juanfrandm98
www.wuolah.com/student/juanfrandm98

2013

Tema-4.pdf

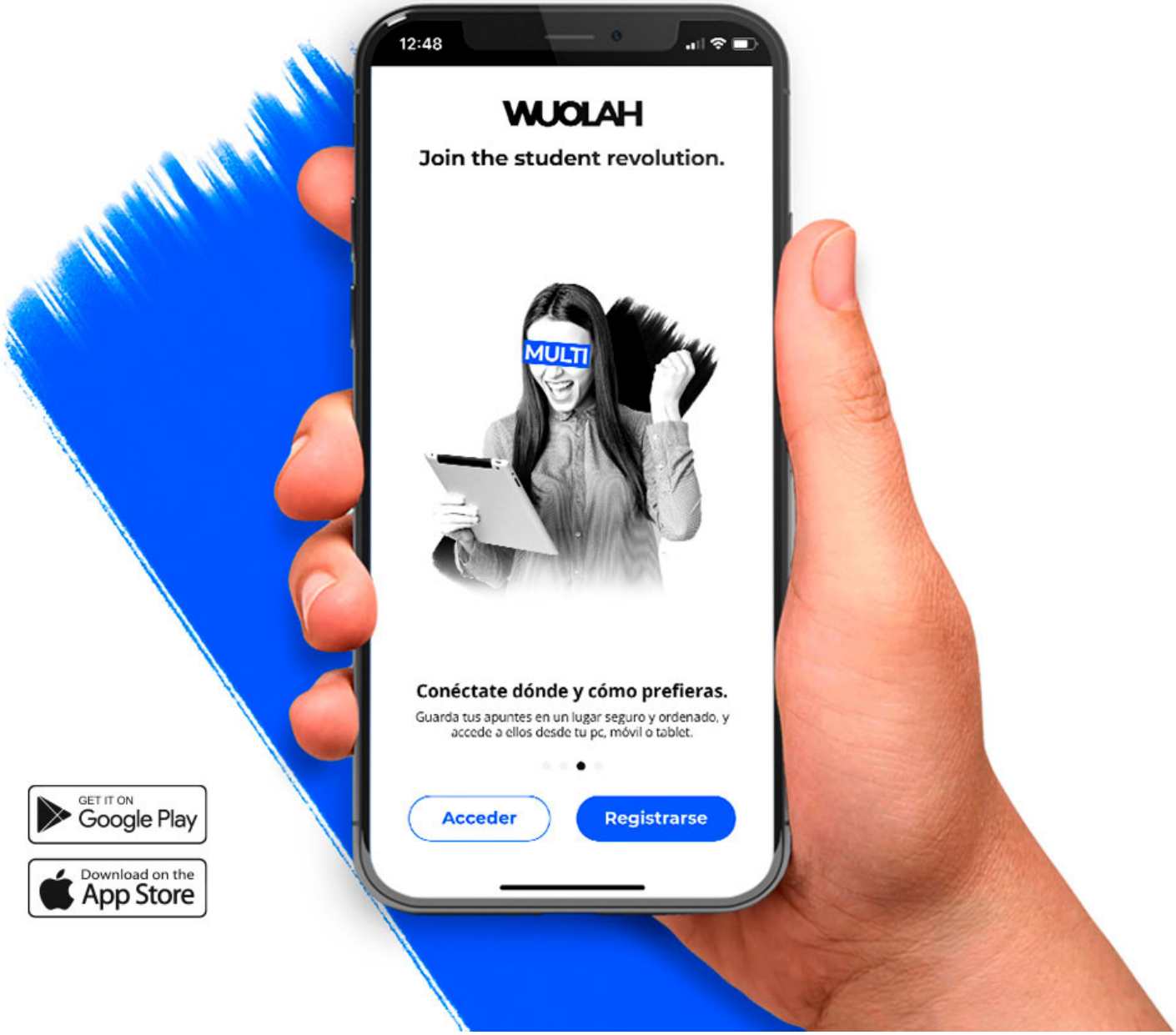
Apuntes de los Libros

2° Sistemas Operativos

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Exámenes, preguntas, apuntes.



GET IT ON Google Play

Download on the App Store

UNIVERSIDAD DE GRANADA

Sistemas Operativos

Tema 4. Gestión de Archivos

Juan Francisco Díaz Moreno
Curso 20/21

Descripción Básica de un Archivo

Concepto de Archivo

Independientemente del dispositivo de almacenamiento utilizado, el SO proporciona una vista lógica uniforme para el almacenamiento de información, el archivo. Los archivos son mapeados por el SO sobre los dispositivos físicos que son, usualmente, no volátiles, de modo que los contenidos persisten aunque se produzcan fallos de alimentación o reinicios del sistema.

Un archivo es una colección de información relacionada con un nombre que se graba en almacenamiento secundario. Para el usuario, es la unidad más pequeña de almacenamiento secundario lógico. Representan programas y datos, pudiendo ser numéricos, alfabéticos, alfanuméricos o binarios debido al formato libre de estos. En general, un archivo es una secuencia de bits, bytes, líneas o registros cuyo significado está definido por el creador y el usuario del archivo.

La información contenida dentro de un archivo es definida por su creador, pudiendo almacenarse todo tipo de información. Dependiendo del tipo de información que contienen, tendrán una estructura definida. Por ejemplo, un archivo de texto es una secuencia de caracteres organizada en líneas y páginas; mientras que un ejecutable es una serie de secciones de código que el cargador puede cargar en memoria y ejecutar.

Atributos de Archivo

Los atributos de un archivo varían de un sistema a otro, pero típicamente son:

- Nombre: permite que el archivo pueda ser referenciado.
- Identificador: etiqueta unívoca, usualmente un número, que identifica el archivo dentro del sistema de archivos, siendo básicamente una versión no legible por las personas de su nombre.
- Tipo: información necesaria para sistemas que soporten varios tipos de archivo.
- Ubicación: puntero a un dispositivo y a la ubicación dentro del mismo.
- Tamaño: tamaño actual en bytes, palabras o bloques; y posiblemente el tamaño máximo permitido.
- Protección: información de control de acceso.
- Fecha, hora e identificación del usuario: puede mantenerse para la creación, la última modificación y el último uso, siendo datos útiles para propósitos de protección, seguridad y monitorización del uso del archivo.

La información acerca de los archivos se almacena en la estructura de directorios, que también reside en almacenamiento secundario. Una entrada de directorio suele estar compuesta por el nombre de un archivo y su identificador. Este último permite localizar los demás atributos del archivo.

Puede ser necesario más de un kilobyte para almacenar los atributos de cada archivo, por lo que los directorios pueden ser del orden de megabytes. Los directorios tampoco son volátiles, por lo que residen en memoria secundaria y se deben cargar en memoria principal por partes, según sea necesario.

Operaciones con los Archivos

Un archivo es un tipo de datos abstracto que requiere definir las operaciones que pueden realizarse con él, siendo seis las básicas:

- Creación de un archivo: primero se debe encontrar espacio para crearlo dentro del sistema de archivos. Luego se debe incluir en el directorio una entrada para el nuevo archivo.
- Escritura en un archivo: se debe realizar una llamada al sistema que indique el nombre del archivo y la información a escribir. El sistema explora el directorio buscando el archivo por su nombre. El sistema debe mantener un puntero de escritura que haga referencia a la ubicación dentro del mismo en la que debe tener lugar la siguiente escritura, y actualizarlo cada vez que se escriba en él.
- Lectura de un archivo: de nuevo, una llamada al sistema especifica el nombre del archivo y dónde debe colocarse (en memoria) el siguiente bloque de archivo. Se explora el directorio en busca del archivo. El sistema mantiene otro puntero de lectura que hace referencia al lugar en el que tendrá lugar la siguiente lectura, actualizándose una vez que se haya completado. Dado a que los procesos suelen leer y escribir archivos, se puede almacenar la ubicación de la operación actual en un puntero de posición actual del archivo, diferente en cada proceso. Las operaciones de lectura y escritura utilizan el mismo puntero, ahorrando espacio y reduciendo la complejidad.
- Reposicionamiento dentro de un archivo: se explora el directorio para hallar la entrada y se reposiciona el puntero de posición actual dentro de un archivo, asignándole un nuevo valor. El reposicionamiento no tiene por qué implicar una operación de E/S. Esta operación se conoce también como búsqueda en el archivo.
- Borrado de un archivo: se explora el directorio en busca del archivo indicado, se libera todo su espacio para que pueda ser reutilizado por otros archivos, y se borra la entrada de directorio.
- Truncado de un archivo: se trata de eliminar el contenido de un archivo sin borrar sus atributos, ahorrando la destrucción y creación de dos archivos. El único atributo que se modifica es la longitud, que se pone a cero, y se libera el espacio que tuviera asignado.

Existen otras operaciones comunes, como la adición de información al final de un archivo o el renombrado de un archivo existente. Las seis operaciones básicas también pueden combinarse para constituir otras operaciones. Por ejemplo, la copia de un archivo puede verse como la creación de un nuevo archivo, la lectura del original y la escritura en el nuevo.

También se deben proporcionar operaciones para que los usuarios puedan consultar y modificar los atributos de los archivos.

La mayoría de estas operaciones implican buscar en un directorio la entrada asociada al archivo cuyo nombre se ha especificado. Para evitar constantes búsquedas, muchos sistemas requieren que se realice la llamada al sistema *open()* antes de utilizar por primera vez un archivo. Para ello, el SO mantiene una tabla de archivos abiertos. Cuando se realiza una operación con un archivo, este se especifica mediante un índice a esta tabla, evitando las operaciones de directorio. Cuando se realiza la llamada al sistema *close()*, se elimina la entrada correspondiente de la tabla.

Algunos sistemas abren implícitamente los archivos cuando se realiza la primera referencia a los mismos y los cierran automáticamente cuando se termina el trabajo o programa que los abrió. Sin embargo, la mayoría de sistemas requieren que el programador utilice la llamada al sistema *open()*, que toma un nombre de archivo y explora el directorio, copiando la entrada de directorio en la tabla de archivos abiertos correspondiente. *Open()* puede aceptar también el modo de acceso (creación, sólo lectura, lectura-escritura, sólo adición...), que se compara con los permisos definidos para el archivo para verificar la posibilidad de la apertura. La llamada devuelve un puntero que hace referencia a la entrada de la tabla de archivos abiertos que se utilizará en las posteriores operaciones de E/S, evitando sucesivas búsquedas y simplificando la interfaz.

La implementación de *open()* y *close()* es más complicada en entornos donde varios procesos pueden abrir un mismo archivo a la vez. Normalmente, el SO usará dos niveles de tablas internas: una tabla por cada proceso, que indica todos los archivos que ha abierto; y una tabla global del sistema.

