasificación adicional es la realizada en base a grupos.

4.1 Hilos.

En la mayoría de los sistemas distribuidos cada proceso tiene un espacio de direcciones y un hilo de control. Pero existen muchas situaciones en donde se desea tener varios hilos de control que compartan un espacio de direcciones. Los hilos se inventaron para permitir la combinación del paralelismo en ejecución y el bloqueo de las llamadas al sistema.

Un ejemplo es un servidor que lee solicitudes y esta solicitud elige a un hilo trabajador inactivo o bien disponible y le envía la solicitud por medio a un apuntador/ id a cada hilo, el servidor activa al trabajador este verifica si puede satisfacer la solicitud por medio del bloque al que tienen acceso todos los hilos.

- Los hilos se ejecutan de forma secuencial compartiendo espacio en memoria.
- Comparten el mismo CPU (a menos que haya varios CPU, en ese caso se ejecutan en paralelo)
- Cada hilo tiene su contador del programa, su pila para llevar el registro de su posición y se ejecutan en forma estrictamente secuencial.

Las formas en que se pueden concluir los hilos son de dos maneras:

- Al finalizar su trabajo
- Al ser eliminados desde el exterior

El espacio de núcleo sabe de los hilos y sabe cómo manejarlos.

Paquetes de hilos

Se permite que cada proceso tenga su algoritmo de planificación adaptado. Los hilos tienen una mejor forma de manipulación ya que los hilos del núcleo requieren espacio para sus tablas y su pila, lo cual es un problema si existe un número muy grande de hilos.

Desventajas

- Cuando un hilo empieza su ejecución, ningún otro hilo del mismo proceso puede ejecutarse a menos que el hilo en ejecución entregue el CPU de manera voluntaria.
- Si un hilo provoca una falla de página, el núcleo puede ejecutar otro mientras que espera que la página requerida sea traída del disco.

Los hilos a nivel de usuario permiten que cada proceso tenga u algoritmo de planificación adaptado. Los hilos se ejecutan en la parte superior de un sistema al tiempo de ejecución, el cual es una colección de procesos que manejan. (Tenembaut,1996)

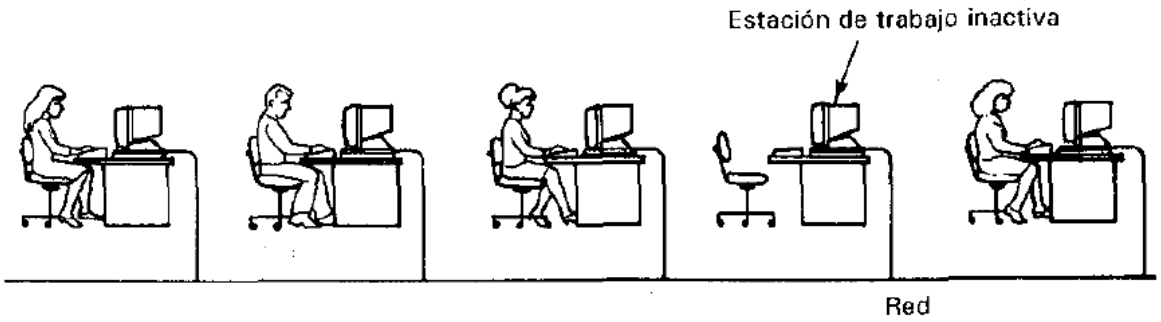
4.2 Modelo de sistemas.

En un sistema tradicional, sólo existe un procesador, por lo que no viene a cuento la pregunta de cómo debe utilizarse éste. En un sistema distribuido, con varios procesadores, éste es un aspecto fundamental del diseño. Los procesadores de un sistema distribuido se pueden organizar de varias formas. Estos modelos se basan en filosofías diferentes en lo fundamental de lo que debe ser un sistema distribuido.

El modelo de estación de trabajo.

El modelo de estación de trabajo es directo: el sistema consta de estaciones de trabajo (computadoras personales para usuarios finales) dispersas en un edificio o campus y conectadas entre sí por medio de una LAN de alta velocidad.

Algunas de las estaciones de trabajo pueden estar en oficinas, con lo que de manera implícita, cada una de ellas se dedica a un usuario, mientras que otras pueden estar en áreas públicas y tener distintos usuarios en el transcurso del día. En ambos casos, en un instante dado, una estación de trabajo puede tener un usuario conectado a ella y tener entonces un"poseedor" (aunque sea temporal) o estar inactiva.



Los modelos sin disco y con disco se resumen en la siguiente imagen:

Dependencia de los servidores de archivos

Uso del disco	Ventajas	Desventajas
(Sin disco)	Bajo costo, fácil mantenimiento del hardware y el software, simetría y flexibilidad	Gran uso de la red; los servidores de archivos se pueden convertir en cuellos de botella
Paginación, archivos de tipo borrador	Reduce la carga de la red comparado con el caso sin discos	Un costo alto debido al gran número de discos necesarios
Paginación, archivos de tipo borrador, binarios	Reduce todavía más la carga sobre la red	Alto costo; complejidad adicional para actualizar los binarios
Paginación, archivos de tipo borrador, binarios, ocultamiento de archivos	Una carga aún menor en la red; también reduce la carga en los servidores de archivos	Alto costo; problemas de consistencia del caché
Sistema local de archivos completo	Escasa carga en la red; elimina la necesidad de los servidores de archivos	Pérdida de transparencia

De arriba hacia abajo, se comienza desde la total dependencia de los servidores de archivos hasta la total independencia. Las ventajas del modelo de estación de trabajo son variadas y claras. Ciertamente, el modelo es fácil de comprender. Los usuarios tienen una cantidad fija de poder de cómputo exclusivo, con lo que tienen un tiempo de respuesta garantizado. Los programas gráficos complejos pueden ser muy rápidos, puesto que tienen acceso directo a la pantalla. Cada usuario tiene alto grado de autonomía y puede asignar los recursos de su estación de trabajo como lo juzgue necesario. Los discos locales favorecen esta independencia y hacen posible que el trabajo continúe en mayor o menor grado si el servidor de archivos falla.

Uso de estaciones de trabajo inactivas.

¿Cómo encontrar una estación de trabajo inactiva? Para comenzar, ¿qué es una estación de trabajo inactiva? A primera vista, parecería que una estación de trabajo sin una persona utilizando la consola es una estación de trabajo inactiva, pero con los modernos sistemas de cómputo las cosas no son tan sencillas. En muchos sistemas, una estación donde ninguna persona está frente a ella puede ejecutar docenas de procesos, como demonios de reloj, de correo, de noticias y todos los demás demonios posibles.

Por otro lado, un usuario que entre al sistema por la mañana pero que después no toque la computadora durante horas no coloca una carga adicional en dicho sistema. Los distintos sistemas toman diversas decisiones acerca del significado de la palabra "inactiva", pero por lo general se dice que la estación de trabajo está inactiva cuando nadie toca el teclado o el ratón durante varios minutos y no se ejecuta algún proceso iniciado por el usuario. En consecuencia, pueden existir diferencias sustanciales en la carga de una estación de trabajo inactiva y otra, debido, por ejemplo, al volumen de correo recibido en la primera pero no en la segunda.

Los algoritmos que se utilizan para localizar las estaciones de trabajo inactivas se pueden dividir en dos categorías: controlados por el servidor y controlados por el cliente. En la primera categoría, cuando una estación de trabajo está inactiva y por lo tanto se convierte en un servidor potencial, anuncia su disponibilidad. Puede hacer esto al proporcionar su nombre, dirección en la red y propiedades en un archivo de registros (o base de datos).

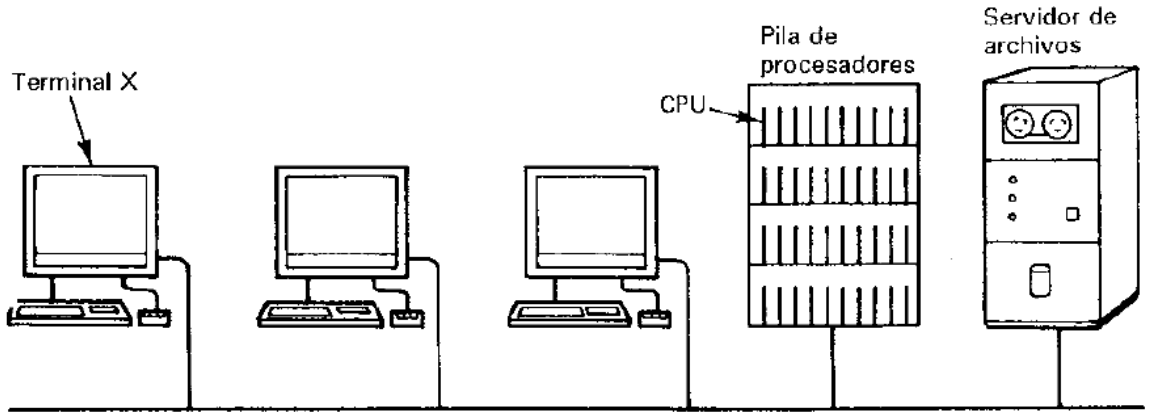
Posteriormente, cuando un usuario desee ejecutar un comando en una estación de trabajo inactiva, puede escribir algo como:

```
remote command
```

Y el programa ***remofe*** busca en el registro una estación de trabajo adecuada. Por razones de confiabilidad, también es posible contar con varias copias del registro. Otra alternativa consiste en que la estación inactiva anuncie el hecho de que no tiene trabajo, al colocar un mensaje que se transmite en toda la red. Todas las demás estaciones registran esto. De hecho, cada máquina tiene su copia del registro. La ventaja de esto es un menor costo en la búsqueda de una estación de trabajo y mayor redundancia. La desventaja es pedir a todas las máquinas que se encarguen de mantener el registro.

