CAPITULO 25

DISTRIBUTED FILE SYSTEM

25.1. - Introducción. [10][11][S.O.]

En cualquier sistema informático y sus aplicaciones es necesario poder mantener y recuperar la información. Para esto es necesario organizarla y administrarla, con ese fin se provee como solución el almacenamiento en unidades denominadas archivos (files) en discos y otras unidades externas.

En el diseño de cualquier sistema operativo la administración de ellos es una de las partes más trascendentales. En el caso de un sistema operativo centralizado no es tan complicada dicha administración. Sin embargo, en uno distribuido no ocurre lo mismo.

En el caso de un sistema distribuido es importante diferenciar dos conceptos fundamentales:

Servicio de archivos: Es la especificación de los servicios que el file system provee a sus clientes, comprendiendo:

- Primitivas disponibles.
- Parámetros que utilizan dichas primitivas.
- Acciones que las primitivas proveen.

Para los clientes especifica claramente con qué servicios pueden contar, pero no dice nada acerca de cómo se lo implementa. Es decir, el servicio de archivos especifica la interfase del sistema de archivos de los clientes.

Servidor de archivos (de aquí en adelante server): Es un proceso que se ejecuta en alguna máquina contribuyendo a la implementación propia del servicio de archivos.

Si bien un sistema puede tener varios servers, los clientes no deberán conocer ni la cantidad de ellos ni su ubicación, es decir se deberá mantener la transparencia en el diseño e implementación de un file system.

Un sistema puede contener varios servidores de archivos (por ejemplo, un proceso de usuario ejecutándose en el kernel de una máquina) y cada uno puede ofrecer un servicio de archivos diferente.

25.2. - DISEÑO DE LOS FILE SYSTEMS [10][11][S.O.]

En general existen dos componentes:

- El servicio propio de archivos: Encargándose de las operaciones en los archivos individuales (creación, borrado, etc.).
- El servicio de directorios: Realiza la administración a nivel de directorios (crear, eliminar directorios, etc.).

25.3. - EL SERVICIO DE ARCHIVOS [10][11][S.O.]

Qué es un archivo? En un sistema como UNIX o DOS un archivo es una secuencia de bytes sin interpretación alguna. El significado y estructura de la información queda a cargo del programa de aplicación que la accede.

En sistemas mainframes existen muchos tipos de archivos con diferentes propiedades. En general se especifica el registro del archivo por medio de un número que es su posición dentro del archivo o por el valor de algún campo. En el segundo caso (DOS) el archivo se construye como un árbol u otra estructura como ser una tabla que indica la ubicación (usualmente dispersa) de los distintos registros.

La mayoría de los sistemas distribuidos soporta el concepto de archivo como secuencia de bytes en lugar de una secuencia de registros con cierta clave.

Otro aspecto importante es si los archivos se pueden modificar después de creados. Lo normal es que si, pero en algunos sistemas distribuidos las únicas operaciones sobre archivos son CREATE y READ. Estos archivos se denominan **inmutables**. Este tipo de archivo facilita el soporte para ocultamiento (caching) y replicación de archivos, ya que elimina la necesidad de actualizar todas las copias del archivo.

La protección se implementa con los mismos métodos estudiados anteriormente :

- listas de capacidades, o
- listas de control de acceso

Por ejemplo UNIX con los bits que controlan la lectura, escritura y ejecución para el dueño, el grupo del dueño y otros usuarios es una lista de accesos simplificada.

Los servicios de archivos se pueden dividir en dos tipos según la forma de acceso a los archivos, el modelo Upload/Download y el modelo de Montaje remoto.

25.3.1. - Upload/Download Model [10][11][S.O.]

Las operaciones básicas provistas (lectura y escritura de un archivo) se basan en el concepto del traslado por completo de la información, pudiendo almacenarse el archivo en memoria del cliente o en su un disco en forma local.

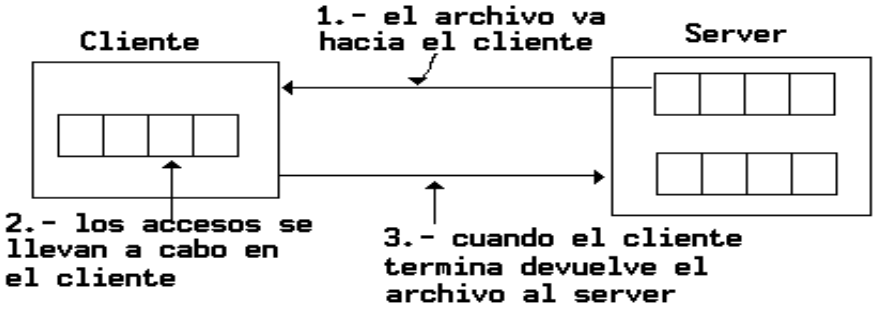


Fig. 25.1. - Modelo Upload/Download.

25.3.2. - Remote Access Model (montaje remoto). [10][11][S.O.]

Aquí lo que se realiza son solicitudes al server que es donde el archivo permanece sin realizarse la transferencia por completo de la información, sino que solamente se opera con el archivo a través de operaciones que se proveen.

Las solicitudes que se realizan tienen una gran variedad de posibilidades, por ejemplo: abrir y cerrar un archivo, leer y escribir partes de un archivo, moverse a través de un archivo, examinar y modificar atributos del archivo, etc.

Las ventajas de ambos métodos son: la sencillez del Upload/Download ya que la aplicación sólo busca los archivos que necesita y los utiliza de manera local, en tanto que en el Remote Access Model existe muy poca necesidad de espacio en los clientes ya que no transfiere los archivos en forma completa sino que solo lo hace con aquellas partes que precisa.

