Irene Dominguez 2020/2021

TEMA 2: SISTEMAS DE ARCHIVOS

Contenido

VISIÓN GENERAL

Sistema de Archivos

La función del **Sistema de Archivos** es implementar archivos y directorios sobre la estructura plana de array de sectores de las unidades. Recibe peticiones de procesos de usuario, envía peticiones a gestores de dispositivos

Servicios (llamadas al sistema):

- **Sobre archivos completos**: crear, destruir, copiar, cambiar nombre, etc.
  Ejemplo: open(), close(), unlink()
- **Sobre contenido de archivos**: leer, escribir, añadir, modificar, truncar, etc.
  Ejemplo: read(), write()
- **Sobre sistema de archivos**: crear o borrar directorios, montar dispositivos, crear sistema de archivos, etc.
  Ejemplo: mkdir(), rmdir()
- **Otros**: mecanismos de protección, encriptado, compartición de archivos, control de concurrencia, etc.

Particiones de una unidad física

Una unidad física se puede dividir en unidades lógicas. Para gestionar las particiones usamos dos sistemas:

- **MBR Partition Table**: permite arrancar el sistema.
  Contienen una tabla en la que permite hasta 4 particiones primarias o 3 primarias + una extendida (esta partición se divide en otras subparticiones)
  Sólo se puede arrancar desde una partición primaria
  - Partición activa: partición primaria desde la que se arranca
  - Partición extendida: contenedor que permite un número arbitrario de subparticiones

  El límite de tamaño para una partición es de 2TB

Comentado [id1]: Cada proceso dispone de una tabla de descriptores de ficheros.

Fichero: archivo
Directorio: carpeta

Comentado [id2]: Los comandos y las llamadas al sistema tienen los mismos nombres

Comentado [id3]: **unlink**(): para borrar un fichero

Comentado [id4]: **mkdir**(): para crear un directorio
rmdir(): para borrar un directorio (sólo lo borra si está vacio)

Comentado [id5]: Partición: sistema de ficheros
Formatear: creación de ficheros
Primero se crea la partición y luego se formatea
El particionado se hace con herramientas como fdisk, cfdisk

- **GUID Partition Table** (GPT): permite hasta 128 particiones de 2ZiB cada una (2^{70} bytes)

Volumen: unidad lógica que contiene un sistema de archivos. Ocupan una unidad física completa o una partición.

Acceso:

- Sistemas tipo UNIX: se montan sobre un único árbol de directorios
- Sistemas tipo Windows: se le asignan letras de unidad

Estructura típica de un volumen:

- Cargador software del sistema operativo (opcional)
- Estructura de datos de gestión del espacio libre
- Estructura de datos de gestión del bloques defectuosos
- Estructuras de gestión del bloques asignados a archivos
- Directorio raíz

Directorio

Estructura de datos del sistema de archivos que contiene información sobre archivos contenidos en el mismo. La estructura de directorios normalmente es jerárquica. Todo volumen contiene al menos un directorio *raíz*. Suele contener información como el nombre, fecha de creación última modificación, último acceso, permisos de acceso, tamaño, dónde se localiza físicamente el archivo…

Gestión del espacio

El espacio de la unidad se asigna en grupos de sectores consecutivos (**bloques**, ***clusters***). La gestión y transferencias son más eficiente pues hay que manejar menos.

Para saber qué bloques están ocupados y cuáles están libre tenemos dos opciones:

- **Mapa de bits**
  Secuencia de bits en la que el bit i-ésimo representa el estado del bloque i-ésimo (1= ocupado, 0= libre). El mapa de bits se guarda en uno o varios bloques (o un archivo)
- **Integración con gestión espacio ocupado**

Gestión de bloques defectuosos

Las unidades siempre tienen sectores no utilizables y dichos sectores no se pueden asignar a archivos. Las soluciones habituales son:

- Asignarlos a un archivo del sistema que nunca se usará
- Crear un archivo que contiene la lista de bloques no utilizables
- Integración con la gestión del espacio ocupado

Comentado [id6]: Todo el sistema de ficheros depende de una única carpeta(carpeta raíz que viene representada por \, si se accede a cmd c\:)

En Windows hay tantos directorios raíces como volúmenes haya y cada volumen está identificado por una letra para identificarlo de manera única

Carpeta: contenedor de ficheros (almacena ficheros pero también contiene más carpetas, tenemos entonces un arbol de carpetas/ directorios)

Comentado [id7]: La unidad de almacenamiento masivo que es un dispositivo de E/S se divide en porciones de igual tamaño, **BLOQUES**

Un fichero está compuesto por N bloques.
Un bloque tiene N bytes (el tamaño de bloque se define en tiempo de formateo del volumen. Si no especificamos un tamaño de bloque se pone el tamaño por defecto 16 Kbytes, 32 Kbytes, 64 Kbytes, …)

Ejemplo: Si el tamaño de bloque es de 16Kbytes y mi fichero almacena 4bytes, en realidad ocupa 16Kbytes pero solo se usan 4bytes de ese bloque. Esto se llama **desperdicio interno**

Si tenemos un sistema de ficheros que va a almacenar ficheros que ocupan 4bytes, la mejor manera de **optimizar** el uso de la unidad de almacenamiento masivo sería definir un tamaño de bloque muy pequeño para reducir el desperdicio interno

Comentado [id8]: **Ejemplo**:
Mapa de bits: 0100001111 (0:ocupado, 1:libre) =>
el bloque 1 está ocupado
el bloque 2 libre
el bloque 3 ocupado

Para buscar espacio disponible en el dispositivo de almacenamiento masivo tengo que barrerlo hasta encontrar un hueco donde pueda meter tantos bloques como necesito. Los bloques de un sistema de ficheros no tienen por qué ser consecutivos

Comentado [id9]: Sucedía en las unidades de almacenamiento antiguas. Los bloques de almacenamiento son zonas de almacenamiento masivo que no funcionan. La forma de probar estos bloques es en tiempo de formateo: cuando está creando el fichero después prueba a escribir en cada bloque y leerlo. Si lo que lee es igual a lo que ha escrito entonces es fiable, pero si el valor no coincide, no es fiable y el sistema marca el bloque como defectuoso.

Gestión del espacio ocupado

Es necesario conocer qué archivos hay en el volumen y dado un archivo, qué bloques lo forman.

Archivo: secuencia de caracteres o bytes. Consiste en secuencia de bytes que se pueden leer byte a byte o bloque a bloque (de tamaño arbitrario). Si es un fichero direccionable, se permite acceso directo.