Cada entrada de la tabla de archivos correspondiente a un proceso apunta, a su vez, a una tabla de archivos abiertos de carácter global, que contiene información independiente de los procesos,como la ubicación dentro del disco o su tamaño. Cuando un proceso abre un archivo, se incluye una entrada para dicho archivo en la tabla global. Si otro proceso lo abre, se añadirá simplemente una entrada a la tabla de ese proceso que apunta a la entrada de la tabla global. La tabla de archivos abiertos almacena también un contador de aperturas para cada archivo, de tal forma que la llamada *close()* lo reduce hasta que alcanza el valor cero, cuando la entrada se eliminará.

Por todo esto, con cada archivo abierto se asocian diferentes tipos de datos:

- Puntero de archivo: en sistemas que no incluyen un desplazamiento de archivo como parte de las llamadas *read()* y *write()*, el sistema registrará la ubicación de la última lectura-escritura mediante un puntero de posición actual dentro del archivo.
- Contador de aperturas del archivo: a medida que se cierran archivos, el sistema debe reutilizar las entradas de la tabla de archivos abiertos. Una entrada sólo podrá eliminarse cuando su contador de aperturas llegue a cero, es decir, cuando el último proceso que estaba utilizando su archivo lo cierra.
- Ubicación del archivo dentro del disco: esta información se almacena en memoria para evitar una operación de lectura de disco cada vez que se quieran modificar los datos dentro de un archivo.
- Derechos de acceso: se almacena en la tabla correspondiente a cada proceso para que el SO pueda autorizar o denegar las solicitudes de E/S.

Algunos SO proporcionan funciones para bloquear un archivo abierto o secciones del mismo, de tal forma que un proceso impida a otros que puedan acceder al mismo archivo. Esto resulta útil para archivos compartidos por varios procesos.

Un bloqueo compartido permite que varios procesos puedan adquirir dichos bloqueos concurrentemente (lectura); mientras que un bloqueo exclusivo implica que el bloqueo pueda obtenerse sólo por un proceso a la vez (escritura). No todos los sistemas proporcionan ambos tipos.

Los SO también pueden proporcionar mecanismos de bloqueo obligatorios o sugeridos. Si un bloqueo es obligatorio, se garantiza la integridad de los bloqueos; mientras que si el bloqueo es sugerido, es responsabilidad de los desarrolladores garantizar que se adquieran y liberen bloqueos adecuadamente. Windows utiliza bloqueos obligatorios, mientras que Linux utiliza los sugeridos.

El uso de los bloqueos de archivo requiere que se adopten las mismas precauciones para la sincronización normal de procesos.

Tipos de Archivo

Todo SO debe reconocer los tipos de los archivos para poder operar con ellos de forma razonable.

Una técnica común para implementar los tipos de archivo es incluirlo como parte del nombre del archivo, dividiéndolo en nombre y extensión. El sistema utiliza la extensión para conocer el tipo de archivo y determinar las operaciones que pueden realizarse con él.

Por ejemplo, los archivos .bat son archivos de procesamiento por lotes, que contienen en formato ASCII una serie de comandos dirigidos al SO.

El sistema UNIX utiliza un número mágico almacenado al principio de algunos archivos para indicar su tipo. No todos los archivos lo tienen, por lo que la funcionalidad del sistema no puede basarse exclusivamente en esta información. UNIX tampoco almacena el nombre del programa que creó el archivo, pero permite las sugerencias de las extensiones en los nombres de los archivos. No son obligatorias, pero ayudan a los usuarios a determinar el tipo de contenido del archivo y, dependiendo de la decisión del programador, pueden ser utilizadas o ignoradas por las aplicaciones.

Estructura de los Archivos

Los tipos de archivo también pueden usarse para indicar la estructura interna de los archivos. Algunos SO amplían esta idea y utilizan un conjunto de estructuras para manipular archivos con determinadas estructuras.

Que el SO soporte múltiples estructuras de archivo hace que su tamaño sea excesivo, ya que debe contar con el código necesario para soportar cada tipo de estructura. Además, todo archivo puede necesitar definirse como uno de los tipos soportados por el SO, y si una aplicación no está estructurada de la forma que el SO la soporta, pueden surgir graves problemas.

Por ejemplo, si un sistema soporta archivos de texto y ejecutables binarios, si un usuario quiere definir un archivo cifrado para evitar que el contenido sea leído por personas no autorizadas, es posible que ninguno de los tipos soportados resulte apropiado, puesto que está compuesto por bits aparentemente aleatorios. Como resultado, es posible que tenga que ignorar o malutilizar el mecanismo de tipos de archivo del SO, o renunciar a implementar el esquema de cifrado.

UNIX considera que cada archivo es una secuencia de bytes de 8 bits, sin que el SO realice ninguna interpretación de dichos bits. De esta forma obtiene una máxima flexibilidad a costa de un escaso soporte: cada programa de aplicación debe incluir su propio código para interpretar los archivos de entrada de acuerdo con la estructura apropiada.

Estructura Interna de los Archivos

Inicialmente, localizar un determinado desplazamiento dentro de un archivo puede ser complicado para el SO, por lo que los sistemas de disco suelen tener un tamaño de bloque bien definido determinado por el tamaño de un sector. Todas las operaciones de E/S se realizan en unidades de un bloque (registro físico) y todos los bloques tienen el mismo tamaño. Resulta poco probable que el tamaño del registro físico se corresponda exactamente con la longitud deseada de los registros lógicos, que pueden variar en longitud, por lo que se suele empaquetar varios registros lógicos dentro de los bloques físicos.

UNIX define todos los archivos como simples flujos de bytes. Cada byte es direccionable de manera individual, a partir de su desplazamiento con respecto al principio o al final del archivo. El tamaño de registro lógico es un byte, por lo que el sistema de archivos empaqueta y desempaqueta automáticamente los bytes en los bloques físicos de disco según sea necesario.

El tamaño del registro lógico, el tamaño del bloque físico y la técnica de empaquetamiento determinan cuántos registros lógicos se almacenan en cada bloque físico. El empaquetamiento puede ser utilizado por la aplicación o por el SO.

Puesto que el espacio de discos se asigna siempre en bloques, generalmente se desperdicia una parte del último bloque de cada archivo, generando fragmentación interna. Cuanto más grande sea el tamaño del bloque, mayor será la fragmentación interna.

Métodos de Acceso

Puede accederse a la información contenida en los archivos de distintas formas. Algunos sistemas sólo proporcionan un método, mientras que otros proporcionan varios, y elegir el método adecuado para cada aplicación constituye uno de los principales problemas de diseño.

Acceso Secuencial

El método de acceso más simple es el acceso secuencial. La información del archivo se procesa por orden, registro tras registro. Es el modo de acceso de editores y compiladores.

Una operación de lectura lee la siguiente porción del archivo e incrementa un puntero de archivo, que controla la ubicación de E/S. La operación de escritura añade información al final del archivo y hace que el puntero avance al final de los datos recién escritos. Los archivos pueden reinicializarse para situar el puntero al principio de los mismos y, en algunos sistemas, un programa puede saltar hacia delante y hacia detrás *n* registros.

El acceso secuencial funciona tanto en los dispositivos de acceso secuencial como en los de acceso aleatorio.

Acceso Directo

Un archivo está compuesto por registros lógicos de longitud fija que permiten a los programas leer y escribir registros rápidamente. El método de acceso directo, o acceso relativo, se basa en un modelo de archivos que corresponde con los dispositivos de disco, que permiten el acceso aleatorio a cualquier bloque de un archivo.

Para el acceso directo, el archivo se considera como una secuencia numerada de bloques o registros, lo que supone una gran ventaja para el acceso inmediato a grandes cantidades de información, como es el caso de las bases de datos.

En el método de acceso directo, las operaciones de archivo incluyen un número de bloque como parámetro. Así, por ejemplo, tendremos la operación *leer n*, donde n es el número de bloque a leer, en lugar de leer el siguiente.

Otra alternativa es mantener el acceso secuencial pero añadir una operación para posicionar en un bloque determinado el puntero de acceso.

El número de bloque proporcionado por el usuario al SO es normalmente un número de bloque relativo, un índice referido al comienzo del archivo, siendo el primer bloque el 0 y el siguiente el 1, aún cuando sus direcciones absolutas sean 14703 y 3192, por ejemplo. Esto permite al SO

decidir dónde hay que colocar el archivo y evitar que el usuario acceda a porciones del sistema de archivos que no formen parte del archivo deseado.

Cuando se produce una solicitud relativa al registro *N* de un archivo, y suponiendo que tengamos una longitud de registro lógico igual a *N*, esta se transforma en una solicitud de E/S donde se piden *L* bytes comenzando en la ubicación L * (N) dentro del archivo, suponiendo que el primer registro sea *N* = 0. Dado que los registros lógicos son de tamaño fijo, resulta muy fácil leer, escribir o borrarlos.

Algunos sistemas de archivos soportan ambos métodos, mientras que otros sólo soportan uno de ellos. Algunos sistemas requieren que los archivos se definan como secuenciales o directos y a cada archivo sólo se permite el acceso de la forma coherente.

Otros Métodos de Acceso

Pueden construirse otros métodos de acceso por encima del acceso directo, pero requieren la construcción de un índice para el archivo que contiene punteros a los distintos bloques. Para encontrar un registro dentro de un archivo, se explora el índice y se usa el puntero para acceder al archivo directamente y hallar el registro deseado.

Esta estructura nos permite explorar un archivo de gran tamaño con un número relativamente bajo de operaciones de E/S, aunque el propio índice puede ser demasiado grande para almacenarlo en la memoria. Una solución consiste en crear un índice del archivo índice. El archivo índice principal contendrá punteros a los archivos de índice secundarios, que a su vez apuntarían a los propios elementos de datos.

Estructura de directorios

Estructura de Almacenamiento

Cualquier dispositivo de almacenamiento suficientemente grande puede utilizarse para albergar uno o varios sistemas de archivos. Cada una de estas partes se conoce como partición, franjas o minidiscos. Podemos crear un sistema de archivos en cada una de dichas particiones.

Las particiones pueden combinarse para formar volúmenes, que también pueden albergar sistemas de archivos. Cada volumen puede considerarse como si fuera un disco virtual y puede almacenar un SO.

Cada volumen que contenga un sistema de archivos debe conocer información acerca de los archivos almacenados, que se almacena como entrada en un directorio de dispositivo (conocido

simplemente como directorio) o tabla de contenidos del volumen. El directorio almacena información de todos los archivos almacenados en el volumen, como el nombre, la ubicación o el tamaño.

Introducción a los Directorios

El directorio puede considerarse como una tabla de símbolosque traduce los nombres de archivo a sus correspondientes entradas de directorio. Se necesita insertar entradas, borrarlas, buscarlas por su nombre y enumerar todas las entradas del directorio.

