Tema 4: Gestión de archivos

A. Interfaz de los Sistemas de Archivos

A.1. CONCEPTO DE ARCHIVO

Archivo: colección de información relacionada y almacenada en un dispositivo de almacenamiento secundario. Contiene un espacio de direcciones lógicas contiguas.

Su ESTRUCTURA INTERNA (lógica):

- *Secuencia de bytes*, el tipo de archivo determina su estructura.
- Secuencia de registros de longitud fija.
- Secuencia de registros de longitud variable.

Los TIPOS DE ARCHIVO son: regulares, directorios, de dispositivo (bloques y caracteres), FIFO, socket, enlace simbólico.
Las formas de acceso son: secuencial y aleatorio.

Atributos

(6)

- Nombre: Información para usuario de aplicación.
- Tipo: no es igual que los tipos de formatos.
- Localización: información sobre dónde está en el dispositivo de almacenamiento secundario.
- Tamaño: tamaño actual del archivo.
- Protección: controla las operaciones permitidas sobre el archivo (quién puede leer, escribir y ejecutar).
- Tiempos y fechas de identificación de usuarios y grupo: para protección, seguridad y monitorización.

Operaciones sobre archivos

- Gestión
  - Crear `open()`, `link()`, `symlink()`, `mknod()`, `mkfifo()`
  - Borrar `unlink()`
  - Renombrar `rename()`
  - Copiar `rename()`
  - Establecer y obtener atributos `stat()`, `chmod()`, `chown()`, `open()`, `read()`, `write()`

- Procesamiento
  - Abrir/cerrar `open()`, `close()`
  - Leer/escribir(modificar, insertar, borrar info) `read()`, `write()`
  - Cambiar el puntero de lectura/escritura `lseek()`

A.2 ESTRUCTURA DE DIRECTORIOS

Directorio: colección de nodos conteniendo información acerca de todos los archivos (organización de archivos).

Tanto la estructura del directorio como los archivos residen en almacenamiento secundario. Los archivos de tipo directorio contienen y mantienen la organización del sistema de archivos.

a) Requisitos de la organización lógica:

- EFICIENCIA: Localización rápida de un archivo en la estructura de directorios.
- DENOMINACIÓN adecuada a las necesidades de usuario:
  - Dos usuarios pueden tener el mismo nombre para diferentes archivos.
  - El mismo archivo puede tener varios nombres (enlaces hard/soft).
- AGRUPACIÓN, se agrupan los archivos según sus propiedades o lo que considere el usuario.

b) Diseño de la estructura

Dos enfoques:

- Árbol:
  - Necesidad de búsquedas eficientes.
  - Posibilita la agrupación de archivos.
  - Permite nombres de camino absoluto y relativo (directorio de trabajo actual).
- Grafo:
  - Incorpora la compartición de directorios y archivos (hay que meter medidas de seguridad).

A.3. PROTECCIÓN DE LOS ARCHIVOS

Consiste en proporcionar acceso controlado a los archivos, es decir, quién puede acceder y qué operaciones puede hacer. Hay varios tipos de acceso sobre archivos y directorios:

- Sobre archivos: leer, escribir, ejecutar, añadir, borrar.
- Sobre directorios: leer, escribir, crear nuevos, borrar, cambiar a un directorio.

a) Listas de acceso

La solución para la protección es hacer el acceso dependiente del UID. Se incorporan las **listas de acceso**.

Las listas de acceso de usuarios individuales tiene el problema del tamaño, se soluciona con clases de usuario:

- PROPIETARIO: UID del propietario.
- GRUPO: GID del grupo.
- PÚBLICO: los demás UID.

Otra alternativa es *asociar un password* con el archivo, aunque presenta varios problemas:

- Recordar todos.
- Si solo se asocia una contraseña al SA, entonces o se tiene el acceso total o no se tiene ningún acceso.

b) Semánticas de consistencia

Especifican cuándo las modificaciones de los datos de un archivo compartido se observan por otros usuarios.

- Semántica de UNIX: hay varios niveles de compartición, de todas formas, la escritura se observa tras finalizar la operación.
- Semánticas de sesión: la escritura no se observa hasta que se cierra sesión.
- Archivos inmutables: si se comparte, no se modifica.