Comentado [id10]: Metadatos: Nombre, tamaño en bytes, fecha de creación, fecha de ultimo acceso, fecha de ultima modificación, propietario(usuario), grupo, permisos que aplican sobre ese fichero

UN SISTEMA DE ARCHIVOS SIMPLE: FAT

Sistema de archivo simple: FAT

Comentado [id11]: **FAT** (File Allocation Table): se llama así por la estructura de datos (una tabla) que se emplea para describir que bloques están libres, ocupados, defectuosos

Organización de un volumen FAT

- **Arranque**: en unidades particionables, contiene MBR mientras que en una unidad no particionable puede contener cargador software del SO
- **FS Information**: datos útiles para acelerar algunas operaciones como el número de bloques libres o el número del último bloque libre asignado
- **FAT**: tabla con una entrada por cada bloque de disco. Normalmente, replicada (dos copias). Posibles valores para cada entrada:
  - Bloque libre: FREE
  - Bloque defectuoso: BAD
  - Último bloque de un archivo: EOF

  Cualquier otro número indica el bloque asignado, y el número es el número del siguiente bloque. Las entradas 0 y 1 están reservadas para el SO.
- **Directorio en FAT**:
  - Directorio raíz en posición fija y tamaño fijo (2 bloques)
  - Subdirectorios pueden crecer (nunca decrecen)
  - Ficheros borrados: marca en primer carácter nombre
  - Nombres largos: varias entradas consecutivas

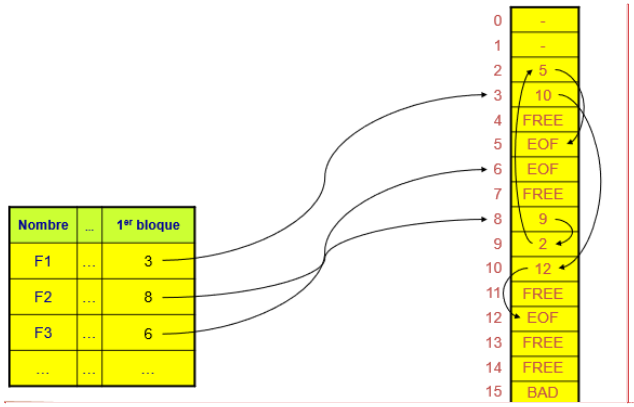
	Contenido	Tamaño
1	Nombre del archivo	8
2	Ampliación o *extensión*	3
3	Atributos	1
4	Reservado	10
5	Hora de última modificación	2
6	Fecha de última modificación	2
7	Número del primer bloque	2
8	Tamaño del archivo	4

Comentado [id12]: Hay diferentes variantes de FAT:
•**FAT16**: el tamaño de los bits que se emplean para una etiqueta que apunta a un bloque es de 16bytes. Con FAT16 => 2^16 bloques. Por **ejemplo**, si el bloque es de 16Kbytes: 2^16* 16 Kbytes

•FAT32 puedo gestionar 2^32 bloques. Por **ejemplo**, si el tamaño de bloque de 16Kbytes: 2^32 * 16 Kbytes = 2^32 * 16384 = 70368744177664 (tamaño max. del volumen que puedo gestionar con FAT)

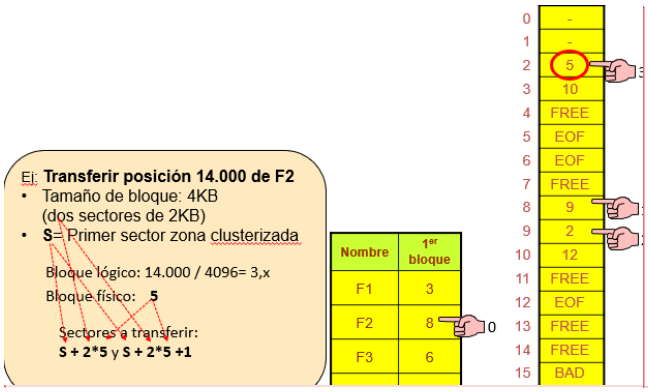
FAT tiene un límite: el tamaño de fichero. El tamaño máximo de un fichero en FAT es de 4 GBytes (esto es por la tabla de directorios)

Ejemplo:



Acceso directo a una posición de un archivo

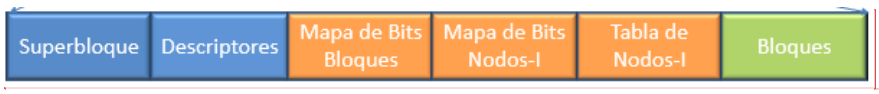
1. Determinar **bloque lógico** dentro del archivo
2. Iterar FAT para encontrar **bloque físico**(Primer bloque en entrada de directorio)
3. Determinar sector/sectores del bloque físico



UN SISTEMA DE ARCHIVOS SIMPLE: EXT2

Organización de un volumen ext2

Estructura de un grupo de bloques:



Comentado [id13]: Hay una única tabla FAT (tabla dcha.) por volumen y tenemos múltiples tablas de ficheros y directorios
Tenemos una tabla de ficheros y directorios por directorio, i.e., el directorio raíz tiene una tabla de entrada de ficheros y directorios (tabla izq.)
Una tabla de entrada puede apuntar a otra tabla. Al final lo que se construye es un árbol de tabla de entradas de directorios y ficheros y hay una única tabla FAT
Hay bloques del sistema de ficheros que guardan datos del usuario, tabla FAT, tabla de entrada de directorio.

El proceso es una lista enlazada a través de un array. Utilizaban arrays porque trabajaban con unidades de almacenamiento muy pequeñas. Otra posibilidad sería gestionar nuestras carpetas de almacenamiento masivo con bytes (una foto está desde el byte 20 al 40) pero no es algo humano

Sin la META información sería imposible referenciar nuestras carpetas.

Ejemplo:
F1: Bloques 3, 10 y 12
F2: Bloques 8, 9, 2 y 5
F3: Bloque 6
Bloque 15 defectuoso

Comentado [id14]: 1.Determinar el bloque lógico dentro del archivo que contiene la posición que se desea transferir. Esto se hace dividiendo dicha posición por el tamaño total del bloque, y el resultado obtenido, redondeado por defecto por aquello de que los informáticos empezamos a contar siempre desde cero, es el número lógico de bloque que buscamos.
2.Traducir ese número lógico de bloque al número físico de bloque en el volumen. Para ello, hemos de obtener la cadena de bloques que conforman el archivo. El bloque cero se encuentra en la entrada de directorio, y a partir de ahí se itera sobre la FAT para ir encontrando los demás bloques.
3.Una vez obtenemos el número físico de bloque iterando a través de la FAT, sólo quedaría calcular el número del sector o sectores que hay que ordenarle transferir al driver. Para ello, hemos de saber el número del primer sector del primer bloque del espacio clusterizado (sea S dicho número) y necesitamos saber cuántos sectores forman el bloque (en este caso, se supone que son dos). Los sectores a transferir son S + 2 * número de bloque físico, y su siguiente sector.