El modelo de la pila de procesadores.

Aunque el uso de las estaciones de trabajo inactivas añade cierto poder de cómputo al sistema, no enfrenta un aspecto todavía más fundamental: ¿Qué ocurre cuando es posible proporcionar 10 o 100 veces más CPU que el número de usuarios activos? Ya hemos visto una solución, la cual consiste dar a cada quien un multiprocesador personal. Sin embargo, éste es un diseño algo ineficiente. Otro método consiste en construir una **pila de procesadores,** repleta de CPU, en el cuarto de las máquinas, los cuales se pueden asignar de manera dinámica a los usuarios según la demanda.



El principal argumento para la centralización del poder de cómputo como pila de procesadores proviene de la teoría de colas. Un sistema de colas es una situación donde los usuarios generan en forma aleatoria solicitudes de trabajo a un servidor. Cuando el servidor está ocupado, los usuarios se forman para el servicio y se procesan según su turno. Algunos de los ejemplos comunes de sistemas de colas son las panaderías, los contadores para ingreso a los aeropuertos, contadores de salida en los supermercados y muchos otros más.

Un modelo híbrido.

Se pide establecer una mediación al proporcionar a cada usuario una estación de trabajo personal y además tener una pila de procesadores. Aunque esta solución es más cara que cualquiera de los dos modelos puros, combina las ventajas de ambos. El trabajo interactivo se puede llevar a cabo en las estaciones de trabajo, con una respuesta garantizada. Sin embargo, las estaciones inactivas no se utilizan, lo cual hace más sencillo el diseño del sistema. Sólo se dejan sin utilizar. En vez de esto, todos los procesos no interactivos se ejecutan en la pila de procesadores, así como todo el cómputo pesado en general.

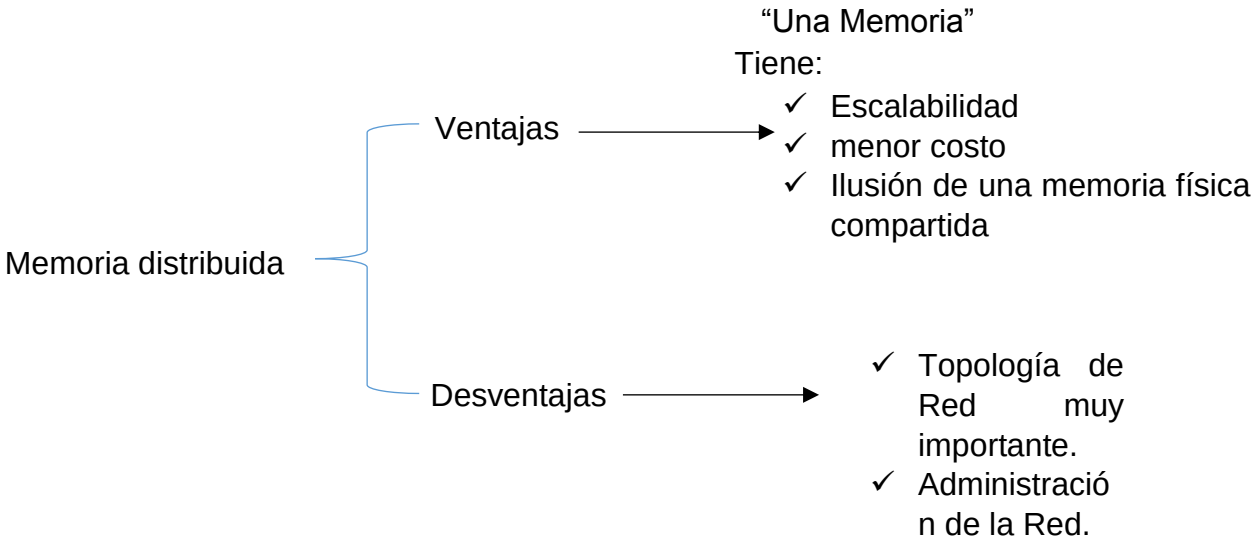
Este modelo proporciona una respuesta interactiva más rápida, un uso eficiente de los recursos y un diseño sencillo.(Tanenbaum A. A.)

5.1 Memoria Distribuida.

En lo ya antes visto en clase de Sistemas Operativos Distribuidos y acompañado de los temas anteriores, se abarcara y describirá el subtema de:

Memoria Distribuida.

Es la memoria que se puede compartir con más de un procesador. Es decir cualquiera puede acceder a ella ya sea leer o escribir. (TANENBAUM, 2015)



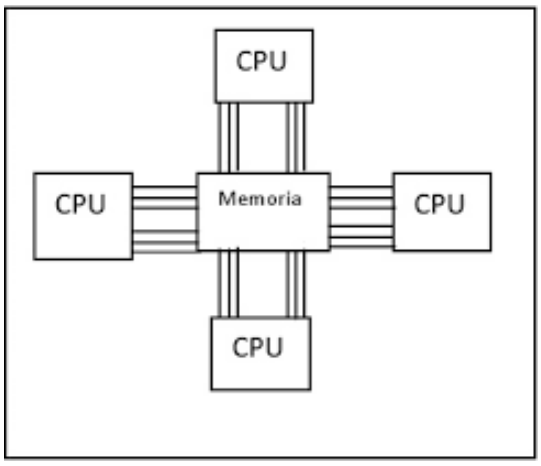


Figura 1

Entorno de la Memoria Distribuido.

- Hay lectura y escritura por medio de **bus** y por **memoria.**
- El microprocesador puede accesar a la memoria.
- El Arbitraje: Se encarga de avisar a la memoria si está escrita o no, arbitraje también es conocido como verificar.
- Un Proceso solo escribe datos en memoria.
- El HD está mapeado en la memoria virtual.
- LRU (Least Recent Use).
- Memoria Cache

Copia de archivos para cada uno de los microprocesadores, esto sirve para cuando el siguiente microprocesador quiera leer en memoria y este ya cambio, pueda accesar a él en su misma copia o en alguna de sus compañeros. La memoria consiste en no compartir todo el espacio, sino una porción para cada uno de los microprocesadores. El bus está grabado por lo general en la tarjeta principal (La tarjeta madre).

5.2 Modelos de consistencia.

Un modelo de consistencia es en esencia un contrato entre el software y la memoria. Dice que si el software acuerda obedecer ciertas reglas, la memoria promete trabajar de forma correcta. Si el software viola estas reglas, todo acaba y ya no se garantiza que la operación de memoria sea la correcta.

Consistencia estricta.

El modelo de consistencia más estricto es el de consistencia estricta. Se define mediante la siguiente condición:

Cualquier lectura a una localidad de memoria x regresa el valor guardado por la operación de escritura más reciente en x.

Esto cuando la memoria tiene consistencia estricta, todas las escrituras son visibles al instante a todos los procesos y se mantiene un orden de tiempo global absoluto. Si se cambia una localidad de memoria, todas las lecturas posteriores desde esa localidad ven el nuevo valor, sin importar qué tan pronto se haga la lectura después del cambio y sin importar los procesos que estén haciendo la lectura ni la posición de éstos. De manera análoga, si se realiza una lectura, se obtiene el valor actual, sin importar lo rápido que se realice la siguiente escritura.

Consistencia secuencia.

La consistencia secuencial es un modelo de memoria un poco más débil que la consistencia estricta. Una memoria con consistencia secuencial es la que satisface la siguiente condición:

El resultado de cualquier ejecución es el mismo que si las operaciones de todos los procesos fueran ejecutadas en algún orden secuencial, y las operaciones de cada proceso individual aparece en esta secuencia en el orden especificado por su programa.

Significa esta definición es que cuando los procesos se ejecutan en paralelo en diferentes máquinas (o aun en seudo paralelo en un sistema de tiempo compartido), cualquier intercalado válido es un comportamiento aceptable, pero todos los procesos deben ver la misma serie de llamadas a memoria.

Una memoria donde un proceso (o procesos) ven un intercalado y otro proceso ve otro distinto no es una memoria con consistencia secuencial.

Una memoria con consistencia secuencial se puede implantar en un sistema DSM o multiprocesador que duplique las páginas que se pueden escribir, garantizando que ninguna operación de memoria comienza hasta que las anteriores hayan concluido.

Consistencia causal.

El modelo de consistencia representa un debilitamiento de la consistencia secuencial que hace una distinción entre los eventos potencialmente relacionados por causalidad y aquellos que no lo están.

Las operaciones que no están relacionadas de forma causal son concurrentes. Para que una memoria sea consistente de forma causal, obedece las siguientes condiciones:

Las escrituras potencialmente relacionadas de forma causal son vistas por todos los procesos en el mismo orden. Las escrituras concurrentes pueden ser vistas en un orden diferente en máquinas diferente.

La implantación de la consistencia causal mantiene un registro de cuáles procesos han visto y cuáles escrituras. Esto significa de hecho que debe construirse y mantenerse una gráfica de dependencia con las operaciones que dependen de otras.