Para considerar una estructura de directorio, necesitaremos las siguientes operaciones:

- Búsqueda de un archivo: exploración de la estructura de directorio para encontrar una entrada correspondiente a un archivo concreto. También se necesita poder encontrar todos los archivos cuyos nombres correspondan con un patrón dado.
- Crear un archivo.
- Borrar un archivo.
- Listar un directorio: enumerar los archivos contenidos y el contenido de la entrada de directorio correspondiente a cada archivo de la lista.
- Renombrar un archivo: también implica que se modifique su posición dentro de la estructura del directorio.
- Recorrer el sistema de archivos: para conseguir una mayor fiabilidad, resulta conveniente guardar el contenido y la estructura de todo el sistema de archivos a intervalos regulares. Esto se suele hacer copiando todos los archivos en una cinta magnética, lo que supone una copia de seguridad por si se produce un fallo del sistema.

Directorio de un Único Nivel

La estructura de directorio más simple es el directorio de un único nivel. Todos los archivos están contenidos en el mismo directorio, lo que resulta fácil de mantener y comprender.

Sin embargo, esto tiene grandes limitaciones cuando el número de archivos se incrementa o cuando el sistema tiene más de un usuario. Controlar un número muy elevado de archivos es una tarea extremadamente compleja. Dos archivos no pueden tener el mismo nombre y se complica la tarea de hacer que los nombres de archivo sean unívocos.

Directorio en Dos Niveles

La solución estándar a los problemas multiusuario de los directorios de un único nivel es la creación de un directorio separado para cada usuario (*user file directory*, UFD). Cada uno tiene una

estructura similar, pero incluye los archivos de un único usuario, de tal forma que cuando el trabajo de un usuario se inicia o cuando se conecta al sistema, se explora el directorio maestro de archivo (*master file directory*, MFD), que está indexado por el nombre de usuario o el número de cuenta. Cada una de sus entradas apunta al UFD de dicho usuario.

Cuando un usuario hace referencia a un archivo concreto, sólo se explora su UFD, por lo que cada usuario puede tener archivos con el mismo nombre, siempre y cuando dentro de cada UFD sean unívocos. Para crear un archivo, el SO sólo explora su UFD para comprobar si ya existe un archivo con el mismo nombre. Es imposible borrar archivos de otros usuarios.

Los directorios de usuario también se pueden crear y borrar con un programa especial del sistema, con el nombre de usuario apropiado y la correspondiente información de cuenta. El programa crea un nuevo UFD y añade su entrada en el MFD. La ejecución de este programa puede estar restringida a los administradores.

Esta estructura sigue teniendo desventajas. El aislamiento entre usuarios es una ventaja cuando son independientes, pero puede ser una desventaja si quieren cooperar y poder acceder a archivos de otro. Si queremos permitir ese tipo de acceso, cada usuario debe tener la posibilidad de especificar un archivo que pertenezca al directorio de otro usuario.

Un directorio en dos niveles puede verse como una estructura de árbol, o de árbol invertido, de altura 2. La raíz es el MFD y sus descendientes directos son los UFD. Los descendientes de los UFD (las hojas) son los archivos. El nombre de ruta de un archivo se define como un nombre de usuario y un nombre de archivo. Si un usuario quiere acceder a su archivo *test*, puede referirse a él símplemente así, pero si quiere acceder al del *user2*, tiene que nombrarlo */user2/test*.

Se necesita una sintaxis adicional para especificar el volumen de un archivo, que depende del sistema. En MS-DOS, se especifica como *C:/user2/test*; en VMS, *u:[stt.jdesck]login.com*; y otros tratan el volumen como parte del nombre de directorio: */u/pbg/test*.

Los programas del sistema están generalmente definidos como archivos, siendo estos un caso especial. Cuando se proporciona el comando apropiado, el cargador carga estos archivos y los ejecuta. Muchos intérpretes tratan dicho comando como el nombre del archivo del programa, que se buscaría en el UFD. Una solución sería copiar los archivos del sistema dentro de cada UFD, lo que supone una cantidad enorme de espacio desperdiciado.

La solución estándar consiste en complicar el proceso de búsqueda definiendo un directorio de usuario especial para contener los archivos del sistema. Cada vez que se proporciona un nombre de archivo para cargarlo, el SO analiza primero el UFD local. Si encuentra el archivo, lo utiliza; pero si no lo encuentra, pasa a explorar el directorio de usuario especial. La secuencia de directorios que se exploran cuando se proporciona un nombre de archivo se denomina ruta de búsqueda, y puede emplearse para contener un número ilimitado de directorios que haya que explorar cada vez que se proporcione un nombre de comando, como ocurre en UNIX.

Directorios con Estructura de Árbol

La generalización natural del directorio de dos niveles es ampliar la estructura para que tenga una altura arbitraria que permita a los usuarios crear sus propios subdirectorios y organizar los archivos. Cada árbol tiene un directorio raíz y todos los archivos tienen un nombre de ruta distinto.

Cada directorio contiene un conjunto de archivos o subdirectorios, que son archivos tratados de forma especial. Un bit de entrada de directorio define si esa entrada es un archivo (0) o un subdirectorio (1).

Cada proceso tiene un directorio actual, que debe contener la mayor parte de los archivos que le interesen y será explorado cuando se haga referencia a un archivo. Si se necesita un archivo que no está en el directorio actual, el usuario tendrá que especificar el nombre de ruta o cambiar el directorio actual, para lo cual se utiliza una llamada al sistema que toma como parámetro el nombre del directorio deseado para redefinir el directorio actual.

El directorio actual inicial de una *shell* de inicio de sesión de un usuario se designa en el momento de comenzar el trabajo del usuario o cuando inicia sesión. El SO analiza el archivo de cuentas para localizar la entrada correspondiente a este usuario almacenada como un puntero o nombre a su directorio inicial. La *shell* será el padre de los procesos que lance, por lo que cada uno tendrá el directorio actual que tuviera la *shell* cuando lo creó.

Los nombres de ruta pueden ser de dos tipos. Un nombre de ruta absoluto comienza en la raíz y sigue una ruta descendente hasta el archivo que especifica, indicando los directorios que componen la ruta. Un nombre de ruta relativo define la ruta a partir del directorio actual.

Permitir al usuario definir sus propios subdirectorios hace que pueda imponer una estructura a sus archivos.

Una decisión importante en la política de directorios es qué hacer si se borra un directorio. Si está vacío, simplemente se borra la entrada correspondiente dentro del directorio que lo contuviera; pero si no lo está, hay dos soluciones:

- Algunos sistemas como MS-DOS no permiten borrar directorios a menos que estén vacíos. Si existen subdirectorios dentro, también tienen que estarlo. Esto supone una gran cantidad de trabajo.
- Otra alternativa adoptada por el comando *rm* de UNIX consiste en proporcionar una opción: cuando se solicita el borrado de un directorio, también se borra todo su contenido.

Ambas opciones son fáciles de implementar, aunque la segunda es más cómoda a la vez que peligrosa, ya que puede eliminarse toda la estructura de directorios con un único comando.

Con una estructura de árbol, podemos permitir que los usuarios accedan a los archivos de otros usuarios especificando su nombre de ruta, ya sea relativo o absoluto.

Los nombres de ruta de estos directorios pueden llegar a ser muy largos, por lo el SO Macintosh automatizaba la búsqueda para programas ejecutables, evitando tener que recordar sus rutas. Este sistema mantiene el archivo *Desktop File* que contiene los nombres y ubicaciones de todos los programas ejecutables que encuentra. Cuando se introduce un disco o se conecta a la red, el sistema recorre sus directorios en busca de ejecutables y almacena la información pertinente, lo que soporta el mecanismo de doble clic.

Directorios en un Grafo Acíclico

Un directorio compartido entre usuarios existe en dos o más lugares simultáneamente. Mientras que la estructura de árbol prohíbe la compartición de archivos y directorios, un grafo acíclico sí permite esa redundancia.

Un archivo (o directorio) compartido no es lo mismo que dos copias del archivo. Con dos copias, si una se modifica, los cambios no se reflejarán en la otra. Con un archivo compartido sólo existe “un archivo real”. Si se crea un archivo en un directorio compartido aparecerá automáticamente en el de todos los usuarios.

El UFD de cada usuario del grupo que comparte un directorio lo contendrá como subdirectorio.

Los archivos y subdirectorios compartidos pueden implementarse de diversas formas. Una bastante común, que aparece en muchos sistemas UNIX, consiste en crear una nueva entrada de directorio denominada enlace (*link*), que es básicamente un puntero a otro archivo o subdirectorio. Cuando se referencia a un archivo, si la entrada de directorio está marcada como enlace, se incluye el nombre de archivo real dentro de la información del enlace. Para resolverlo, se utiliza el nombre de ruta para localizar el archivo real. El SO ignora los enlaces a la hora de recorrer los subárboles de directorio para preservar la estructura acíclica del sistema.

Otra técnica común consiste en duplicar toda la información acerca de esos archivos en todos los directorios los compartan, lo que dificulta el mantenimiento de la coherencia cuando se modifican archivos.

La estructura de grafo acíclico es más flexible que la estructura en árbol, pero también más compleja debido a varios problemas. Los archivos podrán tener ahora múltiples nombres de ruta absoluta, por lo que nombres de archivos distintos pueden referenciar a un mismo archivo. Al recorrer el sistema de archivos completo, no conviene recorrer estructuras compartidas más de una vez.

Otro problema es el borrado. Una opción es borrarlo cuando cualquier usuario lo haga, lo que puede dejar punteros colgantes al archivo inexistente, lo que puede ser más complicado si contienen direcciones reales de disco.

En un sistema en el que la compartición se implementa mediante enlaces simbólicos, la situación es más fácil de manejar. El borrado de un enlace no tiene por qué afectar al archivo original; pero si lo que se elimina es el propio archivo, se desasignará el espacio del archivo, dejando el enlace colgando. Podemos tener en cada archivo una lista de enlaces para reducir el coste de su eliminación consecutiva. En otro caso, puede ocurrir que tras borrar un archivo al que apuntaban enlaces simbólicos, se cree otro con el mismo nombre. En el caso de UNIX, cuando se borra un archivo se dejan los enlaces simbólicos, siendo responsabilidad del usuario darse cuenta de que el archivo original no existe.

Otra técnica de borrado consiste en preservar el archivo hasta que se borren todas las referencias al mismo. Para ello, debe existir un mecanismo para determinar que la que se ha borrado es la última referencia. Puede hacerse mediante una lista de referencias al archivo. Cuando se establece un enlace o una copia de la entrada de directorio, se añade una nueva entrada a la tabla de referencias. Cuando se borra, se elimina la correspondiente entrada. El archivo se borrará cuando se vacíe su lista de referencias.

El problema de esta técnica es el tamaño variable y posiblemente grande de la lista de referencias. No es necesario mantener la lista completa, si no que basta con mantener un contador que se incrementa y decrementa conforme se crean y borran referencias. UNIX utiliza esta técnica para los enlaces duros.