Comentado [id15]: Lo que va en azul tiene tamaño fijo, y lo que va en naranja tiene tamaño no determinado

Comentado [id16]: Cada bloque contiene un superbloque grupo, el grupo de bloques de mapa de bits, mapa de bits i-nodo, seguidos por los bloques de datos reales

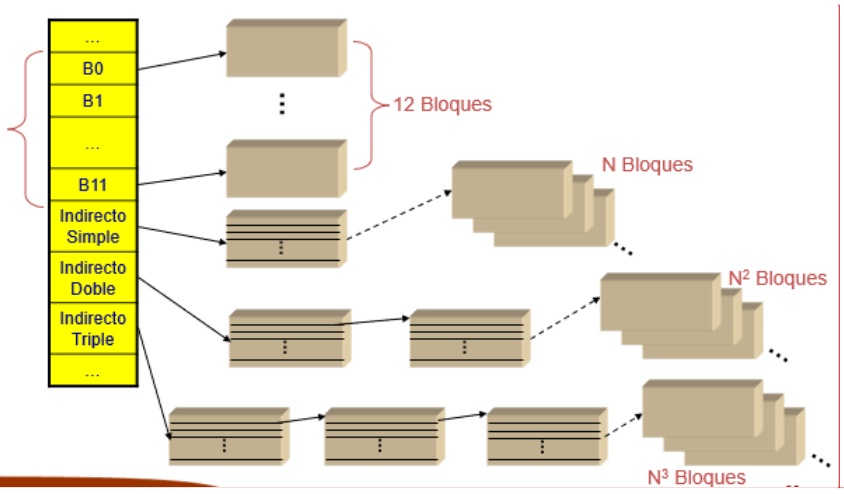
- **Superbloque**: contiene información importante que es crucial para el arranque del SO. Posee un tamaño fijo.
- **Descriptores**: contiene direcciones de comienzo del mapa de bits de bloques, mapa de bits de nodos-i, tabla de nodos-i, bloques del grupo
- **Nodo-i**: es un bloque de metadatos que va a guardar información relativa a un cierto fichero (tamaño en bytes, fecha de creación, última modificación, aunque no da información sobre su nombre) Además, el nodo indica el tipo de archivo (regular, directorio, archivo de dispositivo)

	Contenido	Tamaño
1	Modo (tipo de archivo y bits protección)	2
2	Nº de enlaces sobre el archivo	2
3	UID del propietario	2
4	GID del propietario	2
5	Tamaño en bytes del archivo	4
6	Fecha de creación	4
7	Fecha de último acceso	4
8	Fecha de última modificación	4
9	Números de los 12 primeros bloques	12x4
10	Número del bloque indirecto simple	4
11	Número del bloque indirecto doble	4
12	Número del bloque indirecto triple	4

- **Directorios en ext2**: archivo que contiene una lista de registros de tamaño variable. Cada registro es un nombre de archivo y posee: nodo-i asociado a un archivo, longitud del registro, longitud del nombre, tipo de archivo y nombre de archivo

	Contenido	Tamaño
1	Nodo-i asociado a archivo (0 si registro libre)	4
2	Longitud total de este registro	2
3	Longitud del nombre	1
4	Tipo de archivo	1
5	Nombre del archivo	<255

Gestión del espacio ocupado



Comentado [id17]: Por cada fichero que yo creo hay un i-nodo, el numero límite de i-no lo pone el sistema de ficheros.
En ext2 un fichero que ocupa 0bytes consume un i-nodo y el i-nodo es un bloque

Comentado [id18]: Sirve para poder navegar por el árbol de directorios, este árbol son una composición de tablas de entrada de directorios que tienen referencia a la carpeta padre (..) y su propia referencia(.)
Directorio actual de trabajo: cursor que indica en que tabla de directorio se encuentra el programa que est en ejecución, por eso el bloque (.) es util

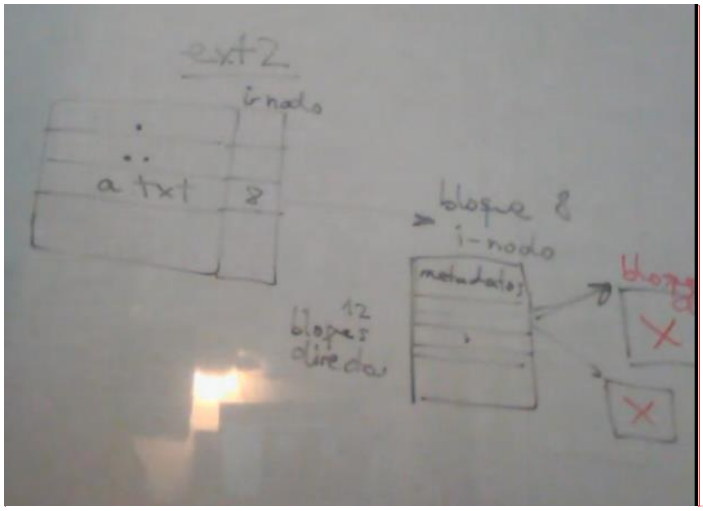
Comentado [id19]: Bloque directo: Con un i-nodo tengo 12 referencias a bloques directos.
nº.bloques/tam.CPU=nº. entradas a bloques directos
Ejemplo: 12 bloques directos y cada bloque es de 8 Kbytes => 96 Kbytes tamaño máximo de fichero

Indirecto simple: Se asigna un bloque de disco que actúa como una extensión de la tabla de 10 referencias que hay en el nodo-i.
Ejemplo: CPU 32bits => 4 bytes (32/8). Un bloque es de 8 Kbytes = 8192 bytes
8192 / 4 = 2048 referencias a bloques (estos bloques directos que van a tener ya datos)
2060 * 8192 = 16875520 bytes = 16 Mbytes (tamaño máximo de fichero)

Indirecto doble: si el archivo necesita más de N+12 bloques, se toma otro bloque de datos. Siendo N el número de referencias que caben en un bloque, se puede gestionar el espacio de archivos de hasta $N^2 + N + 12$ bloques.

Indirecto triple: siendo N el número de referencias que caben en un bloque, a partir de este bloque se pueden almacenar N^3 referencias a bloque más. Con esto ya se puede gestionar el espacio de archivos de hasta $N^3 + N^2 + N + 12$ bloques.

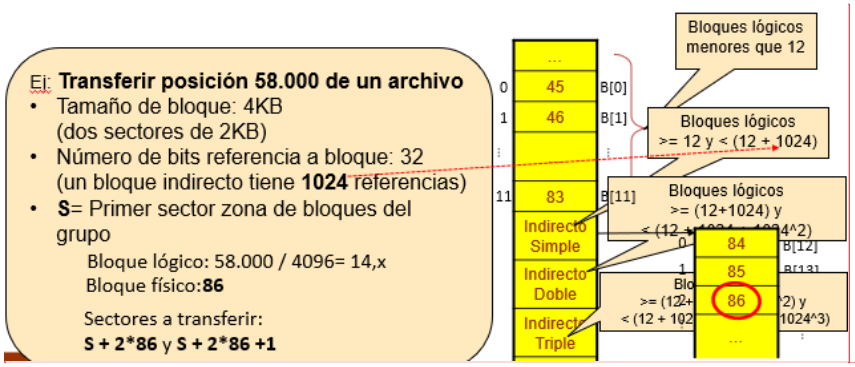
Ejemplo:



Comentado [id20]: PREGUNTA EXAMEN:
Si tengo un sistema de ficheros ext2 con solo 12 bloques directos en cada i-nodo y el tamaño de bloque es de 4KBytes, ¿cuál es el tamaño máximo de fichero? ¿Cuántos bytes puede tener un fichero con las 12 referencias de bloques directos?
4*12=48KBytes

Acceso directo a una posición de un archivo

1. Determinar **bloque lógico** dentro del archivo
2. Encontrar entrada correspondiente a dicho bloque lógico en nodo-i o bloque indirecto correspondiente (**bloque físico**)
3. Determinar sector/sectores del bloque físico