Consistencia PRAM y consistencia del procesador.

En la consistencia causal se permite que las escrituras concurrentes sean vistas en diferente orden en varias máquinas, aunque las relacionadas de forma causal deben verse en el mismo orden por todas las máquinas.

La consistencia PRAM (Pipelined RAM), sujeta a la condición:

Las escrituras realizadas por un proceso son recibidas por los otros procesos en el orden en que son realizadas, pero las escrituras de procesos diferentes pueden verse en un orden diferente por procesos diferentes.

PRAM son las siglas del entubamiento del RAM ya que las escrituras realizadas por un proceso pueden entubarse; es decir, el proceso no tiene que quedarse esperando que termine cada una antes de comenzar la siguiente.

La consistencia del procesador, es tan cercano a la consistencia PRAM intención de establecer una condición adicional sobre una memoria con consistencia de procesador, a saber, la coherencia de la memoria.

Consistencia débil.

Aunque la consistencia PRAM y la del procesador proporcionan mejor desempeño que los modelos más fuertes, siguen siendo innecesariamente restrictivos para muchas aplicaciones.

En otras palabras toda la memoria (compartida) está sincronizada.

Se define este modelo, llamado consistencia débil, diciendo que tiene tres propiedades:

- Los accesos a las variables de sincronización son secuencialmente consistentes.
- No se permite realizar un acceso a una variable de sincronización hasta que las escrituras anteriores hayan terminado en todas partes.
- No se permite realizar un acceso a los datos (lectura o escritura) hasta realizar todos los accesos anteriores a las variables de sincronización.

El punto 1 dice que todos los procesos ven los accesos a las variables de sincronización en el mismo orden.

El punto 2 dice que el acceso a una variable de sincronización "dirige el flujo". Obliga a terminar en todas partes las escrituras que están en progreso, terminadas o de forma parcial en algunas memorias pero no en otras.

El punto 3 dice que cuando se tiene acceso a las variables ordinarias (es decir, que no son de sincronización), ya sea para lectura o escritura, se han realizado todas las sincronizaciones anteriores. Al realizar una sincronización antes de leer los datos compartidos, un proceso puede estar seguro de obtener los valores más recientes.

Consistencia de liberación.

La consistencia de liberación proporciona estos dos tipos. Los accesos de adquisición indican a la memoria del sistema que está a punto de entrar a una región crítica. Los accesos de liberación dice que acaba de salir de una región crítica. Estos accesos se implantan como operaciones ordinarias sobre variables o como operaciones especiales

El contrato entre la memoria y el software dice que cuando el software realiza una adquisición, la memoria se asegurará de que todas las copias locales de las variables protegidas sean actualizadas de manera consistente con las remotas, en caso necesario. Al realizar tina liberación, las variables protegidas que hayan sido modificadas se propagan hacia las demás máquinas.

En general, una memoria distribuida compartida tiene consistencia de liberación si cumple las siguientes reglas:

- *Antes de realizar un acceso ordinario a una variable compartida, deben terminar con éxito todas las adquisiciones anteriores del proceso en cuestión.*
- *Antes de permitir la realización de una liberación, deben terminar la lectura y escrituras anteriores del proceso.*
- *Los accesos de adquisición y liberación deben ser consistentes con el procesador (no se pide la consistencia secuencial).*

En la consistencia de liberación normal, llamada **consistencia de liberación fuerte**, para distinguir la otra variante, a1 realizar una liberación, el procesador que realiza ésta expulsa todos los datos modificados hacia los demás procesadores que tienen una copia en caché y que podrían necesitarlos.

Consistencia de entrada.

Desde el punto de vista formal, una memoria exhibe la consistencia de entrada si satisface las siguientes condiciones

- *No se permite realizar un acceso de adquisición a una variable de sincronización con respecto de un proceso hasta que se realicen todas las actualizaciones de los datos compartidos protegidos con respecto de ese proceso.*
- *Antes de permitir la realización de un acceso en modo exclusivo a una variable de sincronización por un proceso, ningún otro proceso debe poseer la variable de sincronización, ni siquiera en modo no exclusivo.*
- *Después de realizar un acceso en modo exclusivo a una variable de sincroniza- ción, no se puede realizar el siguiente acceso en modo no exclusivo de otro proceso a esa variable de sincronización hasta haber sido realizado con respecto del propietario de esa variable.*

La primera condición dice que cuando un proceso realiza una adquisición, ésta podría no concluir (es decir, regresar el control al siguiente enunciado) hasta actualizar todas las variables compartidas protegidas.

La segunda condición dice que antes de actualizar una variable compartida, un proceso debe entrara una región crítica en modo exclusivo para garantizar que ningún otro proceso intenta actualizarla al mismo tiempo.

La tercera condición dice que si un proceso desea entrar a una región crítica en modo no exclusivo, primero debe verificar con el propietario de la variable de sincronización que protege la región crítica para buscar las copias más recientes de las variables compartidas protegidas.

5.3 Memoria compartida distribuida con base en páginas.

El elemento esencial en este caso es que ningún procesador puede tener acceso directo a la memoria de otro procesador. Tales sistemas reciben a veces el nombre NORMA (sin acceso a memoria remota) en contraste con los sistemas NUMA.

NUMA: Cada procesador puede hacer referencia de manera directa a cada palabra en el espacio global de direcciones, sólo leyendo o escribiendo en él.

Diseño básico.

En un sistema DSM, el espacio de direcciones se separa en pedazos, los cuales están dispersos en todos los procesadores del sistema. Cuando un procesador hace referencia a una dirección que no es local, ocurre un señalamiento, y el software DSM trae el pedazo que contiene la dirección y reinicia la instrucción suspendida.

Réplica.

Consiste en duplicar los pedazos exclusivos para lectura. Otra posibilidad consiste en duplicar todos los pedazos no sólo exclusivos para lectura. Mientras se realicen lecturas, en realidad no habrá diferencia entre la duplicación de un pedazo exclusivo para lectura y uno para lectura-escritura.

Granularidad.

Los sistemas DSM son similares a los multiprocesadores en varios aspectos fundamentales. En ambos sistemas, cuando se hace referencia a una palabra de memoria no local, se trae un pedazo de memoria con la palabra, desde su posición actual, y se coloca en la máquina que hace la referencia (en la memoria principal o el caché, respectivamente).

Obtención de la consistencia secuencial.

Si las páginas no se duplican, no se pretende lograr la consistencia. Existe con exactitud una copia de cada página, y ésta se desplaza de un lugar a otro de manera dinámica según se necesite.

Si sólo se tiene una copia de cada página, no existe el peligro de que las diferentes copias tengan valores diversos. Si se duplican las páginas exclusivas para lectura, tampoco existe problema alguno. Las páginas exclusivas para lectura nunca se modifican, de modo que todas las copias siempre son idénticas. Sólo se conserva una copia de cada página para lectura-escritura, por lo que también son imposibles las inconsistencias en este caso.

Búsqueda del propietario.

Consiste en realizar una transmisión, y solicitar la respuesta del propietario de la página especifica. Una vez que el propietario ha sido localizado de esta manera, el protocolo puede continuar de la manera anterior.

La optimización obvia consiste en no sólo preguntar quién es el propietario, sino también indicar si el emisor desea leer o escribir y si necesita una copia de la página. El propietario puede enviar entonces un mensaje, transfiriendo la propiedad y la página, según sea necesario.

La transmisión tiene la desventaja de interrumpir a cada procesador, obligándolo a inspeccionar el paquete de solicitud.

Búsqueda de las copias.

Consiste en transmitir un mensaje con el número de página y solicitar a todos los procesadores que contengan la página que la invaliden. Este método sólo funciona si los mensajes de transmisión son por completo confiables y nunca se pueden perder.

Otra posibilidad consiste en que el propietario o el controlador de páginas mantengan una lista del conjunto de copias, indicando los procesadores que poseen tal o cual página.

Reemplazo de página.

Una página duplicada que posee el proceso saliente. Basta transferir la propiedad a una de las otras copias, informando a ese proceso. Al controlador de páginas o a ambos, según la implantación. La propia página no tiene que transferirse, lo que produce un mensaje menor. Si ninguna de las páginas duplicadas es un candidato adecuado, hay que elegir una página no duplicada.(Tanenbaum, 2009)

5.4 Memoria compartida distribuida con variables compartidas.

Munin.

Munin es un sistema DSM que se basa fundamentalmente en objetos del software, pero que puede colocar cada objeto en una página aparte, de modo que el hardware MMU pueda utilizarse para detectar el acceso a los objetos compartidos.

El modelo básico utilizado por Munin es el de varios procesadores, cada uno de ellos con espacio de dirección lineal por páginas, en el que uno o más hilos ejecutan un programa multiprocesador con ligeras modificaciones. El objetivo del proyecto Munin es el de tomar los programas multiprocesadores existentes, realizarles cambios menores y hacerlos que se ejecuten de manera eficiente en los sistemas con multicomputadoras que utilicen una forma de DSM.

Las modificaciones consisten en anotar las declaraciones de las variables compartidas con la palabra reservada shared, de modo que el compilador las reconozca. También es posible que el programador especifique la colocación de variables compartidas del mismo tipo en Munin en la misma página. La mezcla de tipos no funciona, pues el protocolo de consistencia utilizado para una página depende del tipo de variables que estén en ella.