Directorio en Forma de Grafo General

El problema más grave que afecta a una estructura de grafo acíclico es garantizar que no existan ciclos. La creación de nuevos archivos y directorios preserva dicha naturaleza, que se rompe cuando aparecen enlaces.

La principal ventaja del grafo acíclico es la relativa simplicidad de los algoritmos requeridos para recorrerlo y para determinar cuándo no existen más referencias a un archivo. Hay que evitar recorrer varias veces las secciones compartidas por razones de rendimiento. Si se busca un archivo que no se encuentra la primera vez en un directorio compartido, explorarlo una segunda vez es un desperdicio. Un algoritmo mal diseñado podría provocar un bucle infinito explorando un ciclo indefinidamente, lo que podría evitarse limitando el número de directorios a los que se accede durante una búsqueda.

Un problema similar surge cuando tratamos de determinar si un archivo puede ser borrado. En un grafo acíclico, un valor de 0 en el contador de referencia significa que no quedan referencias, por lo que puede ser borrado. Sin embargo, si existen ciclos, el contador de referencias puede no ser 0 incluso aunque ya no sea posible hacer referencia a un directorio de archivo. Esta anomalía resulta de la posibilidad de auto-referencia en la estructura de directorios.

La recolección en memoria implica recorrer todo el sistema de archivos, marcando todos aquellos elementos a los que se pueda acceder. Después, una segunda pasada recopila todo aquello

que no esté marcado, incluyéndolo en una lista de espacio libre. Sin embargo, para un sistema de archivos basado en disco, esto consume muchísimo tiempo y no suele utilizarse.

La recolección de memoria solo es necesaria debido a la posible existencia de ciclos dentro del grafo. Por ello, es mucho más sencillo trabajar con estructuras acíclicas. Existen algoritmos que detectan la existencia de ciclos en los grafos, pero son muy costosos computacionalmente, especialmente si el grafo está almacenado en disco. Un algoritmo más simple consiste en ignorar los enlaces durante el recorrido de directorios, evitando ciclos sin necesidad de efectuar procesamiento adicional.

Compartición de archivos

Múltiples Usuarios

Cuando un sistema tiene múltiples usuarios, la compartición, la denominación y la protección de archivos cobran importancia. El sistema debe mediar en un directorio compartido para que un usuario pueda acceder a los archivos de otros de cierta manera o requerir que los usuarios concedan explícitamente acceso a sus archivos.

Para implementar la compartición y los mecanismos de protección, el sistema debe mantener más atributos de archivos y directorios que en sistemas monousuarios. La mayoría de los sistemas han evolucionado para usar los conceptos de propietario y grupo.

El propietario es el usuario que puede cambiar los atributos y conceder acceso y que dispone del máximo grado de control sobre un archivo. Es quien define las operaciones que los miembros del grupo y el resto de usuarios pueden ejecutar.

Los identificadores del propietario y del grupo de un archivo se almacenan junto con el resto de atributos. Cuando un usuario solicita realizar una operación sobre un archivo, se compara su ID con el atributo del propietario para determinar si lo es; de la misma forma que se comparan los identificadores de grupo. El resultado indicará qué permisos son aplicables y los aplicará a la operación solicitada para autorizarla o denegarla.

Sistemas de Archivos Remotos

El desarrollo de las redes ha hecho posible la comunicación entre computadores remotos, permitiendo la compartición de sus recursos y archivos.

El primer método de compartición de archivos se basaba en la transferencia mediante programas como *ftp*. El segundo método utiliza un sistema de archivos distribuido (*distributed file*

system, DFS) por el que directorios remotos son visibles en la máquina local. El tercer método, la *World Wide Web*, es una vuelta al primer método, necesitando un explorador para acceder a los archivos remotos y operaciones para transferirlos.

ftp se utiliza tanto para el acceso anónimo como para el acceso identificado. El acceso anónimo permite al usuario transferir archivos sin disponer de una cuenta en el sistema remoto. La *World Wide Web* utiliza este método casi exclusivamente. DFS requiere una integración mucho más estrecha entre la máquina que está accediendo a los archivos remotos y la máquina que los suministra, lo que añade un alto grado de complejidad.

El Modelo Cliente-Servidor

En este modelo, la máquina que contiene los archivos es el servidor, mientras que la máquina que trata de acceder a ellos es el cliente. Generalmente, el servidor declara que hay cierto recurso disponible y para qué usuarios lo está. Un servidor puede dar servicio a varios clientes y un cliente puede utilizar múltiples servidores.

El servidor especifica usualmente los archivos disponibles en el nivel de volumen o de directorio. Un cliente puede estar especificado por un nombre de red o por otro identificador, como una dirección IP, pero estas identidades pueden ser suplantadas o imitadas, de tal forma que un cliente no autorizado podría tener acceso al servidor. Otras soluciones más seguras incluyen una autenticación segura mediante claves cifradas. En cualquier caso, hay que garantizar la compatibilidad del cliente y del servidor y la seguridad de los intercambios de claves. Esto es complejo, por lo que se suelen usar métodos de autenticación no seguros.

En el caso de UNIX y en sus sistema de archivos en red (NFS, *network file system*), la autenticación tiene lugar mediante la información de conexión de red del cliente. Los identificadores de usuario en el cliente y en el servidor deben corresponderse, de otra manera el servidor no podrá determinar los derechos de acceso a los archivos y el acceso se denegará. El servidor debe confiar en que el cliente presenta el ID de usuario correcto. Este protocolo NFS permite relaciones muchos a muchos; y una máquina puede ser a la vez cliente y servidor.

Una vez que el sistema de archivos remoto ha sido montado, las solicitudes de operaciones con los archivos se envían al usuario a través de la red utilizando el protocolo DFS. Primero se envía la solicitud de apertura de un archivo junto con el ID del usuario solicitante. El servidor aplica las comprobaciones de acceso para comprobar si el usuario tiene acceso en el modo solicitado, concediendo o denegando la solicitud. Si se la concede, se devuelve un descriptor de archivo al cliente a partir del cual puede realizar lecturas, escrituras y otras operaciones con el archivo.

Sistemas de Información Distribuidos

Los sistemas distribuidos facilitan la gestión de los sistemas cliente-servidor proporcionando un acceso unificado a la información. El sistema de nombres de dominio (*domain name system*, DNS) proporciona un servicio de traducción de nombres de host a direcciones de red para todo Internet. Antes de la introducción de DNS, se intercambiaban mediante correo electrónico o *ftp*.

Otros sistemas de información distribuidos proporcionan un espacio de nombres de usuario/contraseña/ID de usuario/ID de grupo para una instalación distribuida. Los sistemas UNIX han empleado diversos métodos de distribución de información.

Sun Micro-system introdujo las páginas amarillas, que después se denominaron servicio de información de red o NIS) y la mayoría del sector las introdujo. Esto centralizaba el almacenamiento de nombres de usuario, nombres de host, información de impresoras y datos similares. Sin embargo, utilizaban métodos de autenticación inseguros, incluyendo envío sin cifrar de contraseñas.

En el caso del sistema común de archivos Internet (CIFS) de Microsoft, se utilizaba la información de red en conjunción con la autenticación de usuario para crear un inicio de sesión de red que el servidor emplea para decidir si puede o no permitir el acceso al sistema solicitado. Para que esta autenticación sea válida, los nombres de usuario deben coincidir con las distintas máquinas (como en NFS). Utilizaba dos estructuras distribuidas para proporcionar un único espacio de nombres para los usuarios.

El sector evoluciona hacia la utilización del protocolo ligero de acceso al directorio (LDAP) como mecanismo distribuido seguro de denominación. Una organización podría utilizar un directorio LDAP para almacenar la información de usuarios, dispositivos y la organización de la que disponga, lo que resultaría en un mecanismo de inicio de sesión unificado seguro para los usuarios, que introducirán una única vez su información de autenticación para acceder a todas las computadores de la organización.

Modos de Fallo

Los sistemas de archivos locales pueden fallar por diversos motivos, tanto por averías como por responsabilidad de usuarios y administradores. Esto puede provocar que se pierda una gran cantidad de datos.

Los sistemas de archivos remotos tienen aún más modos de fallo, debido a los problemas que pueden surgir en las interacciones de la red. Algunas redes de conexión no tienen suficiente protección, por lo que los canales entre los distintos terminales pueden interrumpirse. El sistema cliente tendría que actuar en consecuencia, por ejemplo, terminando las operaciones que estuviera realizando y retardando las interacciones con el servidor hasta que vuelva a estar disponible, evitando así la pérdida de datos.

Para implementar este tipo de recuperación de fallos, puede mantenerse información de estado tanto en el cliente como en el servidor, con la que podrían recuperar su estado al recuperarse del fallo.

NFS adopta un enfoque simple, con un DFS sin memoria de estado, presuponiendo que no se puede producir una solicitud de un cliente para un archivo a menos que el sistema de archivos haya sido montado y esté abierto previamente. El protocolo NFS transporta la información apropiada para localizar el archivo y realizar la operación solicitada. No controla qué clientes han montado los volúmenes exportados, asumiendo que si llega una solicitud, esta es legítima. De esta forma, este protocolo es resistente a los fallos y fácil de implementar, pero inseguro, ya que determinadas solicitudes ilegítimas podrían llegar a ser permitidas en el servidor. Esto se resuelve en el estándar NFS versión 4, que añade la memoria de estado para mejorar su seguridad, prestaciones y funcionalidad.

Semántica de Coherencia

La semántica de coherencia representa un criterio importante para evaluar un sistema de archivos que soporte compartición, ya que especifica cómo pueden acceder simultáneamente a un archivo múltiples usuarios. Especifica cuándo las modificaciones que un usuario realiza serán observables por los otros usuarios. Suele implementarse en el código del sistema de archivos.

En las siguientes secciones, se asume que cualquier acceso a un archivo está encerrado entre las operaciones *open()* y *close()*, lo que se conoce como sesión de archivo.

Semántica de UNIX

El sistema de archivos UNIX utiliza la siguiente semántica de coherencia:

- Las escrituras en un archivo abierto por parte de un usuario son visibles inmediatamente para los otros usuarios que lo hayan abierto.
- Un modo de compartición permite a los usuarios compartir el puntero de ubicación actual dentro del archivo, compartiendo por tanto sus movimientos.

En esta semántica, cada archivo está asociado con una única imagen física a la que se accede en forma de recurso exclusivo. La contienda por dicha imagen provoca retardos en los procesos.

Semántica de Sesión

El sistema de archivos Andrew utiliza la siguiente semántica de coherencia:

- Las escrituras en un archivo abierto por parte de un usuario no son visibles por el resto inmediatamente.

- Una vez que se cierra un archivo, los cambios realizados en él son visibles únicamente en las sesiones que den comienzo posteriormente, mientras que las instancias ya abiertas no reflejarán dichos cambios.