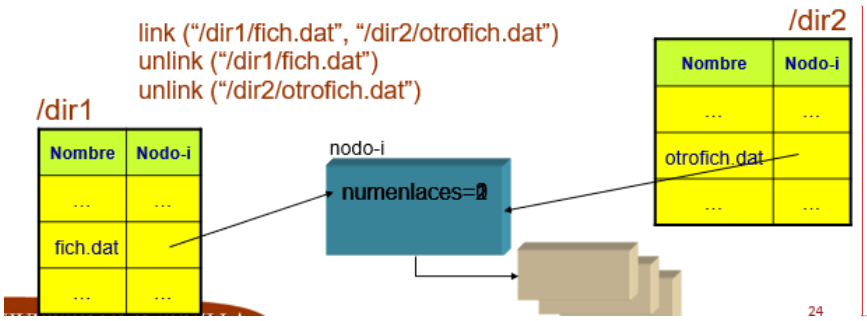
Comentado [id21]: 1.Determinar el bloque lógico dentro del archivo que contiene la posición que se desea transferir. Esto se hace dividiendo dicha posición por el tamaño total del bloque, y el resultado obtenido, redondeado.
2.Traducir ese número lógico de bloque al número físico de bloque en el volumen. Para ello, en función del número de bloque que buscamos y del número de referencias a bloque que cabe en un bloque indirecto (tamaño del bloque dividido entre tamaño de la referencia) determinar si el número de bloque estará en el nodo-i, o en un bloque indirecto. En caso de estar en un bloque indirecto doble o triple, hay que seguir haciendo el cálculo recursivamente. Una vez localizada la entrada correspondiente al bloque lógico, el contenido de dicha entrada es el número de bloque físico.
3.Calcular el número del sector que hay que ordenarle transferir al driver. Para ello, hemos de saber el número del primer sector del primer bloque de la agrupación que contiene al archivo (sea S dicho número) y necesitamos saber cuántos sectores forman el bloque (en este caso, se supone que son dos). Los sectores a transferir serían por tanto S + 2 * número de bloque físico, y su siguiente sector.

Enlace directo/duros

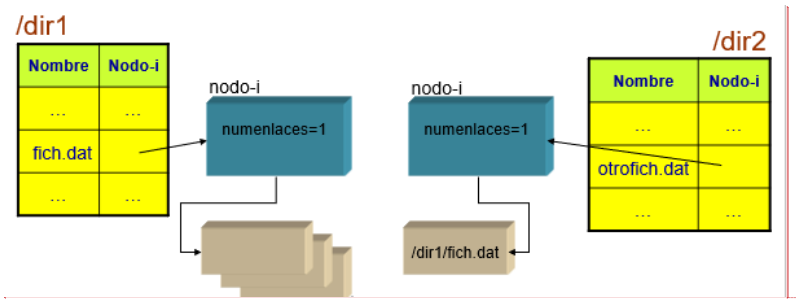
Un mismo archivo puede aparecer múltiples veces en la estructura de archivos

- Llamada **link**: crea un enlace sobre un archivo (se incrementa el contador de enlaces del nodo i. El contador se utiliza para saber cuándo puede borrarse el nodo-i)
- Llamada **unlink**: desenlaza un archivo

Comentado [id22]: Este contador de enlaces se utiliza para saber cuándo puede borrarse el nodo-i y los bloques del archivo (cuando no sea referido desde ninguna entrada de directorio)



Los **enlaces directos** presentan un problema cuando hay cuotas de archivos, un archivo enlazado debe residir en mismo dispositivo que directorio. Para resolver el problema, surge el **enlace simbólico**. Se crea un nuevo i-nodo y con él apuntamos a la dirección de la estructura de datos (de esta manera no tenemos que apuntar al i-nodo anterior)



- **Ventajas**: se puede enlazar cualquier archivo del sistema de archivos y no plantea problemas con las cuotas de disco.
- **Inconvenientes**: consume más recursos que enlace directo pues cada enlace simbólico requiere un nodo-i y un bloque de disco. Si archivo enlazado se borra surge incoherencia.

Los enlaces directos y simbólicos se complementan

RESERVA

La reserva (caché de disco, buffer de disco) permite mantener en memoria copia de los bloques que se están usando actualmente. El acceso a los archivos cumple principio de localidad temporal y espacial. Reservamos en memoria espacio para albergar:

- Copias de un número determinado de bloques de disco
- Identificación y estado de dichos bloques
- Estructuras de datos que nos permitan localizar fácilmente cada bloque

Comentado [id23]: Dos ficheros de la misma tabla pueden apuntar al mismo i-nodo.
Hasta que yo no borro todos los ficheros que apuntan a ese i-nodo no se libera espacio

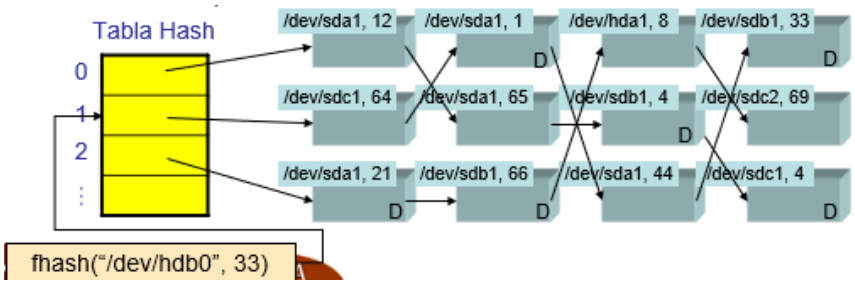
Comentado [id24]: Si hay cuotas de disco:
•Supongamos que usuario A crea un archivo: se le carga su espacio ocupado a su cuota
•Supongamos que A permite que B enlace su archivo: el espacio ocupado sigo cargándose al propietario, A
•A se queda sin cuota de disco: borra archivo
Archivo sigue existiendo (pues B lo tiene enlazado): el archivo ya no lo tiene A, y lo tiene B a su costa ☹

Comentado [id25]: Si hay cuotas de disco:
•Supongamos que usuario A crea un archivo: se le carga su espacio ocupado a su cuota
•Supongamos que A permite que B enlace su archivo: el espacio ocupado sigo cargándose al propietario, A
•A se queda sin cuota de disco: borra archivo
•Archivo sigue existiendo (pues B lo tiene enlazado): el archivo ya no lo tiene A, y lo tiene B a su costa ☹

Comentado [id26]: **Bloque de reserva:** se almacenan en memoria principal, mantienen bloques de datos de ficheros accedidos anteriormente para explotar el Ppio. de localidad temporal

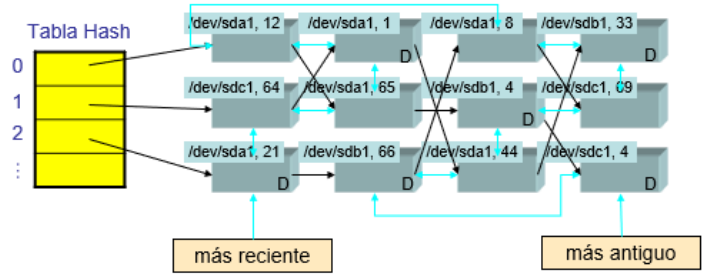
Comentado [id27]: **Principio de localidad temporal:** si accedo a un bloque de datos es muy probable que me vuelva a hacer falta en poco tiempo

Principio de localidad espacial: si accedo a un bloque N, es probable que acceda en poco tiempo al bloque N+1, N+2, …



Ante la necesidad de leer un bloque se comprueba si está en la reserva (buffer de disco). Si está, nos ahorramos lectura, pero si no está, se lee, y se carga en un bloque libre de la reserva. Si la reserva se llena: reemplazo LRU

Hay que añadir las estructuras de datos necesarias para el criterio de reemplazo.