El acceso a las variables compartidas se logra mediante las instrucciones normales de lectura y escritura del CPU. Ni se utilizan métodos de protección especiales. Si ocurre un intento por utilizar una variable compartida que no esté presente, ocurre un fallo de página y el sistema Munin obtiene el control.

Consistencia de liberación Munin se basa en una implantación software de la consistencia de liberación. Lo que hace Munin es proporcionar las herramientas para que los usuarios estructuren sus programas en torno de las regiones críticas, definidas de manera dinámica mediante las llamadas de adquisición y liberación.

Las escrituras a las variables compartidas ocurren dentro de las regiones críticas; y las lecturas ocurren dentro o afuera. Mientras un proceso está activo dentro de una región crítica, el sistema no garantiza la consistencia de las variables compartidas, pero cuando sale de una región critica, las variables compartidas modificadas desde la última liberación son actualizadas en todas las máquinas.

Munin distingue tres clases de variables:

1. Las variables ordinarias no se comparten y sólo pueden ser leídas o escritas por el proceso que las creó.
2. Las variables de datos compartidos son visibles para varios procesadores y parecen secuencialmente consistentes, siempre que todos los procesos las utilicen sólo en las regiones críticas.
3. Las variables de sincronización, como las cerraduras o las barreras, son especiales, y sólo se puede tener acceso a ellas por medio de procedimientos de acceso proporcionados por el sistema, como lock y unlock para las cerraduras e increment y wait para las barreras.

Protocolos múltiples.

Además de utilizar la consistencia de liberación, Munin también utiliza otras técnicas para mejorar el desempeño. La principal de éstas consiste en permitir al programador que realice anotaciones en las declaraciones de las variables compartidas, clasificándolas dentro de alguna de las cuatro categorías siguientes:

1. Las variables exclusivas para lectura son las más sencillas. Cuando una referencia a una variable exclusiva para lectura provoca un fallo de página, Munin busca la variable en el directorio de variables, encuentra a su propietario, y solicita a éste una copia de la página requerida.
2. Las variables compartidas migratorias utilizan el protocolo de adquisición/liberación. Se utilizan dentro de las regiones críticas y están protegidas por variables de sincronización. La idea es que estas variables

emigren de una máquina a otra conforme se entre o salga de las regiones críticas.

3. Una variable de escritura compartida se utiliza cuando el programador indica que es seguro el hecho de que dos o más procesos escriban en ella al mismo tiempo; por ejemplo, un arreglo en el que los diferentes procesos tienen acceso concurrente a diversos subarreglos.

Directorios.

Munin utiliza directorios para localizar las páginas que contienen variables compartidas. Cuando ocurre un fallo por una referencia a una variable compartida. Munin dispersa las direcciones virtuales que provocaron el fallo, con el fin de determinar la entrada de la variable en el directorio de variables compartidas. A partir de esa entrada, ve la categoría de la variable, si existe una copia local, y quién es el probable propietario. El propietario es el último proceso que adquirió el acceso para escritura. Para una variable compartida migratoria, el propietario es el proceso que la posee por el momento.

Se lleva un registro del posible propietario mediante el siguiente algoritmo. Cuando se inicia un proceso en Munin, el proceso raíz posee todas las variables compartidas. Cuando el proceso P1, hace una referencia posterior a una variable compartida, ocurre un fallo, lo que genera un mensaje a la raíz para solicitarla.

Sincronización.

Munin mantiene un segundo directorio con las variables de sincronización. Éstas se localizan de manera similar a la forma de localizar las variables compartidas ordinarias. Desde un punto de vista conceptual, las cerraduras actúan como si fueran centralizadas, pero de hecho se utiliza una implantación distribuida para evitar el envío de tráfico excesivo a cualquier máquina.

Cuando un proceso adquiere una cerradura, primero verifica si él mismo posee la cerradura. Si lo hace y la cerradura está libre, se otorga la solicitud. Si la cerradura no es local, se localiza mediante el directorio de sincronización, el cual mantiene un registro del probable propietario. Si la cerradura está libre, se otorga. Si no está libre, el solicitante se agrega al final de la cola. De esta manera, cada proceso conoce la identidad del proceso siguiente en la cola. Al liberar una cerradura, el propietario la transfiere al siguiente proceso de la lista.

Midway.

Midway es un sistema con memoria distribuida compartida cuya base consiste en compartir las estructuras de datos individuales. Es similar a Munin en varios aspectos, pero tiene ciertas características nuevas, interesantes y propias. Su objetivo es permitir que los programas multiprocesadores existentes y los nuevos se ejecuten de manera eficiente en las multicomputadoras, con unos cuantos ligeros cambios en el código.

Consistencia de entrada.

La consistencia se mantiene pidiendo que todos los accesos a las variables compartidas y las estructuras de datos se realicen dentro de cierto tipo específico de sección crítica conocido como el sistema de tiempo de ejecución de Midway. Cada una de estas secciones críticas es protegida por una variable de sincronización especial, por lo general, una cerradura, pero también posiblemente una barrera. Cada variable compartida a la que se tiene acceso en una sección crítica debe estar asociada de manera explícita con la cerradura (o barrera) de esa sección crítica mediante una llamada a procedimiento. De esta forma, cuando se entra o sale de una sección crítica, Midway conoce con precisión las variables compartidas que podrían tener un acceso en un momento dado.

Para que funcione la consistencia de entrada, Midway requiere que los programas tengan tres características que no tienen los programas de multiprocesador:

1. Las variables compartidas deben declararse mediante la nueva palabra reservada shared.
2. Cada variable compartida debe estar asociada con una cerradura o barrera.
3. Sólo se puede tener acceso a las variables compartidas dentro de las secciones críticas.

Para hacer esto se requiere un esfuerzo adicional del programador. Si no se cumplen por completo estas reglas, no se genera un mensaje de error y el programa produce resultados incorrectos. Puesto que la programación de esta forma es propensa a errores, en especial al ejecutar antiguos programas multiprocesadores que nadie entiende en realidad, Midway también soporta la consistencia secuencia1 y la de liberación. Estos modelos requieren menor información detallada para la operación correcta.

Implantación.

Cuando se entra en una sección crítica, el sistema de tiempo de ejecución de Midway adquiere primero la cerradura correspondiente. Para obtener una cerradura exclusiva, es necesario localizar al propietario de la cerradura, que es el último proceso que la adquirió en forma exclusiva. Cada proceso lleva el registro del probable propietario, de la misma forma en que lo hacen IVY y Munin y sigue la cadena distribuida de propietarios sucesivos hasta que encuentra el actual.

Al momento en que se adquiere la cerradura, el proceso que la adquiere actualiza su copia de todas las variables compartidas. En el protocolo más sencillo, el propietario anterior sólo las envía todas. Sin embargo, Midway utiliza una optimización para reducir la cantidad de datos por transferir.

El uso de la consistencia de entrada implantada de esta forma tiene en potencia un excelente desempeño, puesto que la comunicación sólo ocurre cuando un proceso realiza una adquisición. Además, sólo se necesitan transferir aquellas variables compartidas no actualizadas. En particular, si un proceso entra a una región crítica, sale de ella y entra de nuevo, no se necesita comunicación alguna. Este patrón es común en la programación paralela, de modo que la ganancia en este caso es esencial.

5.5 Memoria compartida distribuida basada en objetos.

Los sistemas DSM basados en páginas que hemos estudiado hasta ahora utilizan MMU para señalar los accesos a las páginas faltantes. Aunque este método tiene ciertas ventajas, también tiene sus desventajas. En particular, los datos se organizan en objetos, paquetes, módulos u otras estructuras de datos, cada una de las cuales tiene una existencia independiente de las demás. Si un proceso hace referencia a una parte de un objeto, en muchos casos se necesitará todo el objeto, por lo que tiene sentido transportar los datos a través de la red mediante unidades de objetos, no unidades de páginas.

Si se avanza en la dirección de un modelo de programación de alto nivel, la programación de los sistemas DSM puede ser más sencilla y menos propensa a errores. El acceso a las variables compartidas y la sincronización mediante éstas también se integra de manera más clara. En algunos casos, también se introducen ciertas optimizaciones que son más difíciles de realizar en un modelo de programación menos abstracto.

Objetos.

Un objeto es una estructura de datos encapsulada definida por el programador, Consta de datos internos, el estado del objeto y procedimientos, llamados métodos u operaciones, que operan sobre el estado del objeto. Para tener acceso U operar sobre el estado interno, el programa llama a alguno de los métodos.

En una memoria distribuida compartida basada en objetos, los procesos de varias máquinas comparten un espacio abstracto ocupado por objetos compartidos, La localización y administración de los objetos es controlada de manera automática por el sistema de tiempo de ejecución. Este modelo contrasta con el de los sistemas DSM basados en páginas, como IVY, que sólo proporcionan una memoria lineal en bruto, con bytes desde O hasta algún máximo.

Cualquier proceso llama a los métodos de cualquier objeto, sin importar la posición del proceso del objeto. El sistema operativo y el sistema de tiempo de ejecución se encargan de que funcione la llamada a un método sin importar la posición del proceso o del objeto.