De esta forma, un archivo puede estar temporalmente asociado con varias imágenes al mismo tiempo, permitiendo que múltiples usuarios realicen accesos concurrentemente tanto de lectura como de escritura sin ningún retardo. No se impone prácticamente ninguna restricción a la planificación de los accesos.

Semántica de Archivos Compartidos Inmutables

En la semántica de archivos compartidos inmutables, una vez que un archivo es declarado como compartido por su creador, no puede ser modificado. Un archivo inmutable tiene dos propiedades clave: su nombre no puede reutilizarse y su contenido no puede ser modificado. La implementación de la semántica es bastante simple, ya que el mecanismo de compartición es de sólo lectura.

Protección

La información almacenada en un sistema informático debe ser protegida frente a daños físicos (fiabilidad) y frente a los accesos incorrectos (protección).

La fiabilidad suele proporcionarse mediante copias de seguridad, mientras que hay distintas formas de proporcionar protección. En un sistema monousuario, basta con extraer físicamente los disquetes y guardarlos bajo llave; pero en los sistemas multiusuario, se necesitan otros mecanismos.

Tipos de Acceso

Los sistemas que no permiten el acceso a los sistemas de otros usuarios no necesitan ninguna protección, bastaría prohibir el acceso. También se podría proporcionar un acceso libre sin protección. Ambas son extremas, lo que se necesita es un acceso controlado.

El acceso controlado limita los tipos de acceso a un archivo que puedan realizarse, permitiendo o denegando el acceso dependiendo de varios factores, como el tipo de acceso solicitado. Se pueden controlar varios tipos de operaciones:

- Lectura del archivo.
- Escritura o reescritura del archivo.
- Ejecución y carga del archivo en memoria.
- Adición de nueva información al final del archivo.

- Borrado del archivo y liberación del espacio para su reutilización.
- Listado del nombre y los atributos del archivo.

Otras operaciones, como el renombrado, e copiado y la edición, también pueden controlarse, pero en muchos sistemas se implementan combinando llamadas al sistema de menor nivel y la protección se proporciona en dicho nivel.

Control de Acceso

La técnica más común para proporcionar protección es que el acceso dependa de la identidad del usuario. El esquema más general es asociar cada archivo con una lista de control de acceso (ACL) que especifique los nombres de usuario y los tipos de acceso que se permiten a cada uno. Cuando un usuario solicita acceder a un archivo, el SO comprueba la lista asociada con este. Si dicho usuario está incluido para el tipo de acceso solicitado, se permite el acceso; en otro caso se deniega.

Esto permite la implementación de complejas metodologías de acceso, pero el tamaño de las listas de acceso puede ser enorme si se permite a muchos usuarios utilizar el mismo archivo. Esto tiene dos consecuencias poco deseables:

- Construir la lista puede ser una tarea tediosa y poco gratificante, especialmente si no se conoce de antemano la lista de usuarios del sistema.
- La entrada de directorio, que era de tamaño fijo, será de tamaño variable, lo que requiere mecanismos de gestión de espacio más complejos.

Estos problemas pueden resolverse utilizando una versión condensada de la lista de acceso, para lo que muchos sistemas clasifican a los usuarios en tres grupos:

- Propietario: usuario que creó el archivo.
- Grupo: conjunto de usuarios que comparten el archivo y necesitan un acceso similar.
- Universo: resto de usuarios.

La técnica reciente más común combina las listas de control de acceso con el esquema más general de la división de usuarios. Deben controlarse estrechamente los permisos y las listas de control de acceso. Por ejemplo, en UNIX, los grupos sólo pueden ser creados y modificados por el administrador, delegando el grado de control a la intervención humana. En VMS, el propietario puede crear y modificar esta lista de control de acceso.

Si utilizamos la clasificación más limitada de protección, sólo se necesitan tres campos para definir los mecanismos de protección. Cada campo es una colección de bit, y cada bit permite o deniega el acceso. Por ejemplo, UNIX define tres campos de tres bits cada uno: rwx para el propietario, para el grupo y para el resto de usuarios.

Una dificultad a la hora de combinar técnicas es la que surge por la interfaz de usuario, ya que deben poder determinar cuándo están activados los permisos ACL opcionales de un archivo.

Otra dificultad es la relativa a la asignación de precedencias cuando los permisos y las listas ACL entran en conflicto. Esto se soluciona con la regla general de que lo más específico debe tener prioridad sobre lo más genérico.

Otras Técnicas de Protección

Otra técnica para proporcionar protección consiste en asociar una contraseña para cada archivo. Si se seleccionan aleatoriamente y se cambian con frecuencia, puede ser muy efectivo, pero el usuario tiene que recordarlas y deberían ser distintas para todos los archivos, por lo que no es práctico. Algunos sistemas permiten asociar contraseñas con directorios, y otros tres contraseñas por minidisco (lectura, escritura y acceso multiescritura).

Algunos SO monousuarios proporcionan pocos mecanismos de protección de archivos. Algunos de ellos que se están conectando a la red se retroimplementan. Diseñar un mecanismo de seguridad suele ser más sencillo en un sistema nuevo que en otro existente, por lo que esas actualizaciones tienden a ser menos efectivas.

En una estructura de directorios multinivel se necesita proporcionar un mecanismo de protección de los directorios. Se debe poder controlar la protección y borrado de archivos, y también puede ser interesante controlar si un usuario puede determinar la existencia de un archivo dentro de un directorio. Si un nombre de ruta hace referencia a un archivo dentro de un directorio, debe permitirse al usuario acceder tanto al directorio como al archivo.

Estructura de un Sistema de Archivos

Los discos constituyen el principal tipo de almacenamiento secundario para mantener sistemas de archivos debido a dos características:

- El disco puede ser reescrito de manera directa, siendo posible leer, modificar y volver a escribir un bloque en el mismo lugar.
- Con un disco se puede acceder directamente a cualquier bloque de información que contenga, lo que hace simple el acceso a cualquier archivo de forma secuencial aleatoria. La conmutación de un archivo a otro sólo requiere mover los cabezales de lectura-escritura y esperar a que el disco acabe de rotar.

Las transferencias de E/S entre la memoria y el disco se realizan en bloques para mejorar la eficiencia. Cada bloque tiene uno o más sectores, que varían entre 32 y 4096 bytes (usualmente son de 512 bytes).

Para proporcionar un acceso eficiente y cómodo al disco, el SO impone uno o más sistemas de archivos. Estos acarrean dos problemas de diseño diferentes:

- Hay que definir qué aspecto debe tener el sistema de archivos, lo que implica definir un archivo y sus atributos, las operaciones permitidas sobre los archivos y la estructura de directorios utilizada.
- Hay que crear algoritmos y estructuras de datos que permitan mapear el sistema lógico de archivos sobre dispositivos físicos de almacenamiento secundario.

El sistema de archivos está compuesto generalmente de muchos niveles diferentes, de tal forma que cada nivel utiliza las funciones de los niveles inferiores para crear funcioens que serán utilizadas por los superiores.

El nivel más bajo, el control de E/S, está compuesto por controladores de dispositivo y rutinas de tratamiento de interrupción que se encargan de transferir la información entre la memoria principal y el sistema de disco. Un controlador de dispositivo es como un traductor cuya entrada está compuesta por comandos de alto nivel (como “extraer bloque 123”) y cuya salida consta de instrucciones de bajo nivel específicas del hardware, que serán utilizadas por la controladora hardware para establecer la interfaz del dispositivo con el resto del sistema.

El sistema básico de archivos sólo necesita enviar comandos genéricos al controlador del dispositivo apropiado para leer o escribir bloques físicos en el disco. Cada bloque físico se identifica mediante su dirección de disco numérica (por ejemplo, unidad 1, cilindro 73, pista 2, sector 10).

El módulo de organización de archivos tiene conocimiento acerca de los archivos y sus bloques lógicos, así como de sus bloques físicos. Conociendo el tipo de asignación de archivos utilizada y la ubicación del archivo, este módulo puede traducir las direcciones lógicas de bloque a direcciones físicas, que serán las que envíe al sistema básico de archivos para que realice las transferencias necesarias. Este módulo incluye también el gestor de espacio libre, que controla los bloques no asignados y proporciona dichos bloques al módulo de organización de archivos cuando los solicita.

Finalmente, el sistema lógico de archivos gestiona la información de metadatos, que incluyen toda la estructura del sistema de archivo salvo los propios datos. Este sistema gestiona la estructura de directorio para proporcionar al módulo de organización de archivos la información que este necesita, a partir de un nombre de archivo simbólico. Este nivel mantiene la estructura de los archivos mediante bloques de control de archivo (FCB), que contienen información acerca de los archivos, incluyendo su propietario, los permisos y la ubicación de su contenido. Este sistema también es responsable de las tareas de protección y seguridad.

Cuando se utiliza una estructura de niveles para la implementación de un sistema de archivos, se minimiza la duplicación de código. Cada sistema de archivos puede tener entonces su propio sistema lógico de archivos y su propio módulo de organización de archivos.

La mayoría de los SO soportan más de un sistema de archivos. UNIX utiliza el sistema de archivos UNIX (*UNIX file system*, UFS), que está basado en el sistema FFS (*Fast File System*). Aunque Linux soporta más de cuarenta sistemas de archivos distintos, el estándar se denomina sistema de archivos extendido, siendo sus versiones más comunes ext2 y ext3.

Métodos de Asignación de Espacio

El acceso directo a los discos proporciona una gran flexibilidad a la hora de implementar los archivos. El problema reside en el almacenamiento de múltiples archivos de forma que el espacio de disco se utilice eficazmente y que se pueda acceder a ellos de formar rápida.

Asignación Contigua

La asignación contigua requiere que cada archivo ocupe un conjunto de bloques contiguos en el disco de forma lineal. De esta forma, el acceso al bloque b+1 no requiere normalmente ningún movimiento del cabezal, minimizando el número de reposicionamientos del cabezal, que sólo se moverá para pasar del último sector de un cilindro al primero del siguiente.

La asignación contigua está definida por la asignación del primer bloque y por la longitud del archivo (en unidades de bloque). Si el archivo tiene *n* bloques y comienza en la ubicación *b*, ocupará los bloques *b* a *b+n-1*. La entrada de directorio de cada archivo indicará dichos parámetros.

Acceder a un archivo asignado de manera contigua es sencillo. Para el acceso secuencial, el sistema de archivos recuerda la dirección del último bloque referenciado y leerá el siguiente cuando sea necesario. Para el acceso directo al bloque *i* del archivo que comience en el bloque *b*, se puede acceder directamente al bloque *b+i*.

La asignación contigua también tiene problemas, como encontrar espacio para un nuevo archivo, que puede verse como un caso concreto del problema general de asignación dinámica de almacenamiento. Las estrategias más comunes para seleccionar un hueco disponible son las de primer ajuste y mejor ajuste. Son similares en lo que se refiere a utilización del espacio, pero la primera es generalmente más rápida.