Formas de usar la reserva

- **Reserva de escritura directa (write-through): cuando** se actualiza un bloque, se actualiza inmediatamente en disco
  - **Ventaja**: el disco está siempre actualizado. Útil en caso de caída del sistema
  - **Inconveniente**: la escritura no se beneficia de la reserva. En estos casos, el software de L/E en espacio de usuario suele crear buffers en espacio de procesos para amortiguar escrituras
- **Reserva de escritura diferida (write-back): c**uando se actualiza un bloque, sólo se actualiza su copia en memoria. El disco se actualiza al cerrar el archivo, al reemplazar el bloque en memoria, cuando proceso lo solicita (llamada al systema *sync()*) y al retirar el dispositivo o el medio.
  - **Ventaja**: la escritura se beneficia de la reserva.
  - **Inconvenientes**: no se pueden extraer discos arbitrariamente ni se puede apagar ordenador arbitrariamente. Desastre en caso de caída del sistema

Comentado [id28]: En caso de caída del sistema, además del riesgo de que se pierda información que los procesos creían haber escrito en los archivos, se da el riesgo de que el propio sistema de archivos pueda quedar corrupto al no haberse actualizado nodos-i, directorios, etc. Este último riesgo de puede paliar si el administrador de archivos no aplica la reserva de bloques sobre aquellos bloques que se sabe que contienen información del sistema de archivos.

RAID

La idea del RAID (Redundant Array of Independent Disks) se basa en que usando varios discos (baratos) en paralelo, podemos conseguir simular un disco con mejor rendimiento y fiabilidad.

Comentado [id29]: RAID nos permiten hacer dos cosas:
-Introducir redundancia en los discos: tener la información multiplicada en varios discos por si hay un fallo en algunas de las unidades de almacenamiento masivo
-Agregación: permite agregar múltiples discos y hacer creer al SO que hay un solo disco

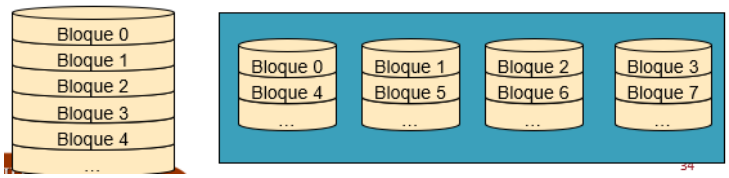
El RAID se implementa a nivel de controlador (harware RAID simula un único disco) y a nivel de gestor de dispositivo (sin hardware dedicado)

Se definen diferentes “niveles” y cada “nivel” tiene como objetivo la mejora de redimiento, capacidad y fiabilidad.

RAID 0

N discos simulan un disco de mayor capacidad. La información se divide en bandas (strips) de N sectores consecutivos. Cada sector de una banda se almacena en un disco **y se transfieren en paralelo:**

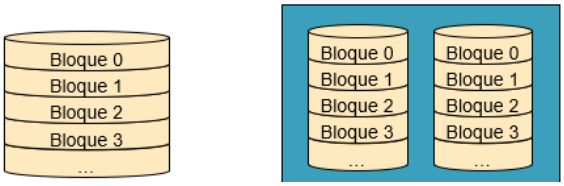
- Mejor tasa de transferencia
- Mayor tamaño
- ¡Peor fiabilidad!



RAID 1

N discos (frecuentemente 2) idénticos, en espejo. La información se escribe simultáneamente en todos los discos. Las lecturas se pueden realizar en paralelo:

- Mejor tasa de transferencia sólo en lectura
- Idéntico tamaño
- Mejor fiabilidad



RAID 4

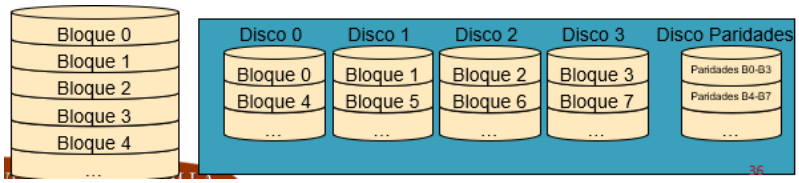
N+1 discos simulan un disco de mayor capacidad. La información está dividida en bandas, (como en RAID 0) + un disco de paridad. Se escribe en paralelo, pero:

- Disco de paridad se convierte en cuello de botella
- Mayor tamaño
- Mayor fiabilidad: si **se detecta** que un disco (**solo uno**) falla, se puede reconstruir la información a partir de la paridad

Comentado [id30]: Solo tiene agregación, no tiene redundancia. Si se pierde algún disco se pierde la información, se presenta al SO como un único disco, pero en verdad tendría por ej. 4 discos

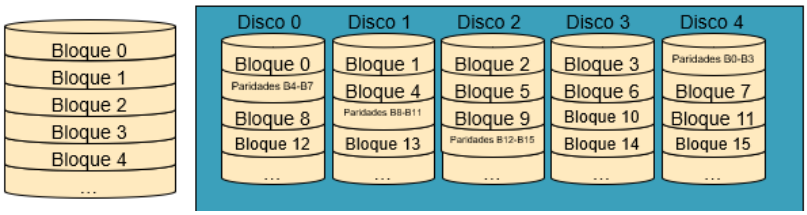
Comentado [id31]: La fiabilidad es peor que la de un solo disco, pues el array falla cuando uno de los discos falla

Comentado [id32]: Si el error no se detecta al leer en un disco, la de paridad sólo permite la detección del error, no su corrección.
Hasta que se sustituya el disco averiado, el sistema puede seguir funcionando, pero con rendimiento degradado, ya que para reconstruir la información hay que leer todos los discos



RAID 5

Igual al RAID 4, pero la paridad se reparte por turnos entre los demás discos. Su objetivo es que disco de paridad no se convierta en cuello de botella.



RAID 6

Igual al RAID 5, pero añade otro bloque más de "paridad". Utiliza un código Reed-Solomon de corrección de errores y eleva a dos unidades la tolerancia a fallos.

COPIA EN CASO DE ESCRITURA (COW)

Es una técnica aplicable sobre volúmenes o archivos. Cuando sobre un volumen/archivo se inicia una COW que queda "congelado". En ese momento, las escrituras se realizan sobre el archivo COW. Para leer, la información se busca en primer lugar en archivo de COW y, si no se encuentra, se busca en volumen/archivo original.