Una vez que se ha tomado la decisión de estructurar una memoria compartida como colección de objetos separados en vez de un espacio lineal de direcciones, si no se utiliza la réplica, todos los accesos a un objeto pasarán por una copia, lo que es sencillo, pero puede conducir a un desempeño pobre

En resumen, la memoria distribuida compartida basada en objetos ofrece tres ventajas sobre los otros métodos:

1. Es más modular que las demás técnicas.

2. La implantación es más flexible debido al control de los accesos.

3. La sincronización y el acceso se pueden integrar de manera limpia

Linda.

Linda proporciona los procesos en varias máquinas. Con una memoria distribuida compartida muy estructurada. El acceso a esta memoria es mediante un pequeño conjunto de operaciones primitivas que se agregan a los lenguajes existentes, como C y FORTRAN.

Este método tiene varias ventajas sobre un nuevo lenguaje. Una de las principales ventajas es que los usuarios no tienen que aprender un nuevo lenguaje. Esta ventaja no debe ser subestimada. Una segunda ventaja es la sencillez: el cambio de un lenguaje X a X-Linda se puede lograr al agregar unas cuantas primitivas a la biblioteca y adaptando el preprocesador de Linda que alimenta con programas Linda al compilador.

Espacio de n-adas.

El concepto unificador detrás de Linda es el de un espacio de n-adas abstracto. El espacio de n-adas es global en todo el sistema, y los procesos de cualquier máquina pueden insertar o eliminar n-adas en el espacio de n-adas sin importar la forma o el lugar donde estén guardados. La implantación real puede implicar a muchos servidores en varias máquinas,

Una n-ada es como tina estructura en C, consta de uno o más campos, cada uno de los cuales es un valor de cierto tipo soportado por el lenguaje base incluyen a los enteros, longint, los números de punto flotante, así como los tipos compuestos, como los arreglos.

Operaciones sobre las n-adas.

Linda no es un sistema basado en objetos por completo generales, puesto que sólo proporciona una cantidad fija de operaciones integradas y no hay forma de definir otras nuevas. Se proporcionan cuatro operaciones sobre las n-adas:

1.-La primera, out, coloca una n-ada: out ("abc",2,5)

En el espacio de n-adas. Coloca la n-ada ("abc",2,5) en el espacio de n-adas. Los campos de out son por lo general constantes, variables o expresiones

2.- in recuperan del espacio de n-adas: in("abc",2,?);

Esta operación "busca" en el espacio de n-adas una n-ada formada por la cadena "abc", el entero 2 y un tercer campo que contenga cualquier entero

El algoritmo de concordancia utilizado por in es directo. Los campos de la primitiva in, llamados plantilla, se comparan con los campos correspondientes de cada n-ada en el espacio de n-adas.

Ocurre una concordancia cuando se cumplen las siguientes tres condiciones:

1. La plantilla y la n-ada tienen el mismo número de campos.

2. Los tipos de los campos correspondientes son iguales.

3. Cada constante o variable en la plantilla concuerda con su campo en la n-ada.

Implantación de Linda.

Son posibles las implantaciones eficientes de Linda en varios tipos de hardware. Adelante analizaremos algunas de las más interesantes. Para todas las implantaciones, un preprocesador analiza el programa en Linda, extrae la información útil y la convierte al lenguaje básico en que debe estar, Una implantación eficiente de Linda debe resolver dos problemas:

1. La forma de simular el direccionamiento asociativo sin una búsqueda masiva.

2. La forma de distribuir la n-adas entre las máquinas y la forma de localizarlas posteriormente.

La clave para ambos problemas es observar que cada n-ada tiene una firma de tipo, que consta de la lista (ordenada) de los tipos de sus campos. Además, por convención, el primer campo de cada n-ada es por lo general una cadena que divide de manera eficaz al espacio de n-adas en su espacios ajenos nombrados por la cadena.

Por último, consideremos la implantación de Linda en sistemas que no tienen capacidad de transmisión alguna. La idea básica es separar el espacio de n-adas en su espacios ajenos, creando primero una partición para cada firma de tipo, y dividiendo después cada una de estas particiones de nuevo con base en el primer campo.

En potencia, cada una de las particiones resultantes puede ir a una máquina distinta, controlada por su propio servidor de n-adas, para difundir la carga. Cuando se realiza un in o un out, se determina la partición necesaria, y se envía un mensaje a esa máquina, para depositar una n-ada o para recuperar una.

La experiencia con Linda muestra que la memoria distribuida compartida se puede controlar de manera radicalmente diferente al movimiento de las páginas completas, como en los sistemas basados en páginas

Orca.

El sistema Orca consta del lenguaje, el compilador, y el sistema de tiempo de ejecución, quien en realidad controla los objetos compartidos durante la ejecución. Aunque el lenguaje, el compilador y el sistema de tiempo de ejecución se diseñaron para trabajar juntos, el sistema de tiempo de ejecución es independiente del compilador y se podría utilizar también para otros lenguajes. Después de una introducción al lenguaje Orca, describiremos la forma en que el sistema de tiempo de ejecución implanta una memoria distribuida compartida basada con objetos.

El lenguaje Orca.

En algunos aspectos, Orca es un lenguaje tradicional cuyos enunciados secuenciales se basan de manera vaga en Modula-2. Sin embargo, es un lenguaje con tipos seguros, sin apuntadores ni sobrenombres. Las cotas de los arreglos se verifican al tiempo de ejecución. Éstas y otras características similares eliminan o detectan muchos errores comunes de programación, como los almacenamientos forzados) en la memoria. Las características del lenguaje han sido elegidas con cuidado para facilitar varias optimizaciones.

Dos características de Orca importantes para la programación distribuida son los objetos-dato (o simplemente objetos) compartidos y el enunciado fork. Un objeto es un tipo abstracto de dato. Encapsula las estructuras dc datos internas y los procedimientos escritos por el usuario, llamados operaciones (o métodos) para que operen sobre las estructuras de datos internas.

Orca tiene un enunciado fork para crear un nuevo proceso en un procesador especificado por el usuario. El nuevo proceso ejecuta el procedimiento nombrado en el enunciado fork. Los parámetros, incluyendo los objetos, se pueden transferir al nuevo proceso, que es la forina en que se distribuyen los objetos entre las máquinas. Por ejemplo:

```
for i in 1..ii do fork foobar(s) on i; od;
```

Genera un proceso nuevo en cada una de las máquinas 1 a n, ejecutando el proceso foobar en cada una de ellas. Como estos nuevos procesos se ejecutan en paralelo, meten y sacan elementos de la pila compartidas como si estuvieran ejecutándose en un multiprocesador con memoria compartida. El sistema operativo debe encargarse de mantener la ilusión de la memoria compartida donde ésta realmente no existe.

Administración de los objetos compartidos en Orca.

La administración de objetos en Orca es controlada por el sistema de tiempo de ejecución. Funciona en las redes de transmisión. Y en las redes puntuales. El sistema de tiempo de ejecución controla la réplica, migración, consistencia y llamadas a operaciones relativas a los objetos. Cada objeto puede estar en uno de dos estados: única copia o duplicado. Un objeto en el estado de única copia sólo existe en una máquina.

Los objetos pueden pasar de un estado a otro durante la ejecución. La gran ventaja de duplicar un objeto en cada máquina es que las lecturas se pueden llevar a cabo de manera local, sin tráfico o retraso en la red. Si un objeto no se duplica, todas las operaciones deben ser enviadas al mismo, y el proceso que hizo la llamada debe bloquearse en espera de la respuesta. Una segunda ventaja de la réplica es un mayor paralelismo: se pueden realizar varias operaciones de lectura al mismo tiempo. Con copia única, sólo se puede realizar una operación a la vez, lo que hace más lenta la ejecución. La principal desventaja de la réplica eso es el costo de mantener consistentes todas las copias.

Ahora analizaremos el algoritmo para decidir si un objeto debe estar en una copia o duplicarse. Inicialmente, un programa Orca consta de un proceso, que tiene a todos los objetos. Cuando se bifurca, las demás máquinas son informadas de este evento y reciben las copias actuales de todos los parámetros compartidos del hijo.

Cada sistema de tiempo de ejecución calcula entonces el costo esperado por duplicar, o no, cada objeto. Para realizar este cálculo, se necesita conocer la proporción esperada de lecturas y escrituras. El compilador estima esta información mediante un examen del programa, tomando en cuenta que los accesos dentro de los ciclos cuentan más y los accesos dentro de los enunciados if cuentan menos que los demás accesos.

Los costos de comunicación también se factorizan en la ecuación. Por ejemplo, un objeto con una proporción lectura-escritura de 10 en una red de transmisión será duplicado, mientras que un objeto con una proporción lectura-escritura de I en una red puntual será colocado en un estado de copia única, donde esta copia está en la máquina que realiza más escrituras. Puesto que todos los sistemas de tiempo de ejecución realizan el misino cálculo, llegan a la misma conclusión. Si un objeto está presente en únicamente una máquina y necesita estar en todas, se disemina. Si está duplicado y ésta ya no es la mejor opción, todas las máquinas, excepto una, descartan su copia. Los objetos pueden emigrar mediante este mecanismo.

Sistemas distribuidos de archivos.