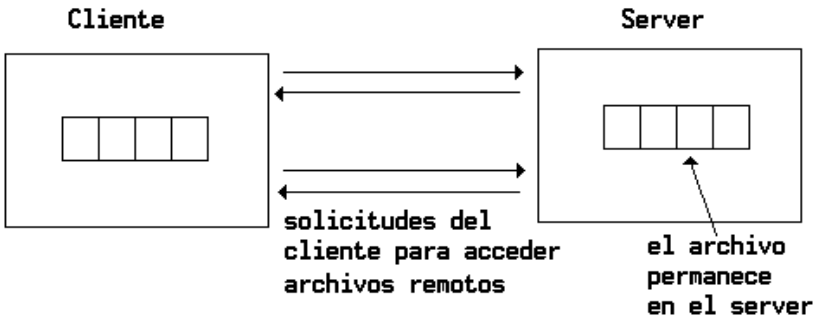


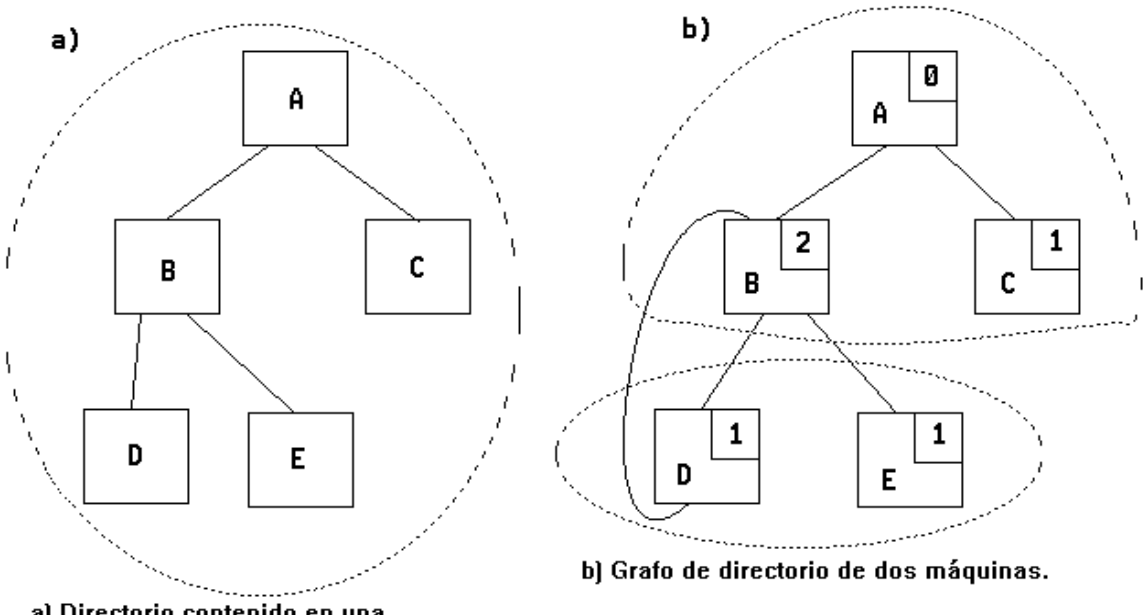
Fig. 25.2. - Modelo Remote Access.

25.4. - EL SERVICIO DE DIRECTORIOS [10][11][S.O.]

En este tipo de servicios son provistas todas las operaciones necesarias para operar a nivel de directorios y la movilidad de los archivos entre los diferentes directorios.

Para nombrar los archivos y directorios (se pueden ver como archivos a su vez con algún carácter que los identifique propiamente por ejemplo) el servicio provee un alfabeto y un conjunto de reglas (sintaxis) para definirlos.

Como es conveniente poder agrupar los archivos con características comunes (por ejemplo porque perte-



a) Directorio contenido en una máquina.

b) Grafo de directorio de dos máquinas.

Fig. 25.3.

necen a un usuario determinado o son archivos ejecutables de uso público), el servicio de directorios provee operaciones para la creación y borrado de directorios.

A su vez cada directorio puede contener subdirectorios dando lugar de esta manera a una **estructura jerárquica de tipo árbol de directorios**, el que se conoce como sistema jerárquico de archivos.

En algunos sistemas es posible la utilización de apuntadores a un directorio arbitrario, con lo cual es posible construir **gráficas** arbitrarias de directorios. Si bien éstas son mucho más poderosas que los árboles, acarrean implícitamente un peligro mayor como es el poder crear directorios huérfanos al eliminar un directorio.

Se puede establecer que en una estructura de directorios jerárquica se puede eliminar un enlace con un directorio si el directorio apuntado está vacío. En una gráfica se permite la eliminación de un enlace mientras exista al menos otro enlace. Se utiliza para ello un contador de enlaces en cada directorio que cuenta cuántos enlaces existen que apuntan al directorio actual.

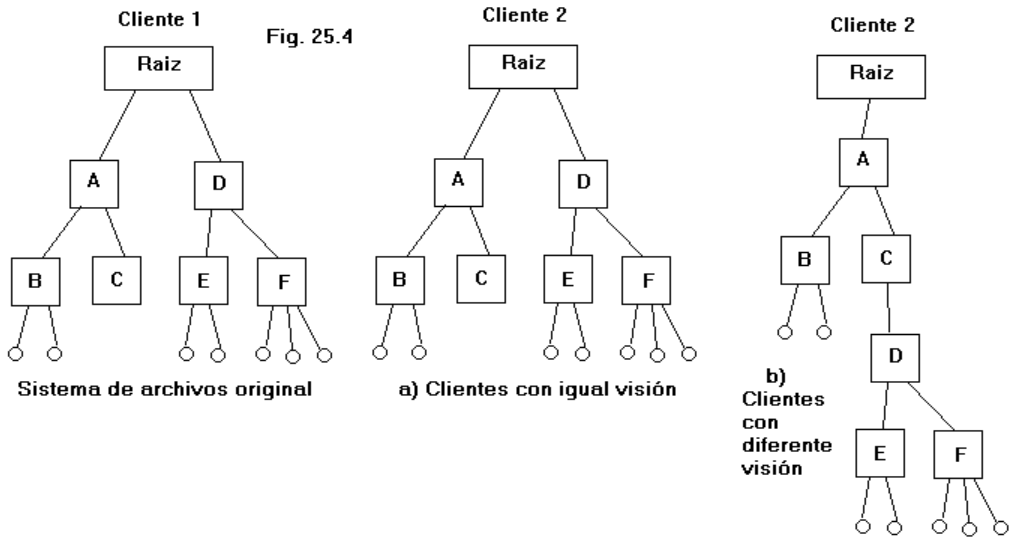
No obstante si en la gráfica de directorios de la figura 25.3 b) se elimina el directorio A, el contador de enlaces de B pasa a 1 pero los directorios B, D y E junto con sus archivos se convierten en huérfanos.

En un sistema distribuido este problema es mucho más grave y difícil de detectar y detener, ya que es casi imposible tener una visión instantánea en todo momento de lo que está sucediendo en todo el sistema (recordar el deadlock visto en los capítulos anteriores).

Otro punto a considerar y de singular importancia en el diseño de un sistema distribuido es la visión jerárquica del sistema de directorios que debe existir en todo momento; siendo las posibilidades que todos los clientes tengan la misma visión o no del file system.

El caso en que no tienen la misma visión generalmente corresponde al montaje remoto (Fig. 25.4 caso a) siendo su ventaja la flexibilidad y su implementación casi directa, en tanto que en el otro caso el file system se ve igual para todos los procesos como en un sistema de tiempo compartido y es fácil de comprender y programar (Fig. 25.4 caso b).

Fig. 25.4



25.5. - TRANSPARENCIA de los NOMBRES [10][11][S.O.]

Sin embargo es importante mantener la transparencia respecto de la ubicación y la independencia respecto a la posición. Con la primera se nota el hecho de que no es posible aún conociendo el path saber la ubicación del archivo, en tanto que en la segunda se dice que un sistema es independiente de la posición si los archivos pueden desplazarse de una máquina a otra sin que cambien sus nombres.

Un sistema que incluye los nombres del servidor o la máquina en el nombre de la ruta de acceso no es independiente con respecto a la ubicación.

Tampoco un sistema con Montaje Remoto por qué no es posible mover un archivo desde un grupo (unidad de montaje) a otro y conservar el antiguo nombre de la ruta de acceso.

Como consecuencia de lo dicho anteriormente existen tres métodos de nombrar archivos y directorios en un sistema distribuido:

- Nombre máquina + path (ruta de acceso) Ej: /maquina/path/file1.
- Montaje de archivos remotos en la jerarquía local.
- Espacio de nombres con la misma apariencia en todas las máquinas.

Los dos primeros son sencillos de implementar y sirven para conectar otros file systems ya existentes que no están diseñados para uso distribuido. El tercero es más difícil pero es necesario si se quiere tener la visión de una única máquina.

25.6. - NOMBRES de DOS NIVELES [10][11][S.O.]

En cualquier sistema de computación existen dos niveles de nombres: los **simbólicos** para uso por parte de los usuarios y los **binarios** que son los nombres con los cuales el sistema realiza la referencia a los archivos. Los directorios hacen una asociación entre estos dos nombres.

En el caso de un sistema autocontenido (es decir, no tiene referencias a directorios o archivos en otros servidores) el último tipo de nombres nombrado puede ser un i-nodo local.