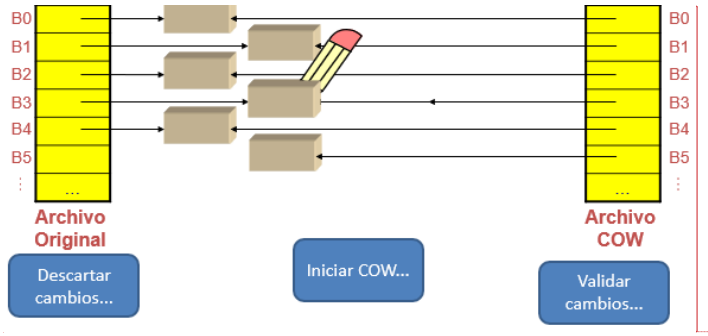
Al eliminarse un archivo COW, se descartan los cambios que se habían hecho en él.

Podemos fusionar el archivo COW con el volumen/archivo original haciendo que los cambios se hagan permanentes.

COW sobre archivos, a nivel de sistema de archivos

Se requiere que el espacio ocupado se gestione mediante tablas de bloques (ej: ext2)

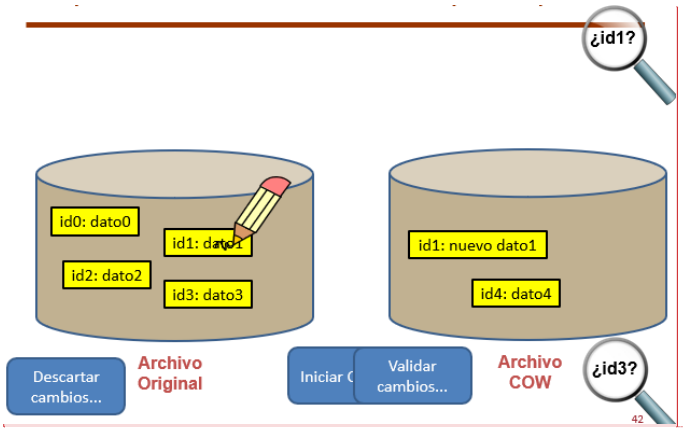
Comentado [id33]: Si tenemos dos procesos, uno que está leyendo de un fichero y otro que lo está actualizando.
Traemos ese fichero a los bloques de reserva, memoria principal:
P1 accede a fichero X
P2 accede a fichero X
los bloques del fichero X van a estar en memoria principal (bloques de reserva)
P1 comienza a actualizar bloques X
pero P2 está leyendo bloques de X
El SO cuando proceso P1 va a actualizar un bloque, lo que sucede es que hace una copia del bloque de reserva y actualiza esa copia para que P2 pueda seguir accediendo al bloque X sin observar cambios



Comentado [id34]: Al crear la copia en caso de escritura (click), se crea el archivo COW como un archivo compuesto por los mismos bloques que el archivo original.
¡Cuidado! Esto podría ser detectado como una incoherencia del sistema de archivos (bloques asignados más de una vez)
Supongamos que se intenta modificar un bloque (click). En dicho caso, se copia el bloque modificado (se crea bloque sombra) y se le asigna en exclusiva al archivo COW, manteniendo el original asignado al archivo original
Si se añade información (click): el nuevo bloque se añade al archivo COW (click)
Lecturas: se hacen del archivo COW
Si se desea descartar los cambios: se borran los bloques asignados al archivo COW y las tablas (nodo-i, bloques indirectos...) del archivo COW
Si los cambios se desean hacer permanentes: se borran los bloques originales que hayan sido modificados, y la tabla del archivo modificado sustituye a la del original

COW sobre archivos, a nivel de proceso de usuario

La información del archivo debe ser estructurada, cada dato se localiza mediante una clave o por su posición. Además, el archivo COW es indexado, y sólo contiene los cambios



Comentado [id35]: Supongamos que se desea realizar COW sobre un archivo, a nivel de proceso de usuario. Es necesario que el archivo contenga información que se pueda localizar o bien por su posición (archivo directo) o mediante el valor de un índice (archivo indexado)....
- Para iniciar la COW, se crea un archivo de COW que será de tipo indexado, usando el mismo tipo de índice que el archivo original (si el archivo original es directo, se usará como índice la posición de la información). Este archivo no se modifica mientras dure la COW.
- Si se pretende modificar un dato asociado a la clave "id1" en el archivo original: el archivo original no se modifica, y se añade el nuevo dato al archivo COW
- Si se pretende añadidr un dato asociado a la clave "id4": se añade al archvio COW
- Si se intenta leer el dato asociado a una clave: se busca en primer lugar en el archivio .COW. Si no se encontrara en dicho archivo (como es el caso de la clave "id2") entonces se buscaría en el archivo original.
- Para descartar los cambios: basta borrar el archivo COW y todo queda como al principio.
- Si se desean hacer permanente los cambios: hay que fusionar el archivo COW con el original, actualizando el archivo original con los datos del archivo COW. Una vez hecho, se borra el archivo COW

COW sobre volúmenes

Se utiliza el mismo método que sobre archivos a nivel de procesos. Consideramos volumen como un archivo de bloques (acceso directo). Además, el archivo COW debe estar en otro volumen y el Sistema Virtual de Archivos usa archivo COW para simular un volumen.

Aplicaciones COW:

- **Creación de *instantáneas* de archivos o volúmenes**: permite revertir cambios muy eficientemente
- **Optimización de espacio**: un volumen puede contener la información común a múltiples volúmenes implementados como archivos COW

JOURNALING

¿Qué ocurre si durante una actualización del sistema de archivos (por ejemplo) el sistema cae?

¡EN EFECTO!
El sistema de archivos puede quedar en un estado inconsistente

La solución sería que todas las actualizaciones sobre las estructuras del sistema de archivo formarán parte de una transacción, conjunto de actualizaciones que se realizan de manera atómica (o se realizan todas o no se realiza ninguna).

Ejemplo de primitivas sobre transacciones: BEGIN_TRANSACTION, END_TRANSACTION / COMMIT, CANCEL_TRANSACTION / ABORT / ROLLBACK

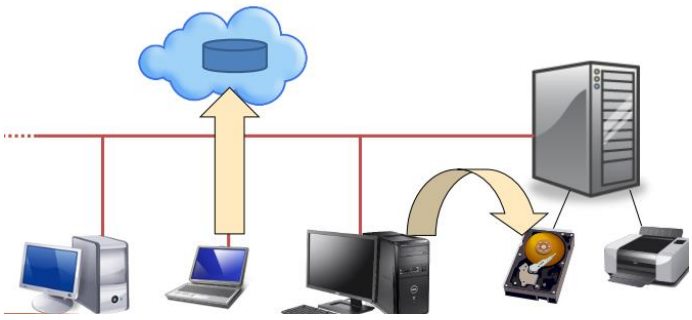
Las posibles implementaciones podrían ser a través de una copia en caso de escritura (complejo en este contexto) o el **Journaling.**

Resgistro de Escritura Anticipada (log/Journal)

Se basa en el uso de un Registro de Escritura Anticipada (log/Journal). Al iniciar cada transacción se crea un archivo de log y antes de cada actualización, se escribe en log: el valor original que se va a modificar y el valor por el que se sustituye. Si transacción se valida se borra archivo de log, pero si la transacción se cancela se recorre archivo de log en sentido inverso, deshaciendo cambios. Tras recuperarnos de un fallo se detecta si había transacciones en vuelo por sus archivos de log.