Un componente fundamental de cualquier sistema distribuido es el sistema de archivos. Como era el caso de los sistemas con un procesador, la tarea del sistema de archivos en los sistemas distribuidos es almacenar los programas y los datos y tenerlos disponibles cuando sea necesario. Muchos de los aspectos de los sistemas distribuidos de archivos son similares a los de los sistemas convencionales, por lo que no repetiremos ese material. En vez de esto, nos concentraremos en aquellos aspectos de los sistemas distribuidos de archivos distintos al caso centralizado.

Para comenzar, 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 de archivos 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 servidor de archivos es un proceso que se ejecuta en alguna máquina y ayuda a implantar el servicio de archivos. Un sistema puede tener uno o varios servidores de archivos, pero los clientes 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 pedidos. De hecho, los clientes ni siquiera deben saber que el servicio de archivos es distribuido. Lo ideal es que se vea como un sistema de archivos normal de un procesador.

Puesto que un servidor de archivos es por lo general un proceso del usuario (o a veces un proceso del núcleo) 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. Por ejemplo, un sistema distribuido podría tener dos servidores que ofrezcan el servicio de archivos en UNlX y el servicio de archivos en Ms-Dos, respectivamente, donde cada proceso usuario utilizaría el servidor apropiado. De esa forma, es posible que una terminal tenga varias ventanas y que en algunas de ellas se ejecuten programas en UNIX y en otros 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 número de servicios de archivo disponibles puede cambiar con la evolución del sistema. Por lo general, un sistema distribuido de archivos tiene dos componentes razonablemente distintos: el verdadero servicio de archivos v el servicio de directorios. El primero se encarga de las operaciones en los archivos individuales, como la lectura, escritura y adición, mientras que el segundo se encarga de crear y administrar directorios, añadir y eliminar archivos del directorio; el etc. En esta sección analizaremos la interfaz del verdadero servicio de archivos; en la siguiente analizaremos la interfaz del servicio de directorios.

La interfaz del servicio de archivos.

El aspecto fundamental para cualquier servicio de archivos, ya sea para un procesador o un sistema distribuido, es la pregunta: "¿Qué es un archivo?" Un archivo es una secuencia de bytes sin interpretación alguna. El significado y estructura de la información en los archivos queda a cargo de los programas de aplicación; esto no le interesa al 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 serie de registros, con llamadas al sistema operativo para leer o escribir un registro particular. Por lo general, se puede especificar el registro mediante su número (es decir, suposición dentro del archivo) o el valor de cierto campo. En el segundo caso, el sistema operativo mantiene al archivo como un árbol B o alguna otra estructura de datos adecuada, o bien utiliza tablas de dispersión para localizar con rapidez los registros. Puesto que la mayoría de los sistemas distribuidos están planeados para ambientes UNIX o MS-DOS, la mayoría de los servidores de archivos soporta el concepto de archivo como secuencia de bytes, en vez de una secuencia de registros con cierta clave.

Un archivo puede tener atributos, partes de información relativas a él 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 en alguno de los atributos. Por ejemplo, se pueden modificar los permisos de acceso pero no el tamaño (a menos que se agreguen datos al archivo). En unos cuantos sistemas avanzados, se podrían crear y administrar atributos definidos por el usuario además de los usuales.

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, no puede ser modificado. Se dice que tal archivo es inmutable. El hecho de contar con archivos inmutables facilita el soporte del ocultamiento y duplicación de archivos, puesto que esto elimina todos los problemas asociados con la actualización de todas las copias de un archivo cada vez que éste se modifique.

La protección en los sistemas distribuidos utiliza en esencia las mismas técnicas de los sistemas con un procesador: posibilidades y listas para control de acceso. Los servicios de archivos se pueden dividir en dos tipos, según si soportan un modelo carga/descarga o un modelo de acceso remoto. En el modelo carga/descarga el servicio de archivos sólo proporciona dos operaciones principales: la lectura de un archivo y la escritura del mismo. La primera operación transfiere todo un archivo de uno de los servidores de archivos al cliente solicitante. La segunda operación transfiere todo un archivo en sentido contrario, del cliente al servidor. Los archivos se pueden almacenar en memoria o en un disco local, como sea necesario. La ventaja del modelo carga/descarga es la sencillez del concepto. Los programas de aplicación buscan los archivos que necesitan y después los utilizan de manera local.

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

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 archivos y mover éstos de un directorio a otro. La naturaleza del servicio de directorios no depende del hecho de que los archivos individuales se transfieran en su totalidad o que se tenga un acceso remoto a ellos.

El servicio de directorios define un alfabeto y una sintaxis para formar los nombres de archivos (y directorios). 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. Por lo general, cada subdirectorio contiene todos los archivos de un proyecto, como un programa o documento de gran tamaño (por ejemplo, un libro). Cuando Se despliega el (sub)directorio, sólo se muestran los archivos relevantes; los archivos no relacionados están en otros (sub)directorios y no agrandan la lista. Los subdirectorios pueden contener sus propios subdirectorios y así en lo sucesivo, lo que conduce a un árbol de directorios, el cual se conoce como sistema jerárquico de archivos**.**

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

En un sistema distribuido existen varias máquinas y no se puede detener toda la actividad, por lo que es difícil, sino es que imposible, tomar una foto "instantánea". Un aspecto fundamental en el diseño de cualquier sistema distribuido de archivos es si todas las máquinas (y procesos) deben tener con exactitud la misma visión de la jerarquía de los directorios. Como ejemplo de lo que queremos decir en esta observación. Es flexible y tiene implantación directa, pero tiene la desventaja de que el sistema no se comporta como un sistema de tiempo compartido tradicional. En un sistema de tiempo compartido, el sistema de archivos se ve igual para todos los procesos

Transparencia de los nombres.

El principal problema de esta forma de los nombres es que no es por completo transparente. En este contexto, son relevantes dos formas de transparencia y es importante distinguirlas. La primera, la transparencia con respecto a la posición, significa que el nombre de la ruta de acceso no sugiere la posición del archivo (o de algún otro objeto). Una ruta como /servidor/ dirlldir2/x indica que x está localizado en el servidor l, pero no indica la posición del servidor. Éste es libre de moverse dentro de la red, sin que el nombre de la ruta de acceso deba ser modificado. Así, este sistema es transparente con respecto a la posición.

Sin embargo, supongamos que el archivo x es muy grande y que hay poco espacio en El servidor 1. Además, supongamos que hay mucho espacio en el servidor 2. El sistema podría desplazar de forma automática al servidor2. Por desgracia, si el primer componente de todas las rutas de acceso es el servidor, el sistema no puede desplazar el archivo al otro servidor en forma automática, aunque dir1 y dir2 existieran en ambos servidores. Un sistema distribuido que incluya los nombres de la máquina o el servidor en los nombres de las rutas de acceso no es independiente con respecto a la posición. Tampoco lo es uno basado en el montaje remoto, puesto que no es posible desplazar un archivo de un grupo de archivos (la unidad de montaje) a otro y conservar el antiguo nombre de la ruta de acceso. La independencia con respecto a la posición no es fácil de lograr, pero es una propiedad deseable en un sistema distribuido.

Para resumir lo anterior, existen tres métodos usuales para nombrar los archivos y directorios en un sistema distribuido:

1. Nombre máquina + ruta de acceso, como /maquina/ruta o maquina: ruta.
2. Montaje de sistemas de archivos remotos en la jerarquía local de archivos.
3. Un espacio de nombres que tenga la misma apariencia en todas las máquinas.

Nombres de dos niveles.

La mayoría de los sistemas distribuidos utilizan cierta forma de nombres con dos niveles. En un sistema con varios servidores de archivos, cada uno de los cuales esté auto-contenido (es decir, no tenga referencias a directorios o archivos en otros servidores), el nombre binario puede ser sólo un número de un nodo-i local, como en UNIX.

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. Un enlace simbólico 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.

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 escritura para evitar problemas. En los sistemas con un procesador que permiten a los procesos compartir archivos, 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. Un sistema distribuido donde todas las solicitudes de archivos deban pasar a un servidor con frecuencia es pobre. Este problema se puede resolver si se permite a los clientes que mantengan copias locales de los archivos de uso frecuente en sus cachés Una forma de salir de esta dificultad es propagar de manera inmediata todas las modificaciones de los archivos en caché de regreso al servidor. Aunque esto es sencillo desde el punto de vista conceptual, el método es ineficiente. Otra solución consiste en relajar la semántica de los archivos compartidos.

La dificultad final con el uso de cachés y la semántica de sesión es que viola otro aspecto de la semántica de UNIX además del hecho de que no todos los READ regresen el valor de escritura más reciente. En UNIX, a cada archivo abierto se le asocia un apuntador que indica la posición actual en el archivo. Una instrucción READ toma los datos a partir de esa posición y WRITE deposita los datos ahí. Este apuntador es compartido por los procesos que abrieron el archivo y todos sus hijos. Con la semántica de sesión, cuando los hijos se ejecutan en máquinas distintas, no se puede lograr compartir el archivo. Para ver las consecuencias del hecho de abandonar los apuntadores de archivo compartidos, consideremos un comando como

Un método por completo distinto a 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 KEAD. Lo que es posible es crear un archivo por completo nuevo e introducirlo en el sistema de directorios, con el nombre de un archivo ya existente, el cual se vuelve inaccesible (al menos con ese nombre). Así, aunque se vuelve imposible modificar el archivo x, es posible remplazarlo (en forma atómica) por un archivo nuevo. En otras palabras, aunque los *archivo nuevo* se pueden actualizar, los *directorios* sí. Una vez que hemos decidido que los archivos no se pueden modificar, desaparece el problema de enfrentarse a dos procesos, uno de los cuales escribe en un archivo y el otro lo lee.