Las alternativas son varias:

a) El nombre binario indica el server y el nombre del archivo en ese servidor.
b) El uso de un enlace de tipo simbólico como una entrada de directorio asociada a una cadena (server, nombre del archivo). El enlace en sí es solo el nombre de una ruta de acceso.
c) Otra posibilidad es utilizar capacidades como nombres binarios. El buscar un nombre en ASCII da como resultado una capacidad que puede tener varias formas. Por ejemplo: puede tener el número físico o lógico de una máquina o la dirección en la red en el server apropiado así como un número que indique el archivo específico.

Otra cosa que sucede en un sistema distribuido y ocurre raramente en un centralizado es que al buscar un nombre ASCII se pueden obtener no uno sino varios nombres binarios (i-nodos o capacidades).

Estos pueden ser el archivo original y todos sus backups. Con estos nombres se puede empezar a buscar uno de los archivos y si no está disponible por alguna razón intentar con otro. Esto provee un cierto grado de tolerancia a fallas a través de la redundancia.

25.7. - SEMÁNTICA de ARCHIVOS COMPARTIDOS [10][11][S.O.]

Al permitir compartir archivos entre 2 o más usuarios surge el inconveniente de tener que definir en qué forma se realizarán las lecturas del archivo y las escrituras y como los cambios que realiza un usuario serán visibles o no a los otros usuarios. Esto es lo que se denomina **semántica de archivos**.

Las semánticas posibles son cuatro, a saber:

- UNIX.
- Sesión.
- Archivos inmutables.
- Transacciones.

A continuación analizaremos cada una de ellas brevemente.

25.7.1. - Semántica UNIX [10][11][S.O.]

Establece un orden absoluto respecto al tiempo ya que secuencia las lecturas y escrituras, esto quiere decir que si se realiza un READ luego de un WRITE las modificaciones son vistas.

En el caso de un sistema donde exista un único servidor y no existen archivos ocultos por parte de los clientes, UNIX los procesa en forma secuencial. En el caso que exista ocultamiento (o sea si existe uso de caches locales en las cuales los clientes copian los archivos del servidor) entonces se obtiene un valor obsoleto en el READ luego del WRITE (ver Fig. 25.5).

Para mantener las modificaciones en los archivos se mantiene un apuntador al archivo abierto de tal manera que al realizar un WRITE se colocan los datos en el servidor.

Existe también el problema de la demora en la red debido a la cual un READ realizado luego de un WRITE llega al servidor antes que éste último.

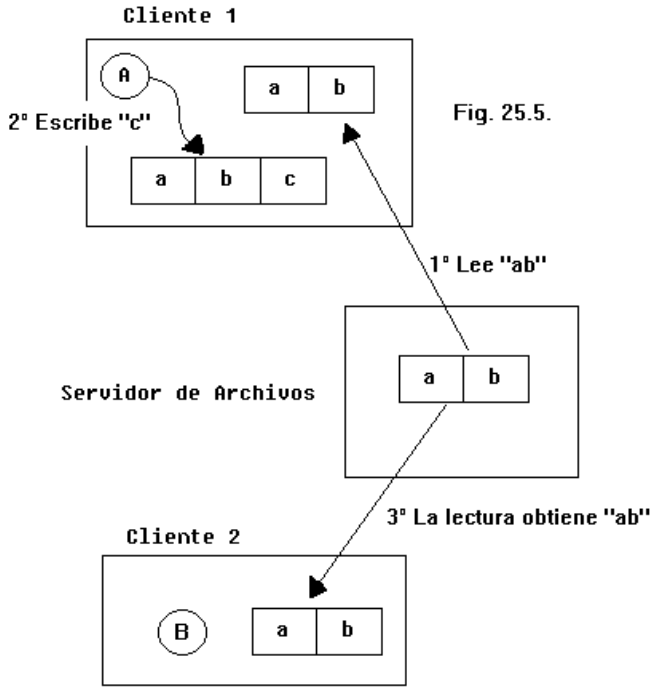


Fig. 25.5.

25.7.2. - Semántica de Sesión [10][11][S.O.]

En el caso anterior si `las modificaciones se devuelven inmediatamente al servidor se logra coherencia pero el método es ineficiente, para evitar ello se 'relaja la semántica' de modo tal que las modificaciones son vistas al cerrar los archivos produciendo de esta manera la semántica de sesión.

En este caso al cerrar un archivo la versión modificada que se verá depende de la implementación. El mantenimiento de un puntero a las modificaciones realizadas en este caso no es posible ya que éstas solamente se verán al cerrar los archivos.

25.7.3. - Semántica de archivos inmutables [10][11][S.O.]

En este tipo de semántica las únicas operaciones permitidas son: CREATE y READ, por lo cual los archivos no se actualizan, sin embargo los directorios sí. Si dos procesos intentan reemplazar el mismo archivo entonces el procedimiento es el mismo que en el caso de la semántica de sesión, se procede ad-hoc dependiendo de esta manera de la implementación utilizada.

Otro problema ocurre cuando al reemplazar un archivo existe otro proceso que ya estaba leyendo el archivo anterior. Se puede permitir al lector que siga utilizando la copia anterior aunque ya no existe en el directorio. Otra solución es detectar la modificación del archivo y hacer que fallen las lecturas posteriores.

25.7.4. - Semántica de transacciones atómicas [10][11][S.O.]

En ésta se utiliza la misma idea que en las transacciones atómicas es decir vale la propiedad del todo o nada, con lo cual las modificaciones serán vistas al cierre del archivo.

25.8. - IMPLEMENTACIÓN de un SISTEMA DISTRIBUIDO de ARCHIVOS [10][11][S.O.]

25.8.1. - Uso de archivos [10][11][S.O.]

Antes de implementar cualquier sistema es útil tener una buena idea de su posible uso para garantizar la eficiencia de las operaciones. En tal sentido, Satyanarayanan realizó en el año 1981 un estudio significativo de los patrones de uso de los archivos en un sistema universitario. En éste las mediciones realizadas fueron del tipo estáticas y dinámicas. Los resultados obtenidos por él fueron más tarde verificados por Mullendor y Tanenbaum (1984) en otro estudio. En este tipo de estudios se trató de determinar diferentes características como ser tipo de acceso a los archivos (lectura, escritura), el tipo de archivos accedidos, la frecuencia, etcétera.

En este tipo de estudios es necesario tener cuidado con las mediciones obtenidas respecto a lo que ha sido medido y cómo fueron realizadas dichas mediciones ya que a priori, no existe razón alguna para asumir que estos estudios valen en otro tipo de ámbitos como puede ser el empresarial.

25.9. - ESTRUCTURA del SISTEMA [10][11][S.O.]

En el caso de un sistema distribuido ¿existe alguna característica que permite diferenciar los servers de los clientes?.

Existen sistemas en los que no existe diferencia entre los clientes y los servers y en otros sí. Sin embargo no existe razón alguna para preferir un tipo de sistema a otro.

En el caso de los primeros la máquina exporta los nombres de los directorios seleccionados de tal modo que todas las otras máquinas pueden tener acceso a ellos. También existen sistemas en los cuales el file server y el directory server son solo programas del usuario quien escoge el software a ejecutar (cliente o servidor).

Otra diferencia existente es la forma de estructurar el servicio a los archivos y a los directorios ya que es posible combinar a ambos o separarlos. En éste último caso existe más funcionalidad ya que no existe relación entre los servicios.

Por ejemplo, para abrir un archivo habrá que ir al servidor de directorios para asociar el nombre simbólico (por ejemplo, máquina + i-nodo) y después ir al servidor de archivos con el nombre binario para realizar la lectura o escritura del archivo.