SISTEMAS DE ARCHIVOS DISTRIBUIDOS

Desde un sistema conectado en red podemos acceder tanto a unidades en un servidor como a unidades en la nube.



Principales tipos de unidades remotas

- **Unidades de bloques:** el servidor expone una abstracción equivalente a unidad física y recibe peticiones a nivel de *driver* de unidad. El sistema de archivo (formato) lo decide y crea el cliente. Es un tipo de unidad no pensado para ser compartida por múltiples clientes.
  El nivel de administración de archivos está en cliente y el servidor soporta operaciones de lectura y escritura de bloques.

Comentado [id36]: Supongamos que estamos en el sistema de ficheros FAT y tenemos bloques de reserva (algunos bloques van a estar en memoria principal). P1 actualiza un bloque (está en memoria principal, porque se almacena en un bloque de reserva). Y ahora... la memoria principal es volátil, es decir, si no recibe alimentación (energía) => los datos de memoria principal se pierden

Ejemplo:
Sup fichero f1.txt y mi fichero f1.txt está compuesto por 3 bloques => bloque 10, bloque 21 y bloque 22. El proceso P actualiza el bloque 10 de f1.txt. CHAS, corte luz y se pierde la actualización hecha sobre el bloque 10

Sup ahora f1.txt tiene 3 bloques => bloque 10, bloque 21 y bloque 22. P1 actualiza bloque 10 y P1 actualiza bloque 22
SO por estrategia de write-back escribe el bloque 10 a disco
CHAS, corte luz. f1.txt tiene bloque 10 (actualizado), bloque 21 (antiguo) y bloque 22 (antiguo)Esto implica que el fichero estará corrupto, la información es incoherente y perderíamos el fichero

Comentado [id37]: Journal: se guardan copias del bloque original y se anota también en archivo de log (la operación pendiente)

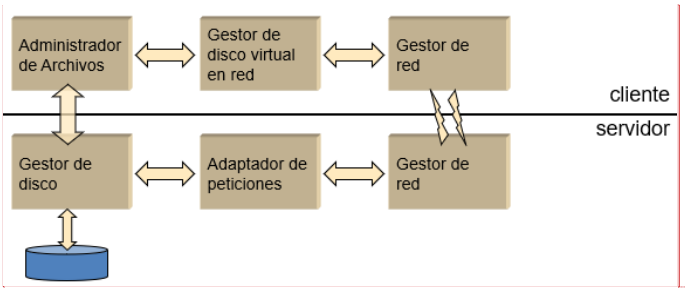
Para aliviar el journaling podríamos cortar la escritura de bloques que están en la zona de reserva

Sistema de ficheros ext2 y FAT no soportan journaling
ext3 si soporta journaling (ext3 = ext2 + journaling)

Ejemplo:
f1.txt tiene 3 bloque => 10, 21 y 22
P1 actualiza 10
Con el journal lo que se sucede es que: se copia el bloque 10 a un bloque en el espacio del journal (se guarda una copia del bloque 10), actualiza el bloque 10 (anotando en el journal la operación), actualiza el bloque 21 (se guarda una copia de bloque 21 en el journal), se vuelca el bloque 10 al disco.
CHAS, corte de luz
Cuando arranca de nuevo el SO con un sistema de ficheros con journaling: primero se consulta el journal y nos encontramos que hay operaciones pendientes. f1.txt: bloque 10 actualizó (y se volcó a disco) y el bloque 21 se actualizó (pero no se volcó a disco). El sistema de fichero revierte el cambio en el bloque 10 (el bloque 21 se perdió porque estaba en memoria volátil)

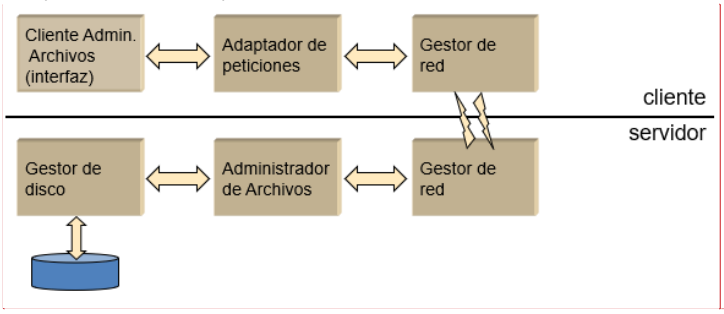
Comentado [id38]: Sistema de ficheros de red:
- •Network FileSystem (NFS)
- •SMB
- •CIFS

El sistema de ficheros virtual hace que el acceso a ese fichero que está en red sea exactamente igual que si estuviera en local



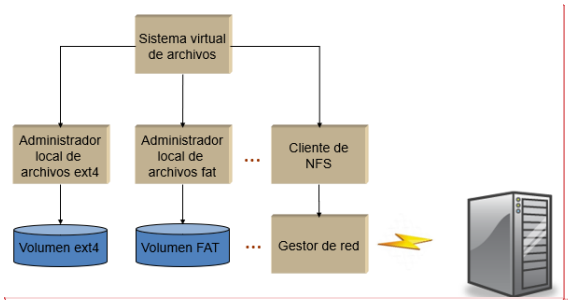
Comentado [id39]: •El **administrador de archivos** es un administrador completo
•El **gestor de disco virtual en red** simula la interfaz de un gestor de disco. Administrador de archivos no tiene por qué saber que se trata de un disco en red.
•El **gestor de disco virtual** es muy simple: utilizando el gestor de red, envía las peticiones que recibe al servidor
•En el **servidor**, el adaptador de peticiones recibe las peticiones recibidas a través de la red del cliente, y se las pasa al gestor de disco, simulando un administrador de archivos, y devolviendo los resultados al gestor de disco virtual en red del cliente.
•El **gestor de disco del servidor** cree que recibe peticiones directamente del administrador de archivos
•Realmente, es como si existiese una conexión virtual entre administrador de archivos y gestor de disco (click)

- **Sistemas de archivos lógicos**: el servidor expone un sistema de archivos y recibe peticiones a nivel de administrador de archivos. El sistema de archivos a usar lo decide y crea el servidor.
  El servidor contiene administrador de archivos completo mientras que el cliente tiene servidor de archivos mínimo, no es más que un cliente del administrador en el servidor e implementa a nivel superior una interfaz de administrador de archivos.



Comentado [id40]:

En la práctica un mismo sistema puede ser cliente y servidor. En un mismo sistema deben convivir sistemas de archivos locales y remotos. Y, además, de distinto tipo (NTFS, ext2, ext3, ext4, NFS...). Para implementar todo eso, se utiliza una **Virtual File System**.



Comentado [id41]: El sistema virtual de archivos es un front-end que dispone de un único sistema de archivos, en el que se integran todos los sistemas de archivos. Para ello, cada sistema de archivos se monta sobre un directorio de un árbol común de directorios
Se pueden integrar nuevos sistemas de archivos. Para ello, basta instalar el código del administrador de archivos correspondiente en el núcleo del sistema operativo

AUTENTICACIÓN DEL USUARIO

La autenticación del usuario permite el acceso al sistema sólo a los usuarios autorizados. La implementación puede ser por autenticación local o por autenticación como servicio (AaaS).