Estos algoritmos sufren el problema de la fragmentación externa, que se basa en la descomposición del disco en pequeños fragmentos a medida que se asignan y borran archivos. Esto será especialmente significativo cuando el fragmento contiguo más grande sea insuficiente para satisfacer una solicitud. La gravedad del problema depende de la cantidad de espacio total y del tamaño medio de los archivos.

En antiguos sistemas para PC con asignación contigua en disquetes, el usuario debía ejecutar una rutina de reempaquetamiento para solucionar la fragmentación externa compactando el espacio libre. El disquete se liberaba completamente, creando un único espacio libre de gran tamaño en el que se iban copiando de nuevo los archivos de forma contigua. El problema de este mecanismo es el tiempo, agravándose a medida que aumenta el tamaño del dispositivo utilizado.

Algunos sistemas requieren que esta función se ejecute fuera de línea, teniendo el sistema de archivos desmontado. Durante este tiempo de detención, generalmente el sistema no puede

seguir operando de forma normal, por lo que los sistemas modernos necesitan otros mecanismos de desfragmentación que realicen la tarea en línea.

Otro problema es determinar cuánto espacio se necesita para un archivo, ya que en el momento de la creación será necesario determinar la cantidad total de espacio que se necesitará, lo que resulta difícil de estimar en la mayoría de los casos. Si se asigna demasiado poco espacio, el archivo no podrá ampliarse si el espacio a ambos lados de este está ya asignado. Para solucionarlo hay dos posibilidades: abortar el programa haciendo que el usuario asigne más espacio y lo vuelva a ejecutar, lo que suele ser muy costoso y obliga al usuario a sobreestimar el espacio necesario; o encontrar un hueco de mayor tamaño al que mover el archivo.

Incluso si la cantidad total del espacio necesario para un archivo se conoce de antemano, el mecanismo de preasignación puede ser poco eficiente. Es posible que un archivo crezca mucho pero a lo largo de mucho tiempo, por lo que la mayor parte de dicho tiempo se estaría desperdiciando espacio, logrando un alto grado de fragmentación interna.

Para minimizar estos problemas, algunos SO utilizan un esquema modificado de asignación contigua, mediante el que se asigna inicialmente un bloque contiguo de espacio y, posteriormente, si no es lo suficientemente grande, se añade otra área de espacio contiguo, denominado extensión. La ubicación de los bloques se registra entonces mediante una dirección y un número de bloques, junto con un enlace al primer bloque de la siguiente expansión. La fragmentación interna puede continuar siendo un problema si las extensiones son demasiado grandes; y la fragmentación externa también puede llegar serlo a medida que se asignen y desasignen extensiones de tamaño variable.

Asignación Enlazada

El mecanismo de asignación enlazada resuelve todos los problemas de la asignación contigua. Con ella, cada archivo es una lista enlazada de bloques que pueden estar dispersos por todo el disco. El directorio contiene un puntero al primer y al último bloque de cada archivo, y cada bloque contiene un puntero al siguiente.

Para crear un nuevo archivo, se crea una entrada en el directorio, que tiene un puntero al primer bloque de disco del archivo que se inicializa con el valor nulo (indica el fin de la lista), al igual que el campo de tamaño. Una escritura en el archivo hará que el sistema de gestión de espacio libre localice un bloque libre, la información se escribirá en dicho bloque y será enlazado al final del archivo. Para leer un archivo, se leen los bloques siguiendo los punteros de uno a otro.

De esta forma, no se produce fragmentación interna y puede utilizarse cualquier bloque de la lista de bloques libres para satisfacer una solicitud. Tampoco es necesario declarar el tamaño del archivo en el momento de su creación, y este puede crecer mientras haya bloques libres, eliminando la necesidad de compactación.

El principal problema de este mecanismo es que sólo se puede utilizar de manera efectiva para los archivos de acceso secuencial. Para encontrar el bloque i-ésimo de un archivo, hay que comenzar en el primero y seguir los punteros hasta llegar hasta el deseado. Cada acceso a un puntero requiere una lectura de disco, y algunos supondrán reposicionamiento de cabezales, por lo que el acceso directo resulta poco eficiente.

Otra desventaja es el espacio requerido para los punteros, que provocará que cada archivo requiera un poco más de espacio del que requeriría con otros mecanismos.

La solución usual es consolidar los bloques en grupos, denominados *clusters* y asignar *clusters* en lugar de bloques. Por ejemplo, un sistema de archivos puede definir un *cluster* como cuatro bloques y operar con el disco en unidades de *cluster*. Los punteros utilizarán un porcentaje menor de espacio. Esto permite que el mapeo entre bloques lógicos y físicos siga siendo simple, pero mejora la transferencia del disco al provocar menos reposicionamientos de cabezales, y reduce el espacio necesario para la asignación de bloques y la gestión de la lista de bloques libres. Sin embargo, incrementa el grado de fragmentación interna al desperdiciar más espacio cuando un *cluster* está parcialmente lleno. Los *clusters* también mejoran el tiempo de acceso a disco para muchos algoritmos, por lo que la mayoría de los sistemas de archivos los utilizan.

Otro problema de la asignación enlazada es la fiabilidad. Un error podría hacer que los punteros se perdieran, se dañaran o se malinterpretaran, de tal forma que un enlace podría apuntar a un bloque libre o al de otro archivo. Una solución parcial consiste en utilizar listas de elementos enlazadas, mientras que otra consiste en almacenar el nombre del archivo y el número de bloques relativo en cada bloque, desperdiciando todavía más espacio.

Una variante importante del mecanismo de asignación enlazada se basa en el uso de una tabla de asignación de archivos (*file allocation table*, FAT). Una sección del disco al principio de cada volumen se reserva para almacenar esta tabla, que tiene una entrada por cada bloque de disco y está indexada por el número de bloque. Cada entrada de directorio contiene el número del primer bloque del archivo. Los bloques no utilizados se indican mediante un valor de tabla igual a 0. Para asignar un nuevo bloque a un archivo, basta con encontrar la primera entrada con valor 0 y sustituir el valor anterior de fin de archivo por la dirección del nuevo bloque. Después, el 0 se sustituye por el valor de fin de archivo.

El esquema de asignación FAT puede provocar un número significativo de reposicionamiento de los cabezales de disco, a menos que la tabla FAT se almacene en caché. El cabezal del disco debe moverse primero al principio del volumen para leer la tabla FAT y encontrar la posición del bloque en cuestión antes de moverse a dicho bloque. En contraposición, mejora el acceso aleatorio, ya que el cabezal del disco puede encontrar la ubicación de cualquier bloque leyendo la tabla FAT.

Asignación Indexada

El método de la asignación enlazada resuelve los problemas de fragmentación externa y de declaración del tamaño que presentaba el método de la asignación contigua; pero, si no se utiliza una FAT, no puede soportar un acceso directo eficiente, ya que los punteros a bloques deben extraerse secuencialmente.

El mecanismo de asignación indexada resuelve este problema agrupando todos los punteros en una única ubicación: el bloque de índice. Cada archivo tiene su propio bloque de índice, que es una matriz de direcciones de bloques de disco. La entrada i-ésima del bloque de índice apunta al bloque i-ésimo del archivo. El directorio contiene la dirección del bloque de índice.

Cuando se crea un archivo, se asigna el valor nulo a todos los punteros del bloque de índice. Cuando se escribe por primera vez en el bloque i-ésimo, se solicita un bloque al gestor de espacio libre y su dirección se almacena en dicha entrada.

El mecanismo de asignación indexada soporta el acceso directo sin sufrir fragmentación externa, ya que puede utilizarse cualquier bloque para satisfacer una solicitud de más espacio. Sin embargo, sí sufre el problema del desperdicio de espacio, ya que necesita generalmente más espacio para almacenar los punteros que la asignación enlazada.

Esto nos lleva a preguntarnos cuál debe ser el tamaño del bloque de índice. Cada archivo debe tener uno, así que ese bloque deberá ser lo más pequeño posible, teniendo en cuenta que si es demasiado pequeño no podrá almacenar suficientes punteros para un archivo de gran tamaño. Los mecanismos que pueden utilizarse para ello son:

- Esquema enlazado: cada bloque índice ocupa normalmente un bloque de disco, por lo que puede leerse y escribirse directamente. Para archivos de gran tamaño se enlazan bloques de índice.
- Índice multinivel: consiste en utilizar un bloque de índice de primer nivel para apuntar a un conjunto de bloques de índice de segundo nivel, que apuntarán a los bloques. Para acceder a un bloque, el SO utiliza el índice de primer nivel para hallar un bloque de índice de segundo nivel, y luego busca en él el bloque de datos deseado. Esta técnica puede emplearse con múltiples niveles.
- Esquema combinado: la técnica utilizada por el sistema UFS consiste en mantener, por ejemplo, los primeros 15 punteros del bloque de índice en el nodo del archivo. Los primeros 12 de estos punteros hacen referencia a bloques directos, es decir, contienen la dirección de una serie de bloques que almacenan datos del archivo. Los siguientes tres punteros hacen referencia a bloques indirecto. El primero apunta a un bloque indirecto de un nivel, que es un bloque índice que no contiene datos sino las direcciones de otros bloques que almacenan datos. El segundo puntero hace referencia a un bloque doblemente indirecto, que contiene la dirección de un bloque que almacena las direcciones de otras series de bloques que contienen punteros a los propios bloques de datos. El último puntero contiene la dirección de un bloque triplemente indirecto.

Los esquemas de asignación indexados sufren de los mismos problemas de rendimiento que el mecanismo de asignación enlazada. Los bloques de índice pueden almacenarse en caché, pero los bloques de datos pueden estar distribuidos por todo el volumen.

Prestaciones

Los métodos de asignación vistos varían en lo que respecta a su eficiencia de almacenamiento y a los tiempos de acceso a los bloques de datos. Ambos criterios son importantes para seleccionar el método, pero también tenemos que determinar cómo se utilizará el sistema: un sistema con acceso preferentemente secuencial no debe utilizar el mismo método que uno fundamentalmente aleatorio.

Para cualquier tipo de acceso, el mecanismo de asignación contigua sólo requiere un acceso para extraer un bloque de disco y el cálculo del bloque posterior es inmediato.

En el mecanismo de asignación enlazada también podemos mantener en memoria la dirección del siguiente bloque y leerlo directamente para el acceso secuencial, pero para el directo, el bloque i-ésimo puede requerir *i* lecturas de disco. Por ello, no debe utilizarse para aplicaciones que requieran acceso directo.

Algunos sistemas soportan los archivos de acceso directo utilizando un mecanismo de asignación contigua y emplean el mecanismo de asignación enlazada para el acceso secuencial. Para ello, debe declararse el tipo de acceso que va a realizarse en el momento de crear el archivo, teniendo en cuenta que el acceso estará limitado para el tipo establecido. Además, el SO tendrá que disponer de las estructuras de datos y los algoritmos apropiados para soportar ambos métodos de asignación. Para convertir un tipo de archivo en otro, habrá que crear uno nuevo, realizar una copia y eliminar el primero.