Sigue presente el problema de qué hacer si dos procesos intentan remplazar el mismo archivo a la vez. Como en el caso de la semántica de sesión, parece que la mejor solución es permitir que uno de los nuevos archivos remplace al anterior. Un problema más molesto consiste en qué hacer si un archivo se remplaza mientras otro proceso está ocupado leyéndolo. Una solución es arreglárselas de tal forma que el lector utilice el archivo anterior. Una cuarta vía para enfrentar el uso de los archivos compartidos en un sistema distribuido es usar las transacciones atómicas.

Sistemas distribuidos de archivos.

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; es decir, cómo se ven ante el usuario. En esta sección veremos la forma en que se implantan dichos sistemas. Comenzaremos con la presentación de cierta información experimental acerca del uso de los archivos. Después revisaremos la estructura del sistema, la implantación del ocultamiento, la réplica y el control de la concurrencia. Por último, concluiremos con un breve análisis de algunas lecciones que nos ha dado la experiencia.

Uso de archivos.

Antes de implantar cualquier sistema, distribuido o no, es útil tener una buena idea de su posible uso, para garantizar la eficiencia de las operaciones de ejecución frecuente. Sin embargo, en primer lugar debemos hacer unas advertencias acerca de éstas y otras mediciones. Algunas de las mediciones son estáticas, lo que quiere decir que representan una toma instantánea del sistema en cierto momento. Las mediciones estáticas se realizan al examinar el disco y ver lo que hay en él. Entre ellas se encuentran la distribución de tamaños de los archivos, la distribución de tipos de archivos y la cantidad de espacio que ocupan los archivos de varios tamaños y tipos. Otras mediciones son dinámicas, se llevan a cabo al modificar el sistema de archivos, de modo que registre todas las operaciones en una bitácora .Estos datos proporcionan información con respecto a la frecuencia relativa de varias operaciones, el número de archivos abiertos en un momento dado y la cantidad de hechos compartidos. Al combinar las medidas estáticas y dinámicas, aunque sean diferentes en lo fundamental, obtenemos una mejor idea de la forma de uso del sistema.

Un problema siempre presente en las mediciones de cualquier sistema existente es saber qué tan típica es la población observada. Otro problema inherente en las mediciones es que se debe tener cuidado con las características que se miden en el sistema. Como ejemplo sencillo, al analizar la distribución de los nombres de archivo en un sistema MS-DOS, uno podría concluir con rapidez que los nombres de archivo nunca tienen más de ocho caracteres (más una extensión opcional de tres caracteres). Sin embargo, sería un error concluir de ahí que son suficientes ocho caracteres, puesto que nadie utiliza más de ocho. Puesto que MS-DOS no permite más de ocho caracteres en un nombre de archivo, es imposible decir lo que harían los usuarios si no tuvieran tal restricción en la longitud.

Una observación interesante es que la mayoría de los archivos tienen tiempos de vida cortos. En otras palabras, un patrón común es crear un archivo, leerlo (una vez) y después eliminarlo. Un ejemplo común podría ser el de un compilador que crea archivos temporales para la transmisión de información entre sus distintas fases. Esto implica que sería una buena idea crear el archivo en el cliente y mantenerlo ahí hasta su eliminación. Esto elimina una cantidad importante de tráfico entre el cliente y el servidor.

El hecho de que unos cuantos archivos se compartan argumenta en favor del ocultamiento por parte del cliente. Como hemos visto, el ocultamiento complica la semántica, parte del cliente y aceptar las consecuencias de la semántica de sesión en favor de un mejor desempeño. Por último, la clara 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 diversas partes, pero es raro que se modifiquen, por lo que tal vez se podrían duplicar en varias partes, aunque esto implique 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 réplica no sirve de mucho. Es posible compartir los archivos ordinarios de datos, por lo que éstos requieren de otro tipo de manejo.

Estructura del sistema.

En ciertos sistemas no existe distinción alguna entre un cliente y un servidor. Todas las máquinas ejecutan el mismo software básico, de modo 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 puedan tener acceso a ellos.

En otros sistemas, el servidor de archivos y el de directorios son sólo programas del usuario, por lo que se puede configurar un sistema para que ejecute o no el software de cliente o servidor en la misma máquina, como se desee. Por último, en el otro extremo están los sistemas donde los clientes y los servidores son máquinas esencialmente distintas, ya sea en términos de hardware o software. Los servidores y clientes pueden ejecutar incluso versiones distintas del sistema operativo. Aunque la separación de funciones es un poco más transparente, no existe razón fundamental para preferir un método por encima de los demás.

Un segundo aspecto de la implantación 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 servidor, que maneje todas las llamadas a directorios y archivos. Sin embargo, 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 (por ejemplo, máquina + nodo-i) y después ir hasta el servidor de archivos con el nombre en binario para llevar a cabo la lectura o escritura real del archivo.

El argumento en favor de la separación es que las dos funciones no tienen relación real entre sí, por lo que es más flexible mantenerlas separadas. Por el momento, consideremos el caso de servidores de archivos y directorios independientes. En el caso normal, el cliente envía un nombre simbólico al servidor de directorios, que a su vez regresa el nombre en binario que comprende el servidor de archivos. Sin embargo, es posible que una jerarquía de directorios se reparta entre varios servidores.

La búsqueda de nombres de rutas de acceso todo el tiempo, en particular si se utilizan varios servidores de directorios, puede ser cara. Algunos sistemas intentan mejorar su desempeño al mantener un caché de indicadores, es decir, de nombres buscados de manera reciente, así como los resultados de esas búsquedas. Al abrir un archivo, se verifica si el caché contiene esa ruta de acceso. En caso afirmativo, se omite la búsqueda directorio por directorio y la dirección del binario se obtiene del caché.

Para que funcione el ocultamiento de los nombres, es esencial que cuando se utilice de manera inadvertida un nombre binario obsoleto, se le informe de esto al cliente de alguna manera, para que pueda recurrirá la búsqueda directorio por directorio para encontrar el archivo y poder actualizar el caché. Además, para que tenga algún beneficio el ocultamiento de los indicadores, éstos deben ser correctos la mayor parte del tiempo. Si se cumplen estas condiciones, el ocultamiento de indicadores puede ser una poderosa técnica aplicable a muchas áreas de los sistemas operativos distribuidos.

El aspecto estructural final a considerar es si los servidores de archivos, directorios o de otro tipo deben contener la información de 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 información de estado de los clientes entre las solicitudes. Después de todo, los sistemas operativos centralizados mantienen la información de estado de los procesos activos.

Para comprender mejor la diferencia, consideremos un servidor de archivos con comandos para abrir, leer, escribir y cerrar archivos. Después de abrir un archivo, el servidor debe mantener la información que relacione los clientes con los archivos abiertos por éstos. Por lo general, al abrir un archivo, el cliente recibe un descriptor de archivo o algún otro número que se utiliza en las llamadas posteriores para identificación del archivo. Al recibir una solicitud, el servidor utiliza el descriptor de archivo para determinar el archivo necesario. La tabla que asocia los descriptores de archivo con los archivos propiamente dichos es información de estado.

En el caso de un servidor sin estado, cada solicitud debe estar auto contenida. 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, de ser posible, quedará por completo a cargo de los clientes. En consecuencia, los servidores sin estado tienden a ser más tolerantes de las fallas que los que mantienen los estados, lo cual es un argumento a favor de los primeros.

Tendencias en los sistemas distribuidos de archivos.

La mayoría de los sistemas de archivos de la actualidad organizan los archivos como una colección de bloques, ya sea como un árbol (por ejemplo, UNIX) o como una lista ligada un almacenamiento del archivo en forma adyacente en la memoria, en vez de separarlo en bloques. Es más fácil llevar un registro de los archivos almacenados en forma adyacente, además de que se pueden transmitir más rápido en la red. La razón de que los archivos adyacentes no se utilicen en los discos es que, si un archivo crece, su desplazamiento hacia un área del disco con más espacio es una operación cara. Por el contrario, el desplazamiento de un archivo a otra área de la memoria es una operación factible.

Sin embargo, los servidores de archivos en la memoria principal presentan un serio problema. Si se interrumpe la energía eléctrica, se pierden todos los archivos. A diferencia de los discos, que no pierden la información por una falla en la energía, la memoria principal se borra al eliminar la electricidad. La solución sería hacer respaldos continuos o por incrementos en cinta de video. Con la tecnología actual

Un desarrollo en hardware que puede afectar a los sistemas de archivos es el disco óptico. En un principio, estos dispositivos tenían la propiedad de que sólo se podía escribir en ellos una vez (haciendo marcas en la superficie mediante un láser), pero no podían modificarse.

A veces se les conocía como dispositivos WORM (write once, read many, una escritura y muchas lecturas). Algunos de los actuales discos ópticos utilizan Láser que afectan la estructura del cristal del disco, pero no lo dañan, por lo que se pueden borrar. Los discos ópticos tienen tres propiedades importantes:

1. Son lentos.
2. Tienen un enorme espacio de almacenamiento.
3. Tienen acceso aleatorio.

También son relativamente baratos, aunque más caros que las cintas de video. Las primeras dos propiedades son iguales a las de las cintas de video, pero la tercera abre la siguiente posibilidad. Imaginemos un servidor de archivos con un sistema de archivos den gigabytes en la memoria principal y un disco óptico de n gigabytes como respaldo. Cuando se crea un archivo, se guarda en la memoria principal y se señala que aún no tiene un respaldo.