Un ejemplo de esto puede ser tener un servidor de directorios en UNIX y otro servidor de directorios en DOS y sin embargo un solo servidor de archivos para el almacenamiento físico, aunque esto tiene el problema de que al haber dos servidores de directorios se necesita mayor comunicación.

Los servicios de directorios pueden estar a su vez partidos en varios servidores de directorios como se muestra en le figura 25.6. En este esquema hay dos formas de obtener la información para llegar al archivo a/b/c.

Una es que el cliente realice la consulta al primer servidor que es quien controla su directorio de trabajo y este le devuelva como respuesta un apuntador al otro servidor y al repetir la consulta al servidor b obtener el apuntador a c quien finalmente devuelve el nombre binario (caso a).

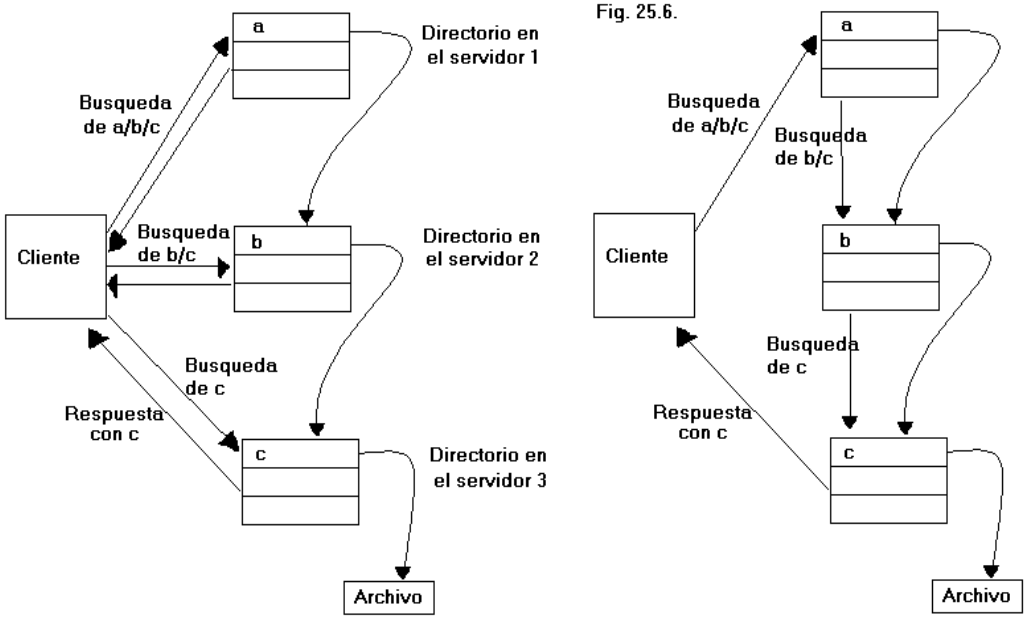
La segunda forma es más eficiente pero no puede implementarse por la RPC común ya que el que responde es un servidor distinto al que se le hizo el pedido originalmente.

Las operaciones de búsqueda son caras y por ello se trata de mejorar su desempeño por ejemplo utilizando caché de nombres recientemente utilizados para evitar la búsqueda en el directorio. Para que esto funcione es esencial que cuando el cliente utilice nombres obsoletos se le informe inmediatamente para que recurra a la búsqueda en el directorio.

Finalmente otro aspecto a considerar en la estructura es si el servidor contiene o no la información de los clientes en cuanto a las solicitudes hechas por éstos. Los métodos en estos casos se denominan **server con estado** (sí mantienen la información) o **sin estado** (en caso contrario).

En el caso de los con estado para mantener la información de los clientes es necesario que el server mantenga tablas y las actualice con las nuevas informaciones a medida que evoluciona el sistema.

Fig. 25.6.



En un servidor sin estado cuando el cliente envía una solicitud el servidor la satisface, envía la respuesta y elimina de sus tablas internas toda la información relativa a dicha solicitud. No guarda tampoco información alguna relativa a los clientes entre las solicitudes.

En los sin estado la solicitud debe ser autocontenida, o sea debe contener todo el nombre del archivo y el offset dentro del mismo para que el servidor pueda satisfacerle. Esto aumenta la longitud del mensaje.

Cada método tiene sus ventajas y sus desventajas sin existir mayores ventajas de un método sobre el otro.

SERVER CON ESTADO	**SERVER SIN ESTADO**
Puede existir lectura adelantada.	No necesita llamadas OPEN/CLOSE.
Es posible la cerradura de archivos.	No existe límite para el número de archivos abiertos.
Mejor desempeño.	Tolerancia de fallas.
Fácilmente aplicable la idempotencia.	No se desperdicia espacio en el server con tablas de mantenimiento.
Mensajes de solicitud más cortos.	No existen problemas al fallar un cliente.
Cuando el servidor se cae se pierde toda la información y la recuperación queda a cargo de los clientes	Mensajes de solicitud más largos
Si el cliente se cae el servidor no sabe si eliminar o no las entradas abiertas inactivas	Si el cliente se cae no hay problema para el servidor

25.9.1. - OCULTAMIENTO (caching) [10][11][S.O.]

En el modelo cliente-servidor los lugares para almacenar los archivos son:

- El disco servidor.
- La memoria del servidor.
- El disco cliente (si existe).
- La memoria cliente.

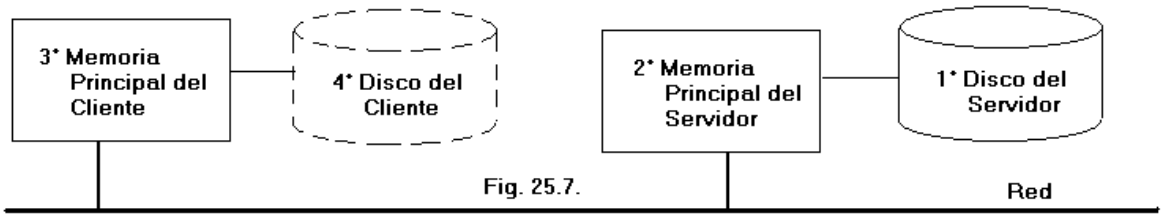


Fig. 25.7.

El disco servidor tiene la ventaja que todos los clientes ven el archivo. En contrapartida su desventaja es que existen problemas en el desempeño. Antes que el cliente pueda leer el archivo debe ser transferido del disco del servidor a la memoria del servidor y de allí a través de la red a la memoria del cliente y estas transferencias llevan cierto tiempo.

Este desempeño puede mejorarse si se utiliza la memoria del servidor en su lugar con los archivos de más reciente uso, proceso denominado **ocultamiento (caching).** Esto elimina la transferencia del archivo desde disco. No obstante es necesario algún modo para determinar qué archivos deberán permanecer en la cache.

Por otra parte, como el server puede mantener copias actualizadas de cache en su disco, entonces no existen problemas de consistencia.

El ocultamiento sin embargo puede hacerse en el cliente eliminando la transferencia desde la memoria del servidor, pero ello ocasiona problemas ya que el uso de la memoria o su disco es un problema de espacio vs. desempeño.