Otros sistemas combinan la asignación contigua con indexada, utilizando la primera para archivos pequeños (de hasta tres o cuatro bloques) y conmutando a un mecanismo indexado si el tamaño del archivo crece. Puesto que la mayoría de archivos son pequeños, la asignación contigua resulta eficiente y las prestaciones medias pueden ser bastante buenas.

En sistemas reales se utilizan muchas otras técnicas de optimización. Dada la disparidad entre la velocidad de la CPU y la del disco, resulta razonable añadir miles de instrucciones adicionales al SO para ahorrarse unos pocos movimientos de los cabezales del disco. Además, esta disparidad se va acrecentando con el tiempo, permitiendo un mayor número de instrucciones para dicha optimización.

Gestión del Espacio Libre

Puesto que el espacio en disco es limitado, es necesario reutilizar el espacio de los archivos borrados siempre que sea posible. Para controlar el espacio libre del disco, el sistema mantiene una lista de espacio libre, que indica todos los bloques de disco libres, aquellos que no están asignados a ningún archivo o directorio.

Para crear un archivo, se explora la lista de espacio libre hasta obtener la cantidad de espacio requerido y se asigna dicho espacio al nuevo archivo. A continuación, el espacio utilizado se elimina de la lista de espacio libre. Cuando se borra un archivo, su espacio de disco se añade a la lista de espacio libre.

Vector de Bits

Recientemente, la lista de espacio libre se implementa como un mapa o vector de bits. Cada bloque está representado por un bit, que se iguala a 1 si está libre o a 0 si está asignado.

La principal ventaja de este enfoque es su relativa simplicidad y la eficiencia a la hora de localizar el primer bloque libre o *n* bloques libres consecutivos. Muchas computadoras proporcionan instrucciones de manipulación de bits que pueden utilizarse de manera efectiva para ese propósito.

Sin embargo, los vectores de bits son ineficientes a menos que se mantenga el vector completo en memoria principal, lo que sólo es posible para discos de pequeño tamaño.

Lista Enlazada

Otra técnica consiste en enlazar todos los bloques de espacio libres, manteniendo un puntero al primer bloque libre en una ubicación especial del disco y almacenándolo en la caché. Ese primer bloque contendrá un puntero al siguiente bloque libre, y así sucesivamente.

Este esquema es poco eficiente, ya que para recorrer la lista hay que leer cada bloque, lo que requiere un tiempo sustancial de E/S. Afortunadamente, no suele ser frecuente tener que recorrer toda la lista, ya que el SO suele necesitar un único bloque cada vez.

El método FAT incorpora el control de los bloques libres dentro de una estructura de datos utilizada para el algoritmo de asignación. En este caso, no haría falta utilizar ningún método separado.

Agrupamiento

Una modificación de la técnica de la lista enlazada consiste en almacenar las direcciones de *n* bloques libres en el primer bloque libre. Los primeros *n-1* estarán realmente libres, el último contendrá las direcciones de otros *n* bloques libres. Esto permite que se encuentran rápidamente un gran número de bloques libres.

Recuento

Otra técnica consiste en aprovechar el hecho de que, generalmente, pueden asignarse o liberarse simultáneamente varios bloques contiguos, en especial cuando se utiliza un algoritmo de asignación contigua o mediante la agrupación en *cluster*. Así, en lugar de mantener una lista de *n* direcciones de bloques libres, se mantiene la dirección del primer bloque y el número *n* de bloques libres contiguos que siguen a este primero.

Aunque cada entrada requiere más espacio, la lista global será más corta, siempre y cuando dicho contador sea generalmente superior a 1.

Implementación de Sistemas de Archivos

Introducción

Se utilizan varias estructuras en disco y en memoria para implementar un sistema de archivos, que dependen del SO, aunque hay unos principios básicos.

En el disco, el sistema de archivos puede contener estructuras como:

- Un bloque de control de arranque (por cada volumen) puede contener la información necesaria para iniciar un SO. Suele ser el primer bloque del volumen y puede estar vacío si no contiene ningún SO. En UFS, se denomina bloque de inicio; en NFTS, sector de arranque de la partición.
- Un bloque de control de volumen (por cada volumen) contiene detalles acerca del volumen, tales como el número de bloques, el tamaño de estos, el número de bloques libres y los punteros a estos, así como un contador de bloques de información FCB libres y punteros FCB. En UFS, esto se denomina superbloque; en NFTS, esto se almacena en la tabla maestra de archivos.
- Se utiliza una estructura de directorios por cada sistema de archivos. En UFS, esto incluye los nombres de los archivos y los nombres de inodo asociados; en NFTS, esto se almacena en la tabla maestra de archivos.

- Un bloque FTB por cada archivo contiene numerosos detalles acerca del mismo, incluyendo los permisos, el propietario, el tamaño y la ubicación de sus bloques de datos. En FUS, esta estructura se denomina inodo. En NFTS, esto se almacena en la tabla maestra de archivos.

La información almacenada en memoria se utiliza tanto para la gestión de un sistema de archivos como para la mejora del rendimiento mediante mecanismos de caché. Los datos se cargan en el momento del montaje y se descargan cuando el dispositivo se desmonta. Las estructuras que pueden incluir son:

- Una tabla de montaje en memoria contiene información acerca de cada volumen montado.
- Una caché de la estructura de directorios en memoria almacena la información relativa a los directorios a los que se ha accedido recientemente.
- La tabla global de archivos abiertos contiene una copia del FCB de cada archivo abierto, además de otras informaciones.
- La tabla de archivos abiertos de cada proceso contiene un puntero a la entrada apropiada de la entrada global de archivos abiertos, así como otras informaciones.

Para crear un nuevo archivo, un programa de aplicación llama al sistema lógico de archivos, que conoce el formato de las estructuras de directorios, y que asigna un nuevo FCB. El sistema lee entonces el directorio apropiado en la memoria, lo actualiza con el nombre del archivo y el nuevo FCB y lo vuelve a escribir en disco.

Algunos SO, incluyendo UNIX, tratan a los directorios igual que a los archivos, salvo porque se tratará de un archivo cuyo campo de tipo indicará que es un directorio. Independientemente de esto, el sistema lógico de archivos puede llamar al módulo de organización de archivos para mapear las operaciones de E/S de directorio sobre los números de bloque del disco, que se pasarán a su vez al sistema básico de archivos del sistema de control de E/S.

Para utilizar un archivo para E/S, primero hay que abrirlo. La llamada *open()* pasa un nombre del archivo al sistema de archivos. Primero busca en la tabla global de archivos abiertos para ver si está siendo utilizado por otro proceso. Si lo está, se crea una entrada en la tabla de archivos abiertos del proceso que apunte a la tabla global de archivos abiertos existente. Este algoritmo puede ahorrar una gran cantidad de trabajo. Cuando un archivo está abierto, se busca en la estructura de directorio para encontrar el nombre de archivo indicado. Usualmente, se almacena parte de esta estructura en una caché de la memoria para acelerar las operaciones de directorio. Una vez encontrado el archivo, el FCB se copia en la tabla global de archivos abiertos existente en memoria. Esta tabla no sólo almacena el FCB, sino que también controla el número de procesos que ha abierto el archivo.

A continuación, se crea una entrada en la tabla de archivos abiertos del proceso, con un puntero a la entrada de la tabla global de archivos abiertos y algunos otros campos de información, como un puntero a la ubicación actual dentro del archivo (para *read()* o *write()*) y el modo de acceso en el que se ha abierto el archivo. La llamada *open()* devuelve un puntero a la entrada apropiada de la tabla de archivos abiertos del proceso; todas las operaciones de archivo se realizan a partir de ahí mediante dicho puntero. El nombre del archivo no puede formar parte de la tabla de archivos abiertos, ya que el sistema no lo utiliza para nada una vez que se ha localizado el FCB apropiado en el disco. Sin embargo, se puede almacenar en caché, para ahorrar tiempo en las subsiguientes

aperturas del mismo archivo. El nombre que se proporciona a esas entradas varía de unos sistemas a otros. En UNIX, se utiliza el término *file descriptor*, mientras que en Windows se prefiere *file handle*. En castellano, se usa el término descriptor de archivo. Hasta que se cierre el archivo, todas las operaciones con el mismo se llevan a cabo mediante el descriptor de archivo contenido en la tabla de archivos abiertos.

Cuando un proceso cierra el archivo, se elimina la entrada de la tabla del proceso y se decrementa el contador de aperturas en la tabla global. Cuando todos los usuarios que hayan abierto el archivo lo cierren, los metadatos actualizados se copian en la estructura de directorio residente en el disco y se elimina la entrada de la tabla global de archivos abiertos.

Algunos sistemas complican más este esquema utilizando el sistema de archivos como interfaz para otros aspectos del sistema, como la comunicación en red. Por ejemplo, en UFS, la tabla global de archivos abiertos almacena los inodos y otras informaciones para los archivos y directorios, pero también alberga información similar para las conexiones de red y para los dispositivos.

No deben infravalorarse los aspectos de almacenamiento en caché de las estructuras de los sistemas de archivos. La mayoría de ellos mantienen en la memoria toda la información acerca de un archivo abierto, salvo los propios bloques de datos. El sistema BSD UNIX resulta bastante típico en su uso de cachés para tratar de ahorrar operaciones de E/S de disco. Su tasa media de aciertos de caché del 85% muestra que merece la pena implementarlo.

Particiones y Montaje

La disposición de la información en un disco puede tener múltiples variables en función del SO. Un disco puede dividirse en particiones, o bien un volumen puede abarcar múltiples particiones en múltiples discos (tecnología RAID).

Cada partición puede ser una partición “en bruto” (sin formato), que no contenga ningún sistema de archivos, o una partición procesada, que sí lo contenga. Los discos sin formato se utilizan cuando no resulte apropiado emplear un sistemas de archivos, como el espacio de intercambio en UNIX o los discos de algunas bases de datos. Los discos sin formato también pueden almacenar la información que necesitan los sistemas de discos RAID, como los mapas de bits que indican qué bloques están ubicados en espejo y cuáles han sido modificados y necesitan ser duplicados.

La información de arranque puede almacenarse en una partición independiente, que tiene su propio formato, porque en ese momento no se han cargado los controladores de dispositivo para los sistemas de archivos, cuyos formatos no podrán interpretarse. En lugar de ello, la información de arranque es, usualmente, una serie secuencial de bloques que se cargan como una imagen binaria en la memoria. La ejecución de esa imagen comienza en la ubicación predefinida. Esta imagen de arranque puede contener más información, a parte de las instrucciones requeridas para iniciar el SO. Por ejemplo, en máquinas con arranque dual, se puede almacenar en el espacio de arranque un cargador que pueda comprender múltiples sistemas de archivos y múltiples SO que, una vez

cargado, puede iniciar uno de los SO disponibles. El disco puede tener distintas particiones, cada una de las cuales contendrá un tipo diferente de sistema de archivos y SO.