A.4. FUNCIONALIDAD BÁSICA DEL SA

Funciones del Sistema de Archivos:

- **Conocimiento** de todos los archivos del sistema de archivos.
- **Controlar** la compartición y forzar la protección de archivos.
- **Asignación/liberación** de espacio en disco.
- **Traducir** las direcciones lógicas del archivo en direcciones físicas usando LBA(*Logical Block Addressing*), donde los usuarios especifican las partes que quieren leer/escribir en términos de direcciones lógicas relativas en el archivo.

B. Aspectos de diseño del Sistema de Archivos

B.1 ESTRUCTURA DEL SA

Un Sistema de archivos posee dos ámbitos de diseño diferentes:

- Definir cómo debe ver el usuario el sistema de archivos, es decir, qué abstracciones proporciona:
    - Definir un archivo y sus atributos.
    - Definir las operaciones permitidas sobre un archivo.
    - Definir la estructura de directorios.
- Definir los **algoritmos y estructuras de datos** que deben crearse para establecer la correspondencia entre el sistema de archivos lógico y los dispositivos físicos donde se almacenan.

Por eficiencia, el SO mantiene una tabla indexada (por descriptor de archivo), de archivos abiertos. Esta tabla contiene también información de sesión de apertura sobre un archivo. Además incorpora el *Bloque de control de archivo*, una estructura con información de archivo en uso.

B.2. MÉTODOS DE ASIGNACIÓN DE ESPACIO

a) Contiguo

Los datos de un archivo se almacenan en un conjunto de bloques contiguos en disco.

- Ventajas :)
    - Metadatos de localización (solamente se pasa el nº de primer bloque y el número de bloques asignados).
    - Es bueno tanto en el acceso secuencial como en el acceso directo.
- Desventajas :(
    - No se conoce inicialmente el tamaño que podrá requerir el archivo. Si se asigna un tamaño fijo se puede tener fragmentación externa.
    - La asignación dinámica de espacio a archivos agravará la fragmentación externa.
    - Los archivos no pueden crecer, a no ser que se realice compactación del espacio de disco.

Acceso y tratamiento de datos

Además, tiene que haber correspondencia de dirección lógica en el archivo a dirección física de bloque de disco:

```
Dir_lógica/tamaño_bloque -> Cociente(C), resto(R)
```

El acceso a los datos se hará mediante:

```
nº de bloque = nº primer bloque asignado +C
```

El desplazamiento dentro del bloque será `R`.

b) Enlazado

Los datos de un archivo se almacenan en bloques que no tienen por qué ser contiguos, se guardan en una lista enlazada. Los metadatos de localización: nº primer bloque.

- Ventajas :)
  - Evita fragmentación externa.
  - Resuelve el crecimiento dinámico, evita que haya que compactar.
- Desventajas :(
  - El acceso directo no es efectivo (el secuencial si).
  - El espacio desperdiciado por bloque. SOLUCIÓN: asignar grupos de bloques.
  - Problema de seguridad, si hay un bloque dañado se pierde la lista. SOLUCIÓN: lista doblemente enlazada.

La correspondencia de dirección lógica a física será:

```
Dir_logica/(tama_bloque – tamaño nº_sig_bloque) => cociente(C), resto(R)
```

El acceso de datos:

```
Nº de bloque = C-ésimo bloque de la lista.
```

```
Desplazamiento dentro de bloque = tamaño nº_sig_bloque+R
```

c) Enlazado (FAT)

Es una variación (*FIle Allocation Table*), que reserva una sección de disco en bloques consecutivos al comienzo de la partición para almacenar la FAT.
Esta tabla contiene una entrada por cada bloque del disco y está indexada por número de bloque de disco.

Localización de bloque, se lee en la FAT.

- Ventajas:
  - Optimización de acceso directo (si está en MP).
- Desventajas:
  - Pérdida de enlaces si hay un bloque dañado, se soluciona con dos copias de la FAT.

d) Indexado

En este método los números de bloque asignados a un archivo están en una localización concreta **bloque índice**.
El metadato de localización indica el número de bloque índice, cada archivo tiene su propio bloque índice.

PARA LEER el i-ésimo blque, buscamos el num. de bloque de la i-ésima entrada del bloque índice.

- Ventajas:
  - Buen acceso directo.
  - No produce fragmentación externa.
- Desventajas:
  - Se puede desperdiciar espacio en los bloques índices.
  - Tamaño del bloque índice.