En general los sistemas que utilizan el disco del cliente lo hacen en memoria principal. Para ubicar su posición existen 3 opciones:

a)- Ocultar archivos en el espacio de direcciones del proceso usuario. Lo usual es que el caché sea administrado por la biblioteca de llamadas al sistema. Al operar con el archivo la biblioteca del sistema mantiene los de más uso en algún sitio para su acceso. Al terminar el proceso todos los archivos modificados si escriben en el server. Sirve si los procesos abren y cierran varias veces un archivo, pero en general no es así.

b)- Cache en el núcleo. Su desventaja es que es necesario realizar llamadas al kernel en todo momento. Su ventaja es que el cache sobrevive al finalizar el proceso. Por ejemplo un compilador de dos etapas que en su primera etapa deja el archivo listo para la linkedición no debe recurrir al servidor al necesitárselo en la segunda etapa. Si el núcleo administra la cache, entonces decide la cantidad de memoria necesaria.

c)- Colocar una cache en un proceso independiente a nivel del usuario. Su ventaja es que libera al microkernel del código del sistema de archivos, es más fácil de programar y resulta más flexible. Si el proceso administrador del cache ejecuta en una máquina con memoria virtual algunas páginas podrían ser removidas de la memoria principal del cliente perdiendo de este modo sentido el ocultamiento. Sin embargo esto se evitaría si el proceso puede fijar las páginas en memoria principal.

En general la cantidad de RPC's es mayor si existe ocultamiento. Sin embargo si la red de transmisión es muy rápida entonces las RPC's adicionales de las llamadas no consumirán mucho tiempo por lo que el desempeño dependerá de la tecnología existente.

25.9.2. - CONSISTENCIA [10][11][S.O.]

Ocultando a través del cliente se produce inconsistencia, por lo que la solución es adoptar la semántica apropiada.

El problema mayor se produce en la escritura del archivo. Para eliminar la inconsistencia es necesario el uso de la escritura a través del cache (write through); mediante el cual una modificación hecha en ella se envía de inmediato al server por lo que al leer el archivo desde el server se obtiene el valor más reciente.

Otro problema que se visualiza en la figura 25.8 puede ocurrir cuando desde el momento en que la máquina A obtuvo el dato otro cliente lo actualizó y cuando el proceso P3 lo solicita obtiene una versión desactualizada del mismo. Su solución puede ser que el administrador del caché en A verifique al servidor antes de entregar el archivo al cliente si el mismo está actua-

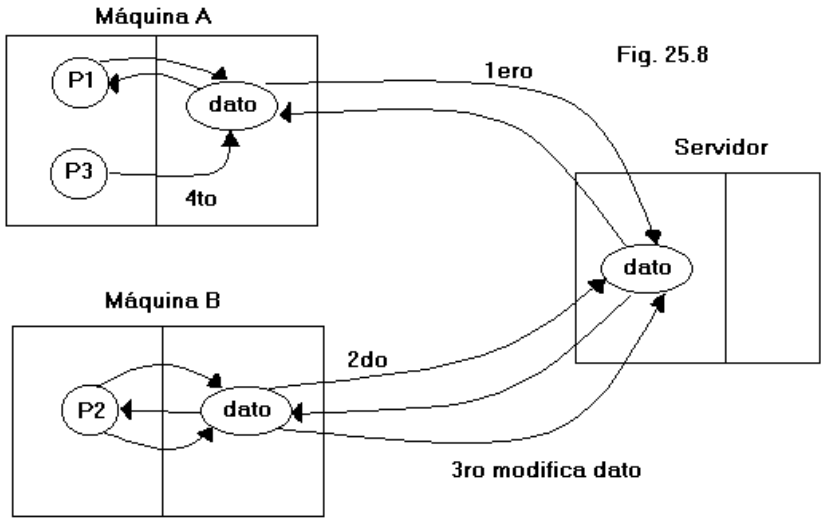


Fig. 25.8

lizado, ya sea por medio de versiones o a través de marcas temporales.

En el caso del write through para mejorar el desempeño el tráfico en la red en las escrituras es igual que en el caso de no-ocultamiento, entonces lo que se hace es realizar la escritura cada cierto intervalo de tiempo ya que leer un bloque grande es más eficiente que varios pequeños (escritura demorada). Esto obscurece la semántica ya que si otro proceso lee el archivo lo que verá dependerá de la sincronización de los eventos.

El siguiente paso es utilizar la semántica de sesión y solo escribir el archivo una vez que éste se cierra (escritura al cierre).

Otro método consiste en utilizar un algoritmo de control centralizado por el cual al abrir un archivo se le envía un mensaje al server el que mantiene un registro de los archivos abiertos y se administra su apertura según se quiera utilizar el archivo para lectura o escritura. Si un proceso quiere usar el archivo para escritura el servidor puede informar a los clientes que lo tienen en sus cachés que invaliden sus copias antes de otorgar el acceso. Al cerrar el archivo se informa al server para actualizar la información.

Método	**Comentarios**
Escritura a través del caché	Funciona pero no afecta el tráfico de escritura
Escritura demorada	Mejor desempeño pero es posible que la semántica sea ambigua
Escritura al cierre	Concuerda con la semántica de sesión
Control centralizado	Semántica UNIX, pero no es robusto y es poco escalable
Algoritmos para administrar el ocultamiento del cliente	

25.10. - IMPLEMENTACIONES DE FILE SYSTEMS [10][11][S.O.]

A continuación presentaremos y analizaremos dos implementaciones de file systems utilizados en diferentes sistemas distribuidos:

NFS (Network File System)

AFS (Andrew File System)

25.11. - NFS [10][11][S.O.]

NFS es el sistema de archivos implementado por Sun Micro Systems. Este sistema es la especificación e implementación de un sistema de software para acceso a archivos remotos a través de LANs o WANs.

Una de sus grandes ventajas es que opera en un amplio rango de máquinas, sistemas operativos, entornos y arquitecturas de red.

25.11.1. - Introducción [10][11][S.O.]

NFS ve el conjunto de workstations interconectadas como un conjunto de máquinas independientes con file systems independientes. Su gran ventaja es que permite compartir los datos entre estos file systems con una gran transparencia. Esta compartición está basada en la relación cliente-servidor que existe entre las distintas máquinas y puede ser establecida entre cualquier par de ellas ya que cualquier máquina puede ser cliente y servidor.

Dado que un directorio remoto es accesible desde cualquier máquina, ésta debe realizar un montaje primero para poder accederlo y este se vuelve parte de la jerarquía de directorios en la máquina del cliente. La semántica para ello es que el directorio a montar lo hace sobre el directorio del file system local. Por lo cual una vez realizada dicha operación, el directorio montado forma parte del subárbol local reemplazando al subárbol existente hasta ese momento en ese punto de montaje. A partir de aquí los usuarios pueden acceder a los archivos remotos en forma totalmente transparente aunque es necesario realizar la operación de montaje en forma no transparente.

Para ilustrar el montaje consideremos el file system de la figura 25.9 donde tenemos tres File Systems independientes en las máquinas U, S1 y S2.

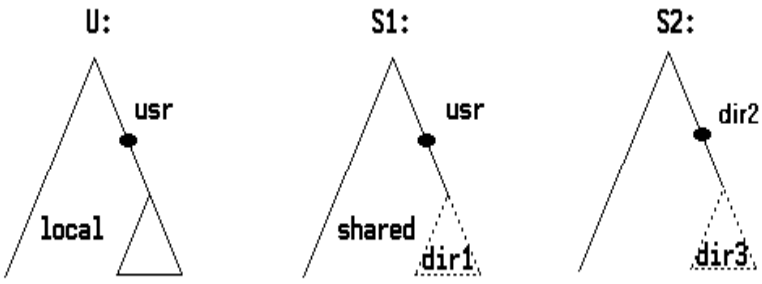


Fig. 25.9. - File Systems independientes.

En la figura 25.10 podemos ver los resultados de haber realizado el montaje de S1:/usr/d1 sobre U:/usr/local. Los usuarios de U pueden acceder a los archivos dentro de d1 utilizando el path /usr/local/d1 en U una vez completado el montaje. A su vez este directorio no es visible desde fuera.