La partición raíz, que contiene el *kernel* del SO y a veces otros archivos del sistema, se monta en el momento del arranque. Otros volúmenes pueden montarse automáticamente en el arranque o pueden ser montados posteriormente, dependiendo del SO. Como parte de la operación de montaje, el SO verifica que el dispositivo contenga un sistema de archivos válido pidiendo al controlador del dispositivo que lea el directorio del dispositivo y verifique si éste tiene el formato esperado. Si el formato no es válido, será necesario comprobar y posiblemente corregir la coherencia de la partición, con o sin intervención del usuario. Finalmente, el SO registrará en su tabla de montaje de la memoria que se ha montado un sistema de archivos, indicando su tipo.

En UNIX, los sistemas de archivos pueden montarse en cualquier directorio. El montaje se implementa configurando un indicador dentro de la copia en memoria del inodo correspondiente a dicho directorio, que atestigua que lo es. Otro campo apunta a una entrada dentro de la tabla de montaje, que indica qué dispositivo está montado allí. La entrada de la tabla de montaje contiene un puntero al superbloque del sistema de archivos existente en dicho dispositivo. Este esquema permite que el SO recorra su estructura de directorios, conmutando transparentemente entre sistemas de archivos de diferentes tipos.

Sistemas de Archivos Virtuales

Los SO actuales soportan concurrentemente múltiples tipos de sistemas de archivos. Un método subóptimo de implementarlos consiste en escribir rutinas diferentes de acceso a directorios y a archivos para cada uno de los tipos existentes. La mayoría de SO, incluyendo UNIX, utilizan técnicas de orientación a objetos para simplificar, organizar y modularizar la implementación.

Se utilizan estructuras de datos y procedimientos para aislar la funcionalidad básica de llamadas al sistema de los detalles de implementación. Así, la implementación de sistemas de archivos está compuesta por tres niveles:

1. El primer nivel es la interfaz del sistema de archivos, basada en las llamadas *open()*, *read()*, *write()* y *close()* y en los descriptores de archivos.
2. El segundo nivel se denomina sistema de archivos virtual (VFS), que cumple dos funciones importantes:
    a. Separa las operaciones genéricas del sistema de archivos con respecto a su implementación, definiendo una interfaz VFS muy clara. Pueden coexistir diversas implementaciones de la interfaz VFS en la misma máquina, permitiendo el acceso transparente a diferentes tipos de sistemas de archivos montados de modo local.
    b. El VFS proporciona un mecanismo para representar de forma coherente los archivos a través de una red. Está basado en una estructura de representación de archivos denominada vnodo, que contiene un designador numérico para los archivos unívocamente identificables dentro de la red (los inodos de UNIX solo son unívocos dentro de un único sistema de archivos). Esta unicidad en el nivel de red es

obligatoria para poder soportar los sistemas de archivos de red. El *kernel* mantiene una estructura de vnodo para cada nodo activo (archivo o directorio).

Así, el VFS distingue entre los archivos locales y los remotos, y los locales se distinguen entre sí de acuerdo con el tipo de sistema de archivos.

3. El tercer nivel es el que implementa el tipo de sistema de archivos o el protocolo del sistema de archivos remoto.

En la arquitectura VFS de Linux, hay cuatro tipos de objetos principales:

- El objeto inodo, que representa un archivo individual.
- El objeto archivo, que representa un archivo abierto.
- El objeto superbloque, que representa un sistema de archivos completo.
- El objeto entrada de directorio, que representa una entrada de directorio individual.

Para cada uno de estos cuatro tipos de objeto, el VFS define un conjunto de operaciones que hay que implementar. Cada objeto contiene un puntero a una tabla de funciones, que indica las direcciones de las funciones reales que implementan las operaciones definidas por el objeto concreto. Por ejemplo, la definición completa del objeto archivo se especifica en la estructura *file_operations*, ubicada en el archivo /usr/include/linux/fs.h.

Así, el nivel de software VFS puede realizar una operación sobre uno de estos objetos invocando la función apropiada a partir de la tabla de funciones del objeto, sin tener que conocer de antemano exactamente el tipo de objeto con el que está tratando. El VFS no conoce, ni tampoco le importa, si un inodo representa un archivo de disco, un archivo de directorio o un archivo remoto. Por ejemplo, la función apropiada para la operación *read()* de dicho archivo se encontrará siempre en el mismo lugar dentro de su tabla de funciones, y el nivel de software VFS invocará dicha función sin preocuparse de cómo se lean en la práctica los datos.

Implementación de Directorios

La selección de los algoritmos de asignación de directorios y gestión de directorios afecta significativamente a la eficiencia, las prestaciones y la fiabilidad del sistema de archivos.

Lista Lineal

El método más simple para implementar un directorio consiste en utilizar una lista lineal de nombres de archivos, con punteros a los bloques de datos. Esto es simple de programar, pero requiere mucho tiempo de ejecución.

Para crear un archivo, primero se explora el directorio para asegurar que no haya ya ninguno con el mismo nombre. Después, se añade una nueva entrada al final del directorio. Para borrar un archivo, se explora el directorio en busca del archivo y se libera el espacio que tenía asignado.

Para reutilizar la entrada de directorio, se pueden hacer varias cosas: se puede marcar la entrada como no utilizada (con un nombre especial o un bit usado-libre en cada entrada), se puede insertar en una lista de entradas libres de directorio, o se puede copiar la última entrada del directorio en la ubicación que ha quedado libre reduciendo la longitud del directorio. También podría utilizarse una lista enlazada para reducir el tiempo de borrado de un archivo.

La principal desventaja es que para localizar un archivo se requiere realizar una búsqueda lineal, lo que es muy lento. Muchos SO implementan una caché de software para almacenar la información de directorio utilizada más recientemente. Cada acierto de caché evita la necesidad de volver a buscar en disco. Una lista ordenada permite efectuar una búsqueda ordinaria y reduce el tiempo de búsqueda, pero mantenerla ordenada puede complicar la creación y el borrado de archivos, ya que se puede necesitar mover una gran cantidad de información.

Tabla Hash

Con la utilización de tablas *hash*, las entradas de directorio se almacenan en una lista lineal, pero se utiliza una estructura de datos *hash*. La tabla *hash* toma un valor calculado a partir del nombre del archivo y devuelve un puntero a la ubicación de dicho nombre de archivo dentro de la lista lineal, reduciendo enormemente el tiempo de búsqueda.

La inserción y el borrado son también sencillas, aunque hay que tener en cuenta las colisiones, aquellas situaciones en las que dos nombres de archivo proporcionan al aplicar la función *hash* la misma ubicación.

Las principales dificultades asociadas con las tablas *hash* son que su tamaño generalmente es fijo y que la función *hash* depende de dicho tamaño. Un cambio en el tamaño implicaría volver a mapear los nombres de los archivos ya ubicados.

Alternativamente, se puede utilizar una función *hash* con desbordamiento encadenada. Cada entrada *hash* puede ser una lista enlazada en lugar de un valor individual, resolviendo las colisiones añadiendo una nueva entrada a esa lista enlazada. Esto puede ralentizar las búsquedas, ya añade el recorrido de la lista enlazada, pero siguen siendo mucho más rápidas que una búsqueda lineal a través de todo el directorio.

Recuperación

Comprobación de Coherencia

Parte de la información se almacena en la memoria principal o en caché para acelerar el acceso. Esta estará más actualizada que la correspondiente información en el disco, ya que las actualizaciones de caché no se escriben necesariamente en el disco al producirse.

Un fallo de la computadora podría hacer que el contenido de la caché, los búferes y las operaciones de E/S se perdieran, lo que implicaría la pérdida de los cambios registrados sólo en caché. Con frecuencia, se ejecuta un programa especial durante el reinicio para comprobar posibles incoherencias de este tipo y corregirlas.

El comprobador de coherencia (*fsck* en UNIX), compara los datos de la estructura de directorios con los bloques de datos del disco y trata de corregir todas las incoherencias que detecte. Los algoritmos de asignación y gestión del espacio libre dictan los tipos de problemas que el comprobador debe tratar de detectar y dictan también el grado de éxito que puede tener en esa tarea.

Para paliarlo, UNIX almacena en caché las entradas de directorio para las lecturas, pero todas las escrituras de datos que provoquen cambios en la asignación del espacio o en algún tipo de metadato se realizan síncronamente, antes de escribir los correspondientes bloques de datos. Sin embargo, también pueden aparecer problemas si se interrumpe la escritura síncrona.

Copia de Seguridad y Restauración

Para evitar pérdidas de información por fallos en los discos, se pueden utilizar programas del sistema para realizar copias de seguridad de los datos en otro dispositivo de almacenamiento. La restauración de una pérdida sería simplemente la recuperación de los datos a partir de dicha copia.

Para minimizar la cantidad de datos que haya que copiar, se puede utilizar la información contenida en la entrada de directorio de cada archivo. Por ejemplo, si el programa de copia de seguridad determina que la última copia de seguridad de un archivo es posterior a su última modificación, no sería necesario copiar el archivo. Así, un plan típico sería:

- Día 1: copiar en el soporte de copia de seguridad todos los archivos del disco, realizando una copia de seguridad completa.
- Día 2: copiar en otro soporte físico todos los archivos que se hayan modificado desde el Día 1, siendo una copia de seguridad incremental.
- Día N: copiar en otro soporte físico todos los archivos que se hayan modificado desde el día N - 1. Después, volver al Día 1.

Si se utilizan distintos soportes físicos, la restauración comenzaría con la copia de los archivos de la copia de seguridad completa; y continuaría con cada una de las incrementales.

Una ventaja de este sistema es que se puede restaurar cualquier archivo que haya sido borrado accidentalmente durante este ciclo, extrayendo el archivo borrado de la copia de seguridad del día anterior. La longitud del ciclo será un compromiso entre la capacidad de soportes físicos requeridos y el número de días pasados a partir de los cuales podemos realizar una restauración.

Para reducir el número de cintas que haya que leer al efectuar la restauración, una opción consiste en realizar una copia de seguridad completa y luego copiar cada día todos los archivos que hayan cambiado desde la última copia de seguridad completa. Así puede realizarse la restauración utilizando sólo la copia de seguridad incremental más reciente y la copia de seguridad completa.

Puede que un usuario detecte que un archivo concreto falla o está corrompido mucho después de que el daño se haya producido. Por ello, conviene planificar la realización de copias de seguridad completas de tiempo en tiempo, que se guardarán para siempre. Estas copias se deben almacenar en lugares alejados de las copias de seguridad normales para garantizar su protección frente a desastres físicos o la reutilización accidental de estos dispositivos.