Todos los accesos son a través de la memoria principal. Cuando la carga de trabajo es baja, los archivos que no hayan sido respaldados todavía se transfieren al disco óptico de manera secundaria, de manera que el byte que de la memoria vaya a dar al byte que del disco. Como el primer esquema, lo que tenemos aquí es un servidor de archivos en la memoria principal, pero con un dispositivo de respaldo conveniente y una asociación uno aun con la memoria. Otro desarrollo interesante en hardware son las redes de fibras ópticas de alta velocidad.

Pero supongamos que podemos equipar al sistema con un servidor de archivos en la memoria principal y una red de fibras ópticas de alta velocidad. Podría ser factible deshacerse dcl caché del cliente y del disco del servidor, para operar con la memoria de éste último, con respaldos en el disco óptico. Esto simplificaría es mucho el software.

Al estudiar el uso de cachés del cliente, vimos que gran parte del problema es provocada por el hecho de que si dos clientes ocultan el mismo archivo y uno de ellos lo modifica, el otro no descubre esto, lo cual conduce a ciertas inconsistencias. Un poco de reflexión en torno a este tema revelará que la situación es análoga a los cachés de memoria en un multiprocesador. Sólo que en este caso, cuando un procesador modifica una palabra compartida, se envía una señal de hardware a través del bus de la memoria a los demás cachés, con el fin de permitirles que invaliden o actualicen dicha palabra. Esto no se hace en los

Sistemas distribuidos de archivos.

¿Por qué no se hace esto? La razón es que las interfaces actuales en la red no soportan tales señales. Sin embargo, podría ser posible construir interfaces de red que lo hicieran. El procesador ha actualizado recientemente el archivo. La activación de un bit hace que la interfaz cree y envíe un paquete a través del anillo que verifique y active el bit correspondiente en las demás interfaces.

Si el paquete recorre todo el camino sin encontrar otras máquinas que intenten utilizar el archivo, algún otro registro en la interfaz toma también el valor 1. En caso contrario, toma el valor O. De hecho, este mecanismo proporciona una forma para cerrar el archivo en forma global en todas las máquinas, en unos cuantos microsegundos. Después de establecer la cerradura, el procesador actualiza el archivo. Se anota cada uno de los bloques modificados del archivo (por ejemplo, mediante el uso de bits en la tabla de páginas). Al terminar la actualización, el procesador limpia el bit del mapa de bits, cual hace que la interfaz de la red localice el archivo mediante una tabla en memoria y que deposite en forma automática todos los bloques modificados.

Es claro que esta sencilla solución se puede mejorar de varias formas, pero muestra la forma en que un hardware bien diseñado puede resolver problemas difíciles de administrar a nivel de software. Es probable que los futuros sistemas distribuidos sean apoyados por Hardware especializado de varios tipos.

Escalabilidad.

Una tendencia definida en los sistemas distribuidos es hacia los sistemas cada vez grandes. Esta observación tiene implicaciones para el diseño de los sistemas distribuidos de archivos. Los algoritmos que funcionan bien para los sistemas con 100 máquinas pueden trabajar un poco mal para los sistemas con 1 000 máquinas y no funcionar para sistemas con 10 000 máquinas. Para los principiantes, los algoritmos centralizados no se escalan bien. Si la apertura de un archivo necesita el contacto con un servidor centralizado para registrar el hecho de que el archivo está abierto, ese servidor se convertirá en un cuello de botella en cierto momento de crecimiento del sistema. La forma general de enfrentar este problema es separar el sistema en unidades más pequeñas e intentar que cada una de ellas sea relativamente independiente de las demás. Si se tiene un servidor por cada unidad de asignación. El hecho de que todos los servidores registren todas las aperturas podría ser aceptable bajo estas circunstancias.

Las transmisiones son otra área problemática. Si cada máquina realiza una transmisión por cada segundo, con n máquinas, hay un total de n transmisiones eri la red por cada segundo, lo cual genera n2 interrupciones. Es claro que si n crece, esto se puede convertir en un problema.

Redes de área amplia

En el futuro, muchos sistemas distribuidos basados en LAN serán conectados entre sí, para formar sistemas distribuidos transparentes a través de países y continentes. Por ejemplo, la PTT de Francia está colocando una pequeña computadora en cada departamento y casa de ese país. Aunque el objetivo inicial

es eliminar la necesidad de las operadoras de información y los directorios telefónicos. Aunque las máquinas francesas son idénticas, en la mayoría de las redes de área amplia existe gran variedad de equipo. Así, un sistema distribuido de área amplia

Necesita enfrentarse a la heterogeneidad. Esto hace que surjan preguntas tales como la forma de almacenar un archivo de caracteres si no todos utilizan ASCII, o el formato que se debe utilizar para los archivos que contienen números de punto flotante, si existen varias representaciones de éstos. También es importante el cambio esperado en las aplicaciones. La mayoría de los Sistemas distribuidos experimentales que se construyen en las universidades se centran en la programación los propios investigadores todos los días (al menos mientras no se encuentran en reuniones de comités o escribiendo propuestas para apoyos económicos).

Tolerancia de fallas.

Los sistemas de cómputo de la actualidad, excepto por algunos muy especializados, como los que se utilizan para el control del tráfico aéreo, no son tolerantes de fallas. Cuando la computadora falla, se espera que los usuarios acepten esto como un hecho de la vida. Por desgracia, la población en general espera que las cosas funcionen. Si un canal de televisión, el sistema telefónico o la compañía de luz eléctrica fallan durante media hora, al otro día existen muchas personas descontentas. Con la difusión de los sistemas distribuidos, crecerá la demanda de sistemas que esencialmente nunca fallen. Los sistemas actuales no Pueden cumplir ese requisito.

Es claro que tales sistemas necesitarán una considerable redundancia en el hardware y la infraestructura de comunicación, pero también la necesitarán en el software y particularmente en los datos. La réplica de archivos, que a menudo es una idea tardía en los sistemas distribuidos actuales, será un requisito esencial en los sistemas futuros. También se tendrán que diseñar los sistemas de modo que puedan funcionar cuando sólo se disponga de una parte de los datos, puesto que la insistencia en la disponibilidad de todos los datos no conduce a la tolerancia de fallas. Los tiempos de falla que ahora consideran aceptables los programadores y otros usuarios complejos, lo serán cada vez menos, al difundirse en computadoras menos especializadas.

BIBLIOGRAFÍA

belarmino. (13 de 05 de 2015). *Diapositiva del modelo Osi*. Obtenido de http://galeon.com/belarmino/modeloosi.html

CLAROS, I. (13 de 05 de 2015). *belarmino.galeon.com hispavista*. Obtenido de url: http://belarmino.galeon.com/

Edicion, A. S. (2009). *Sistemas Operativos Modernos.* México: pearson educacion .

FalcomHive. (17 de 05 de 2015). *blogspot.mx*. Obtenido de http://marcosventuraosorio261v.blogspot.mx/2009/03/sistemas-operativos-de-red-y-sistemas.html

Laura. (10 de mayo de 2015). *Modo de transferencia asíncrona/síncrona*. Obtenido de http://es.slideshare.net/lauriz19cour/modo-de-transferencia-asncronasncrona-atm

Martínez, D. L. (s.f.). *Sistemas Operativos .* Universidad Regional del Nordeste.

Martínez, M. D. (2015). *Sistemas Operativos.* 05: 18.

Modo de Transferencia Asíncrona ATM. (10 de mayo de 2015). Obtenido de http://www.angelfire.com/planet/netstechnology/atm.htm

NANCY, A. E. (10 de mayo de 2015). *Comunicación en los sistemas operativos distribuidos*. Obtenido de http://sistemasoperativosequipo5.blogspot.mx/2011/07/unidad-2-comunicacion-en-los-sistemas.html

Rojo, J. O. (18 de 05 de 2015). *http://augcyl.org/*. Obtenido de http://augcyl.org/: http://augcyl.org/?page_id=231

Sergio. (17 de 05 de 2015). *SlideShared*. Obtenido de http://es.slideshare.net/sergiooney/sistemas-operativos-distribuidos-13355041

TANENBAUM. (18 de 05 de 2015). *Sistemas Operativos Distribuidos*. Obtenido de http://arminluer.cl/archivos/8voSemestre/TecnologiasServiciosInternet/Libros/Sistemas%20Operativos%20Distribuidos%20TANENBAUM.pdf

Tanenbaum, A. A. (s.f.). *Sistemas Operativos Distribuidos.* phh,prentice hall.

Tanenbaum, A. S. (2009). *SISTEMAS OPERATIVOS DISTRIBUIDOS.* PRENTICE HALL.

Tanenbaum, A. S. (2009). *SISTEMAS OPERATIVOS DISTRIBUIDOS.* PRENTICE HALL.

Tanenbaum, A. S. (s.f.). *SISTEMAS OPERATIVOS DISTRIBUIDOS .*

Tenenmaum, A. S. (1996). Sistemas Operativos Distribuidos. Estado de México, Naucalpan de Juarez.