No obstante, no existe transitividad ya que el cliente que monta un FS remoto no obtiene acceso a los FS montados sobre éste en la máquina remota.

Dos propiedades interesantes que ofrece esta implementación son:

Los montajes en cascada: es decir, montar un file system sobre otro también montado. En la Fig. 25.10 se montó sobre el directorio montado desde S1 al dir3 de S2.

La movilidad: esto es si un F.S. compartido es montado sobre los directorios home de los usuarios en todas las máquinas del sistema, un usuario puede loguearse en cualquier estación de trabajo y tener su entorno propio.

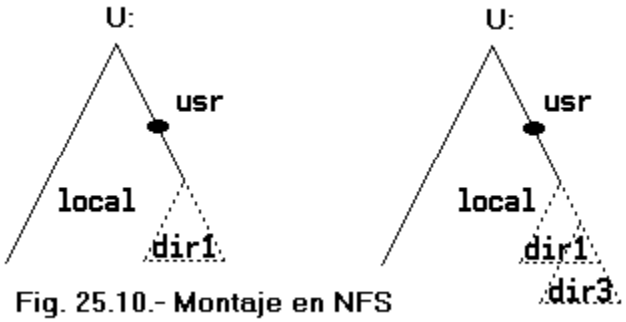


Fig. 25.10.- Montaje en NFS

La especificación de NFS distingue dos tipos de protocolos para realizar las distintas operaciones de montaje. los cuales se hallan implementados a través de primitivas de RPC's construidas sobre un protocolo de representación externa de datos (XDR).

Los protocolos son :

- Mount Protocol y
- NFS Protocol.

25.11.2.- El Mount Protocol [10][11][S.O.]

Este protocolo es utilizado para establecer la conexión lógica inicial entre el servidor y el cliente. En la implementaron de Sun cada máquina tiene un proceso servidor ejecutando fuera del kernel que hace las funciones del protocolo.

Una operación de montaje requiere el nombre del directorio remoto a ser montado y el nombre del servidor que almacena dicho directorio. Al pedido de montaje se lo relaciona con el correspondiente RPC y es enviado al servidor. Este mantiene una lista de exportaciones que especifica los file systems locales exportados para montaje y las máquinas permitidas para ello.

El servidor al recibir un pedido de montaje verifica dicha lista y en caso de tener el permiso adecuado retorna un **file handle** al cliente el cual es la unidad componente de todo pedido.

El file handle contiene toda la información que el server necesita para distinguir un archivo individual que se guarda.

En términos de UNIX, el file handle consiste de un identificador del archivo dentro del sistema y un número de i-nodo para identificar exactamente al directorio montado dentro del FS exportado.

El server mantiene también una lista de las máquinas clientes y sus correspondientes montajes para uso administrativo.

Usualmente un sistema tiene un esquema de montaje estático en tiempo de booteo el cual puede variar a posteriori.

Es importante recalcar que la operación de montaje solo afecta la visión del cliente y no la del server.

25.11.3. - El NFS Protocol [10][11][S.O.]

Este protocolo provee un conjunto de RPC's para realizar las operaciones remotas correspondientes a:

- Lectura de un conjunto de entradas de directorio.
- Lectura y escritura de archivos.
- Búsqueda de un archivo dentro de un directorio.
- Acceso a los atributos de un archivo.
- Manipulación de links y directorios.

De más esta decir que dichos procedimientos pueden ser invocados una vez obtenido el handle del directorio correspondiente.

Los servidores de NFS no mantienen la información de los clientes desde un acceso al siguiente, es decir, son sin estado.(nótese que no existen operaciones OPEN ni CLOSE).

El diseño resultante es robusto ya que no hay que tomar medidas especiales en caso de la caída del servidor.

Si bien antes mencionamos que el servidor mantiene una lista de las exportaciones hechas a los clientes lo cual es contrario a la modalidad de servidores sin estado, esto no afecta la correcta operación de NFS ya que no es necesario restaurar la lista luego de una caída del servidor.

Una implicación importante de los servidores sin estado y la sincronía de RPC's es que los datos modificados deben ser 'comiteados' antes que sean devueltos al cliente. Si bien de este modo se pierden las ventajas de tener caches locales de los clientes, en caso que el server caiga, los datos se mantendrán íntegros.

Una llamada a un procedimiento para grabar un archivo en NFS se garantiza que es atómica y no se mezcla con otras llamadas para grabar el mismo archivo. Sin embargo, el protocolo de NFS no provee mecanis-

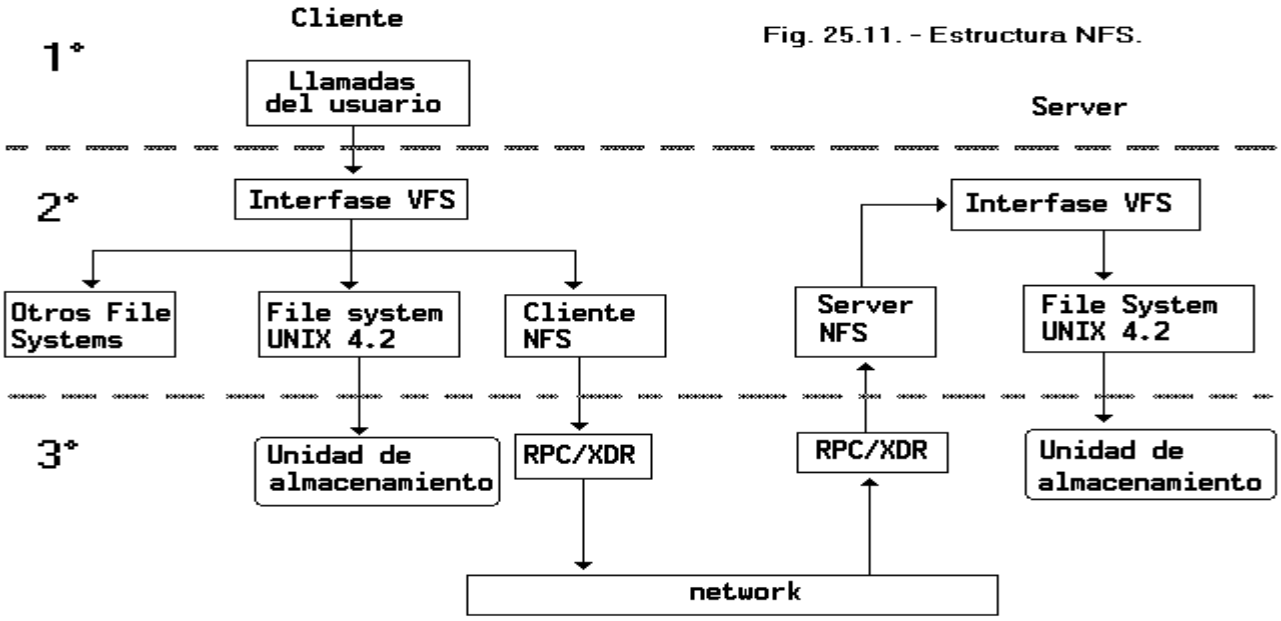


Fig. 25.11. - Estructura NFS.

mos de control concurrentes para el caso de las escrituras de un mismo archivo, y ya que una operación de grabación puede descomponerse en varias RPC's, dos o más usuarios que graben sobre el mismo archivo en forma remota pueden obtener sus datos entremezclados.

La moraleja es que ya que el bloqueo de archivos es una operación netamente con estado se debe proveer un servicio por fuera de NFS que administre este problema (por ejemplo Sun OS lo provee).

25.11.4. - ARQUITECTURA NFS [10][11][S.O.]