Para solucionar el problema del tamaño del bloque índice aparecen varios enfoques:

- Bloques índice enlazados.
- Bloques índice multinivel.
  - PROBLEMA: acceso a disco necesario para recuperar la dirección del bloque para cada nivel de indexación.
  - SOLUCIÓN: mantener algunos bloques en MP.
- Esquema combinado.

B.3. GESTIÓN DE ESPACIO LIBRE

El sistema mantiene una lista de los bloques que están libres, **lista de bloques libres**.
En el caso de la FAT, no se necesita ningún método adicional.

a) Implementaciones de la LBL

- MAPA/VECTOR DE BITS:

  - Cada bloque se representa con un bit (0-libre; 1-ocupado).
  - Es fácil encontrar un bloque libre, o n bloques libres consecutivos.
  - Es fácil tener archivos en bloques continuos.
  - Es ineficiente si no está en MP.

- LISTA ENLAZADA:

  - Enlaza todos los bloques libres del disco, y guarda el número del primer bloque libre.
  - :) no derrocha espacio
  - :(es relativamente ineficiente, porque proporciona bloques y el acceso directo no es necesario.

- LISTA ENLAZADA CON AGRUPACIÓN:

  - Cada bloque de la lista almacena n-1 bloques libres, por tanto obtener muchos bloques

libres es rápido.

- CUENTA:
    - A cada entrada de la lista con agrupación se le añade el número de bloques libres consecutivos.

B.4. IMPLEMENTACIÓN DE DIRECTORIOS

Una **entrada de directorio contiene: (depende)**

- MS-DOS: nombre de archivo + atributos + nº primer bloque de datos (FAT).
- UNIX: nombre de archivo + referencia a los Atributos.

Además, cuando se abre un archivo (sesión de trabajo), se produce la siguiente secuencia:

- El SO busca en su directorio la entrada correspondiente.
- Extrae sus atributos y la localización de ss bloques de datos y los coloca en una tabla en MP.
- Cualquier referencia posterior usa la información de dicha tabla.

a) Implementación de la compartición de archivos(enlaces)

Hay dos tipos:

- Enlaces simbólicos **soft link:**
    - Se crea un nuevo archivo de tipo enlace que almacena el camino absoluto o relativo del archivo al que enlaza.
    - Se puede usar en entornos distribuidos.
    - Provoca gran número de accesos a disco.
- Enlaces duros **hard links:**
    - Se crea una nueva entrada en el directorio con un nuevo nombre y se copia la referencia de atributos.
    - PROBLEMA: Borrar un nombre de archivo no implica borrar los atributos. SOLUCIÓN: contador de enlaces (cuando llega a 0 borra atributos).

B.5. DISTRIBUCIÓN DEL SISTEMA DE ARCHIVOS

Los sistemas de archivos se almacenan en discos que se pueden dividir en una o más particiones.

Existen varios tipos de **formateo de disco:**

- FÍSICO: Pone los sectores (cabecera y código de corrección de errores) por pista.
- LÓGICO: escribe la info. que el SO necesita para conocer y mantener los contenidos del disco:
  - Directorio inicial vacío.
  - Lista de bloques libres.
  - Espacio para atributos, ...

Se implementa un bloque de arranque para incializar el sistema localizado por bootstrap, junto con métodos para detectar y manejar bloques dañados.

B.6 TOLERANCIA A FALLOS Y RECUPERACIÓN

Como los archivos y directorios están en MP y en disco, el sistema debe asegurar que un fallo no genere pérdida o inconsistencia de datos.

Se producen distintos enfoques:

- **Comprobador de consistencia:** compara los datos de la estructura de directorios con los bloques de datos en disco y trata cualquier inconsistencia.
  - Es más fácil en listas enlazadas que con bloques índices.
- **Copias de seguridad (backup)** realizado por programas del sistema, de los datos de disco a otros dispositivo; y de recuperación de archivos perdidos.

C. Implementación de la gestión de archivos en Linux

C.1. SISTEMA DE ARCHIVOS

El SO implementa al menos un SA estándar o nativo (`ext2, ext3, ext4`), pero debe dar soporte a otros SA distintos (`vfat, iso9660, ntfs`).