Autenticación local

- **Mediante información secreta:** el usuario comparte algún tipo de información con el sistema y si es correcta, se permite acceso. Es un método simple y barato pues no hacen falta dispositivos, elementos físicos o infraestructura externa. La forma más habitual de implementar este sistema es mediante contraseñas que se guardan encriptadas con algoritmo unidireccional. Se solicita la contraseña a usuario, se encripta, y se comprueba coincidencia con la almacenada.
- **Elementos físicos**: usuario se identifica mediante algún objeto legible por sistema (tarjeta con chip, tarjeta con lector por proximidad). Para prevenir uso indebido se refuerza mediante una contraseña que se guarda encriptada en la propia tarjeta. No hay que almacenar ni transferir claves. Sin embargo, se necesitan dispositivos físicos y existe la posibilidad de duplicado de tarjeta.
- **Técnicas biométricas**: se usa alguna característica física del propio usuario como identificación (huella digital, capilares de retina, examen del iris, análisis de voz). Hay una muy baja tasa de aceptación indebida. Sin embargo, se necesitan elementos físicos potencialmente caros y algunos métodos posee una elevada tasa de rechazos indebidos.

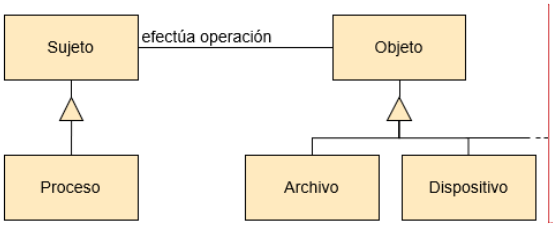
Autenticación como servicio (AaaS)

Protocolos que permiten delegar la autenticación se delega en un sistema externo que establece conexión segura con dicho sistema. El servidor se encarga de autenticar el usuario, e informa al sistema sobre la identidad autenticada. Normalmente, el servidor usa contraseña. La seguridad está centralizada en un servidor de autenticación que puede estar detrás de un *cortafuegos*. También podemos usar Sistemas *"one only signature"* donde nos autenticamos una sóla vez para múltiples servicios. No obstante, necesitamos de una infraestructura de red y de servidor.

- **LDAP** (Lightweight Directory Access Protocol): formado por un directorio LDAP. Consiste en un conjunto de objetos organizados de forma lógica y jerárquica (en forma de árbol). Almacena información de autenticación (usuario y contraseña) e información adicional (permisos, certificados, datos contacto) Algunas implementaciones LDAP pueden ser Active Directory, OpenLDAP, Red Hat Directory
- **Oauth** (Open Authorization): Estándar abierto que permite compartir información de autenticación. Permite a un usuario de un sistema A (proveedor) compartir información de su cuenta de usuario en A con un sistema B (consumidor). Proporciona un API estándar para las aplicaciones. El sistema es utilizado por GAFAM (Google, Amazon, Facebook, Apple, Microsoft)

CONTROL DE ACCESO

Una vez que usuario ha sido autenticado sólo debe acceder a los recursos que esté autorizado. ¿Cómo sabemos a qué recursos puede acceder un **proceso**? Se utiliza un sistema informático compuesto por **sujetos** y **objetos** donde los sujetos efectúan operaciones sobre objetos.



Comentado [id42]: Realmente, el usuario no accede a ningún recurso. Son sus **procesos** quien accede a los recursos.

Sujetos: procesos
Objetos: cualquier recurso que se desee proteger: archivos, segmentos, estructuras de datos, etc…

Protocolo de control de acceso

Está formado por dos elementos:

- **Derecho de acceso**: autorización para aplicar una operación sobre un objeto
- **Dominio de protección**: Conjunto de derechos de accesos. Asignar un dominio de protección a un sujeto es equivalente a concederle los derechos de acceso del dominio. Características de los dominios de protección:
  - Pueden solaparse (intersección no vacía)
  - Son dinámicos
  - Sujetos pueden cambiar de dominio

Matriz de Control de Acceso

Para mantener toda esta información sobre dominios, objetos y sujetos usamos una **Matriz de Control de Acceso** que relaciona conjunto de dominios con conjunto de objetos. Los dominios pueden considerarse a su vez objetos sobre los que se puede aplicar la operación **entrar.**

Objetos

Dominios	Archivo1	Archivo2	Archivo3	D1	D2	D3
D1	Leer Escribir		Ejecutar		Entrar	
D2	Leer	Leer Escribir		Entrar		Entrar
D3		Escribir	Ejecutar Leer			

Comentado [id43]: Cada entrada de la matriz contiene las operaciones aplicables sobre el objeto en ese dominio
Entrada en blanco: no se puede hacer nada con ese objeto en ese dominio

PREGUNTA EXAMEN:
Supongamos que a los procesos, cuando inician su ejecución, se le adscribe al dominio D1…
P: ¿Pueden leer achivo D1?
R: Sí, de cajón.
P: ¿Pueden leer Archivo2?
R: Sí, pueden entrar en dominio D2 y ahí leerlo
P: ¿Pueden copiar Archivo3 sobre Archivo1?
R: NO: pues para poder leer Archivo3 necesitan entrar en D3, y en dicho dominio, no se puede escribir Archivo1.
NOTAS: Si no existe una operación COPIAR, se supone que para copiar es necesario leer y poder escribir en el destino
Para poder copiar, es necesario poder leer el origen y escribir el destino en el mismo dominio, a menos que se especifique que hay permiso para crear archivos temporales o algún otro mecanismo para mantener la información temporalmente

Sin embargo, la matriz control de acceso es poco práctica pues en un sistema real la matriz sería enorme. Además, sería ineficiente dado que la mayoría de las entradas estarían vacías.

Para solucionar eso, se plantea la idea de *cortar* la matriz de control de acceso por columnas. Se crea una l**ista de control de acceso (ACL) de un objeto** que relacionan los derechos sobre dicho objeto tienen los distintos sujetos. La lista estará formada por las entradas no vacías de su columna en la matriz de control de acceso.

Archivo1: {(D1, [Leer, Escribir]), (D2, [Leer])}
Archivo2: {(D2, [Leer, Escribir]), (D3, [Escribir])}
Archivo3: {(D1, [Ejecutar]), (D3, [Ejecutar, Leer])}

	Archivo1	Archivo2	Archivo3
D1	Leer Escribir		Ejecutar
D2	Leer	Leer Escribir	
D3		Escribir	Ejecutar Leer

Para su implementación se asocia a cada archivo una lista de control de acceso a cada archivo. El propietario del archivo puede manipular su lista. Además, el SO comprueba la lista **por cada acceso** a un archivo. Sin embargo, surge el inconveniente de coste de exploración de lista por cada acceso. Para su simplificación se podría comprobar la lista sólo al abrir fichero, pero se podrían producir incoherencias (un administrador nos puede quitar los permisos del fichero mientras nosotros lo estamos modificando, revocación). Para solucionarlo, se comprueba se cerraría el archivo o se denegaría el permiso. También, podemos abreviar las listas mediante comodines para expresar que un archivo tiene todos los permisos.

Archivo4: {([*,profesores], [leer, escribir]), ([*,alumnos], [leer])