Esta arquitectura que puede visualizarse en la figura 25.11 consta de 3 capas: UNIX file system, Virtual file system y NFS propiamente dicha. A continuación resumiremos las características y funciones de cada una de ellas.

25.11.4.1. - UNIX file system [10][11][S.O.]

Compuesta por las operaciones open, read, write y close y los file descriptors.

25.11.4.2. - Virtual file system [10][11][S.O.]

Define una interfase que separa las operaciones genéricas sobre el file system de su implementación. Pueden coexistir varias implementaciones en una misma máquina para permitir un acceso transparente a los distintos tipos de file systems montados localmente.

NFS se basa en una estructura para representar los archivos denominada **vnodo** el cual contiene un identificador numérico que permite identificar al archivo en forma única a lo largo de la red (recuérdese que los i-nodos de UNIX son únicos dentro de un solo file system).

El VFS distingue archivos locales de remotos y los archivos locales se diferencian de acuerdo al tipo de file system. En forma similar a UNIX, el kernel mantiene una tabla en la que guarda la información de los detalles del montaje en el que tomó parte como cliente. Más aún, los vnodos de cada directorio que fue montado se mantienen en memoria constantemente de manera tal que los requerimientos para esos directorios serán relacionados con el correspondiente file system vía la tabla de montaje. Esencialmente las estructuras de vnodo conjuntamente con las tablas de montaje proveen un puntero para cada archivo hacia el file system del que desciende así como hacia el file system sobre el cual está montado.

Las operaciones que utiliza el VFS son:

- Específicas del file system para manejar accesos locales de acuerdo al tipo de file system.
- Invocar al protocolo de NFS para accesos remotos.

25.11.4.3. - NFS propiamente dicha [10][11][S.O.]

Es la capa más baja e implementa el protocolo NFS. Se la denomina la capa de servicio NFS.

25.11.5. - PATH-NAME TRANSLATION [10][11][S.O.]

Supongamos estar en una máquina U. Podemos montar un file system realizando las correspondientes operaciones y sobre éstos realizar otro montaje.

Cada cliente tiene una visión única de su propio espacio lógico de nombres que varía según los montajes que realizó y por lo tanto se hace necesario realizar Path-Name-Translation.

El path-name-translation es 'cortar' los nombres de los file systems montados realizando una llamada (look-up) para cada par montado(nombre de componente y directorio vnodo).

Esto causa un RPC separada para cada server. Podría haber sido más eficiente manejar el path al servidor y recibir un vnodo cada vez que se encuentra un punto de montaje. Pero debido al montaje en cascada el servidor sin estado no sabría que ese montaje en el cliente está realizado sobre otro montaje anterior.

El cache del cliente guarda los nombres completos de los vnodos informados por el servidor para agilizar las operaciones.

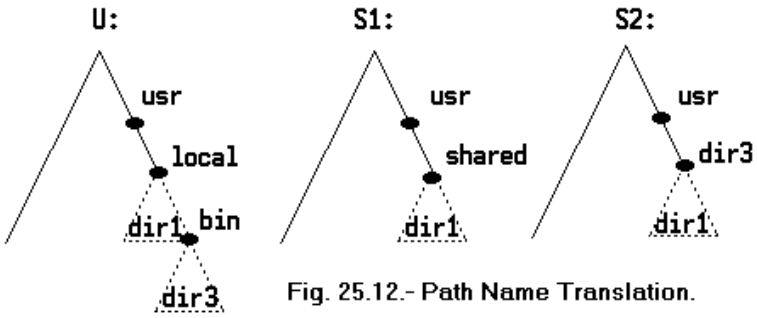


Fig. 25.12.- Path Name Translation.

25.11.6. - OPERACIONES REMOTAS [10][11][S.O.]

Las RPC's del protocolo NFS tienen su correspondencia una-a-una con las llamadas al sistema en UNIX, excepto para la apertura y cierre de archivos.

No existe correspondencia directa entre las operaciones remotas y las RPC's. En lugar de ello los bloques de archivos y sus atributos son obtenidos por las RPC's (fetch) y guardados localmente en la cache.

Los caches se dividen en **file blocks** y **file atributes** (información de i-nodo). En una apertura de archivo el kernel cliente chequea con el servidor si debe cargar los datos o si puede revalidar los atributos cargados en cache. Los bloques del archivo se usan solo si los atributos en cache son válidos.

Los atributos en caché se actualizan cada vez que llega nueva información del servidor. Se los descarta luego de 3 segundos para el caso de archivos y cada 30 segundos si son atributos de directorios.

Se utilizan técnicas de lectura anticipada y escritura demorada hasta que el server confirmó que escribió en el disco. La semántica UNIX no se preserva ya que la escritura demorada es bloqueante aún cuando el archivo haya sido abierto concurrentemente.

Es difícil caracterizar la semántica NFS. Los archivos recién creados en una máquina no serán visibles hasta tanto no transcurran 30 segundos. No se puede determinar si el grabar un archivo en un cliente será visible a otros clientes que tengan ese archivo abierto en lectura y si alguien lo abre en este momento solo verá aquellos cambios que hayan sido enviados al servidor.

Por ello NFS no posee una estricta emulación de la semántica UNIX ni de la semántica de sesión, no obstante lo cual es muy utilizada en sistemas operativos distribuidos.

25.12. - AFS [10][11][S.O.]

25.12.1. - Introducción [10][11][S.O.]

Andrew File System fue desarrollado desde 1983 en la Universidad de Carnegie-Mellon. Una de sus más formidables características es su escalabilidad ya que está pensado para soportar cerca de 5000 estaciones de trabajo (workstations).

25.12.2. - Panorama [10][11][S.O.]

AFS diferencia entre máquinas clientes (usualmente llamadas workstations) y máquinas servidores dedicados. Los clientes y servidores corren UNIX 4.2BSD y están interconectados por una LAN.

Los clientes tienen el espacio de nombres particionado en dos: local y compartido. Los servers dedicados, llamados en su conjunto VICE (nombre que proviene del software que ellos ejecutan), presentan a los clientes el

espacio de nombres compartido como homogéneo, idéntico y de jerarquía de archivos de ubicación transparente. El espacio local es el root file system del cual desciende el espacio compartido.

Las workstations ejecutan el protocolo VIRTUE para comunicarse con el VICE y precisan de discos locales para guardar su espacio local. Este, si bien es pequeño, contiene la suficiente información para su auto-operación.

Mirando con un poco más de detalle los clientes y servers son estructurados en clusters interconectados por una LAN troncal (backbone LAN). Cada cluster lo forman una cantidad de workstations interconectados por una LAN y un cluster server que no es ni más ni menos que un representante del VICE. La LAN se conecta a la troncal a través de un router.

La descomposición en clusters es lo que permite salvar el problema de la escala y por ello representa uno de los puntos a favor de AFS.

La heurística básica es descargar el trabajo de los servidores hacia los clientes. Siguiendo esta idea el mecanismo seleccionado para operaciones remotas sobre archivos se basa en cargarlo completamente en la cache del cliente. No obstante si los archivos son muy grandes (por ejemplo una base de datos) y no caben en la memoria del cliente se hace necesario adicionar mecanismos extra a AFS para manejar estas situaciones.

Otras características importantes son:

Movilidad de los clientes: Los clientes pueden acceder al espacio de compartido desde cualquier workstation.

Seguridad: El VICE define una clara división ya que los programas ejecutados en sus máquinas son programas no clientes. El paquete de comunicaciones provee funciones de seguridad en la transmisión y de autenticación. La información sobre el cliente y grupos es almacenada en las bases de datos protegidas las que se hallan replicadas en cada server.