Para poder permitirlo, el kernel incluye una capa entre los procesos de usuario (o biblioteca estándar), y la implementación del SA particular, que se llama **Sistema de archivos Virtual o VFS**.

Esta capa permite proporcionar una interfaz uniforme para la gestión de archivos que abstrae las particularidades de cada SA. Para ello presenta un conjunto único de llamadas para trabajar con archivos y SA (la llamada `write()` , se encarga del flujo de operaciones/datos por las distintas partes del sistema).

Tipos de SA

- SA basados en disco, para almacenar archivos en medios no volátiles (ext2, ext3, ...).
- SA Virtuales, generados por el kernel: permiten la comunicación entre programas y usuarios (`/proc`).
    - No requieren espacio de almacenamiento en ningún dispositivo hardware (la información está en MP).
- SA de Red: permiten acceder a los datos mediante la red.

C.2 INODOS

Archivo común

Para un programa de usuario, un archivo se identifica mediante un *descriptor de archivo*, que se usa como índice en la tabla de descriptores, asocia el archivo con las operaciones relacionadas con él.

Cosas sobre descriptores:

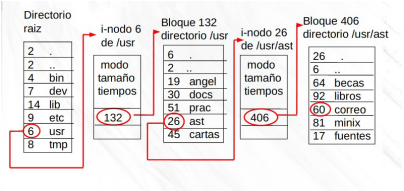
- El desciptor lo asigna el kernel al abrir el archivo, sólo val en el proceso.
- Dos procesos distintos pueden usar el mismo descriptor sin que tengan que apuntar al mismo archivo.

Un **inodo** es una estructura asociada a cada archivo y directorio, contiene sus metadatos. (También se pueden mantener el puntero de lectura/escritura y el modo de apertura en otra estructura).

I-nodo

Un **inodo** es la representación interna de un archivo: metadatos o atributos de un archivo almacenados en una estructura de datos.
Un archivo tiene asociado un único inodo, aunque puede tener varios nombres en el espacio de nombres (hard links).

- Si un proceso:
    - Crea un archivo: se asigna una nueva estructura de datos en disco.
    - Referencia un archivo por nombre: se analizan permisos y se lleva el inodo a memoria principal hasta que cierre.
- **CAMPOS DE UN INODO:**
    - IDENTIFICADOR: del propietario y del grupo asociado (`UID, GID`).
    - TIPO DE ARCHIVO(reg, dir, disp, cauce, link): si =0;



inodo libre.
    - PERMISOS.
    - TIEMPOS DE ACCESO: última modificación, último acceso,..
    - CONTADOR DE ENLACES
    - CAMPO DE LOCALIZACIÓN: mantiene los números de bloques de disco que soportan los datos del archivo.
    - TAMAÑO

C.3 VIRTUAL FILE SYSTEM

VFS sigue un diseño orientado a objetos con dos entidades, archivo y SA.
Se representan con una familia de estructuras de datos, existen 4 tipos de objetos primarios:

- Objeto `superblock`: SA montado.
- Objeto `inode`: un archivo de cualquier tipo.
- Objeto `dentry`: una entrada de un directorio.
- Objeto `file`: un archivo abierto, es una estructura del proceso local. Las anteriores son estructuras del sistem.

Además, todos estos objetos tienen un vector de operaciones `operations`.

Funciones que el kernel invoca sobre los o.p.:

- **super_operations**: métodos que el kernel puede invocar sobre un SA concreto (`write_inode(), sync_fs()`).
- **inode_operations**: métodos que el kernel puede invocar sobre un archivo concreto (`create(), link()`).
- **dentry_operations**, sobre una entrada de directorio: `d_compare()`, `d_delete()` -
- **file_operations()**, sobre un archivo abierto: `read(), write()`.

OTRAS ESTRUCTURAS:

- `file_system_type`: representa cada SA registrado, describe el SA y sus capacidades.

- `vfsmount`: representa cada punto de montaje, contiene info sobre él como sus localizaciones y flags.

- `fs_struct`, describen el SA asociado con el proceso.

- `files_struct`, archivos asociados con el proceso.

C.4. EXT2

Divide el disco duro en un conjunto de bloques del mismo tamaño, donde se almacenan los datos de archivos y de admin. Su elemento central es el **grupo de bloques**.