Protección: AFS provee listas de acceso para los directorios y los clásicos bits de UNIX para protección de archivos.

Heterogeneidad: A través de la definición del VICE existe una clara interfase entre diversos hardware de workstations y sus sistemas operativos. Luego algunos archivos en el directorio */bin* local son links simbólicos a archivos ejecutables específicos de cada máquina y residen en el Vice.

25.12.3. - EL ESPACIO COMPARTIDO DE NOMBRES [10][11][S.O.]

Está constituido por unidades inusualmente pequeñas denominadas **volúmenes**, a los cuales generalmente se los asocia con los archivos de un cliente y permiten la identificación y localización unívocamente de cada uno de ellos.

Los volúmenes son 'unidos' similarmente al mecanismo de montaje de UNIX; con la diferencia que en este último solo una partición entera de disco puede ser montada y en cambio una parte del disco puede tener varios volúmenes en AFS.

El VICE es asociado a un identificador de bajo nivel: el **fid**. Cada entrada de directorio en AFS mapea un componente de path-name con un fid.

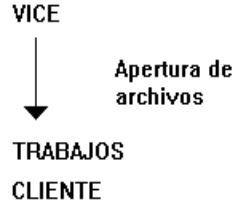
Un fid tiene 96 bits de longitud y consta de tres componentes de igual longitud.: **volume number**, **vnode number** y un **uniquifier**: El vnodo se usa como un índice en un arreglo que contiene los i-nodos de los archivos en cada volumen. El uniquifier permite la reutilización de los números de vnodo.

Los fid's tienen la propiedad de transparencia de ubicación, es decir al ser movidos de server a server, sus entradas siguen siendo válidas.

La información de ubicación se mantiene en una base de datos de ubicación de volúmenes replicada en cada server. De esta manera un cliente puede identificar la ubicación de cualquier volumen. Esta agregación de archivos en los volúmenes es la que hace posible el mantenimiento de las bases de datos en un tamaño manejable.

Para balancear la utilización de los servers y el espacio en disco, los volúmenes necesitan ser migrados. Esta operación se realiza en forma atómica de la siguiente forma:

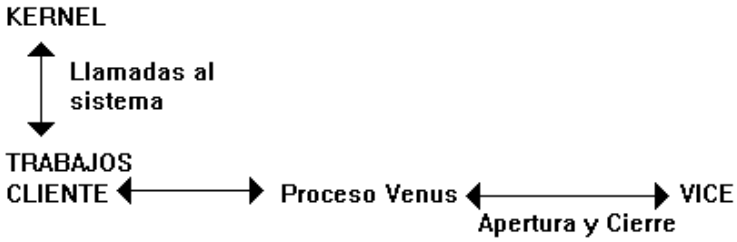
Cuando un volumen es colocado en la nueva ubicación, su server original actúa como forwarding y por lo tanto la base de datos de ubicación no necesita ser actualizada sincrónicamente. Mientras el volumen está siendo transferido, el server original aún recibe los updates los cuales son enviados más tarde al nuevo server. En algún punto el volumen es desconectado para que las nuevas modificaciones sean procesadas, luego el nuevo volumen es levantado en el nuevo server completando la operación.



25.12.4. - OPERACIONES DE ARCHIVOS Y SEMÁNTICAS [10][11][S.O.]

El principio fundamental en esta arquitectura es guardar por completo los archivos en las cachés locales a los clientes. De acuerdo a esto los clientes interactúan con los servidores VICE solo durante la apertura y cierre de archivos.

El sistema operativo en cada workstation intercepta las llamadas al sistema de archivos y las envía a un proceso a nivel del cliente en esa workstation. Este proceso llamado **Venus** hace el cache de los archivos desde el Vice cuando se los abre y guarda las modificaciones de vuelta al cerrarlos. Esta interacción Venus – Vice solo ocurre en la apertura y cierre de archivos, las lecturas o escrituras se realizan directamente en la cache y no interviene el Venus.



Venus asume que las entradas en caché son válidas a menos que se le informe lo contrario, y por ende, no contacta al Vice al abrir un archivo que ya figura en cache.

Para manejar esta situación se utiliza un mecanismo que se llama **callback**. Cuando el cliente guarda en caché un archivo o directory el servidor actualiza su información de estado. El servidor notifica al cliente antes de permitir una modificación de la información por otro cliente.

En ese caso se dice que el server elimina el callback del cliente viejo. Un cliente puede usar un archivo en cache solo si tiene callback. Si un cliente cierra un archivo luego de modificarlo todos los otros clientes que lo tienen en caché pierden sus callback y cuando vuelvan a abrir el archivo deben obtenerlo nuevamente desde el servidor.

En este mecanismo el server mantiene la información del callback propiamente dicha y el cliente la de la validación.

Cuando una workstation rebootea Venus considera las entradas en cache como posiblemente inválidas y genera un pedido de revalidación cada vez que se quiere acceder a ella por primera vez.

Si la cantidad de información de callbacks en el servidor es grande, entonces el server puede invalidar algunos.

El Venus cachea directorios y links simbólicos para path-name-translation. Cada componente en el path se obtiene (fetch) y se establece un callback para él. Las llamadas (look-up) se hacen localmente por el Venus sobre los directorios obtenidos (fetch) utilizando fids. Al final del camino del path todas las entradas del directorio y el file están en caché y cada uno tiene su callback. Las aperturas posteriores del archivo no implican tránsito en la red a menos que un callback haya sido invalidado.

La semántica utilizada es la de Sesión. Sin embargo existen excepciones, por ejemplo, las operaciones de archivos tales como cambios en la protección a nivel del directorio, las cuales son visibles inmediatamente en toda la red cuando la operación se completa.

25.12.5. - IMPLEMENTACIÓN [10][11][S.O.]

Los procesos de los clientes tienen una interfase con el kernel UNIX con el conjunto usual de llamadas al sistema. El kernel se modifica para detectar referencias al Vice y para enviar los requerimientos al nivel de procesos Venus.

Venus tiene un cache de mapeo que asocia volúmenes con servers. Si un volumen no se halla presente, entonces Venus contacta cualquier server con el que tiene conexión, requiere la información y la almacena en su propio cache. A menos que Venus ya tenga una conexión establecida con ese servidor, el establecimiento de la nueva conexión es necesario por cuestiones de autenticación y seguridad. Cuando un archivo es encontrado se crea una copia en el caché local. Venus retornará el handle al cliente, y éste abre la copia en caché.

El file system UNIX se utiliza como sistema de almacenamiento de bajo nivel para el cliente y para el server. El cache del cliente es un directorio local en el disco de la workstation. Las entradas en este directorio son los nombres de las entradas en la cache.

Venus y los procesos del server acceden a los archivos de UNIX directamente a través de los i-nodos para evitar la costosa traducción del i-nodo al path-name. Se agregan algunas llamadas al sistema debido a que la interfase del i-nodo no es visible a los procesos cliente.

Venus tiene dos caches, una para status y otra para data, a las cuales administra con el algoritmo LRU.

Cuando un archivo es removido de cache Venus notifica al server para que elimine el callback.

En el servidor hay un solo proceso a nivel cliente para brindar todos los servicios de archivos. Este proceso utiliza técnicas multithreading con threads no desalojables para atender varios pedidos concurrentemente.

Las RPC's se integran con estos threads. Existe una conexión RPC por cada cliente pero no hay un binding entre los threads y estas RPC's de manera que cualquier thread puede atender cualquier RPC.

El uso de un solo servidor multithreading permite el ocultamiento (caching) de las estructuras de datos necesarias para atender los requerimientos. Por otro lado la caída de este servidor paraliza toda su operatoria.