ARQUITECTURA Y SISTEMAS OPERATIVOS

CLASE 8

SISTEMA DE ARCHIVOS

BIBLIOGRAFIA

- Operating System Concepts. By Abraham, Silberschatz.
  - Capítulo XIII

TEMAS DE LA CLASE

- Concepto de Archivo
  - Atributos de un archivo
  - Operaciones sobre archivos
  - Gestión de archivos abiertos
  - Tipos de archivos
  - Estructura de archivo
  - Estructura interna de un archivo
- Métodos de Acceso
  - Acceso secuencial
  - Acceso directo
- Estructura de Directorio
  - Directorios de 1 nivel
  - Directorios de 2 niveles
  - Directorios de estructura de árbol

TEMAS DE LA CLASE

- Protección
  - Tipos de acceso
  - Control de acceso
- Archivos Mapeados en Memoria
  - Mecanismo básico

CONCEPTO DE ARCHIVO

CONCEPTO DE ARCHIVO

Para la mayoría de los usuarios, el sistema de archivos es el aspecto más visible de un sistema operativo de propósito general. Consta de dos partes diferenciadas: un conjunto de archivos, donde cada uno almacena datos relacionados, y una estructura de directorios que los organiza y proporciona información sobre ellos.

El sistema operativo se basa en las propiedades físicas de sus dispositivos de almacenamiento para definir una unidad de almacenamiento lógico: el archivo. Estos dispositivos suelen ser no volátiles, por lo que su contenido persiste entre reinicios del sistema.

Un archivo es una colección de información relacionada, con un nombre, que se registra en almacenamiento secundario. Desde la perspectiva del usuario, es la asignación más pequeña de almacenamiento lógico; es decir, los datos no se pueden escribir en almacenamiento secundario a menos que estén dentro de un archivo.

CONCEPTO DE ARCHIVO

Los archivos representan comúnmente programas (en formato fuente u objeto) y datos. Son secuencias de bits, bytes, líneas o registros, cuyo significado lo definen su creador y el usuario. Por tanto, el concepto de archivo es extremadamente general.

Ejemplos:

- Archivo de texto: secuencia de caracteres organizados en líneas.
- Archivo fuente: secuencia de funciones o declaraciones.
- Archivo ejecutable: contiene secciones de código listas para cargarse en memoria y ejecutarse.

CONCEPTO DE ARCHIVO

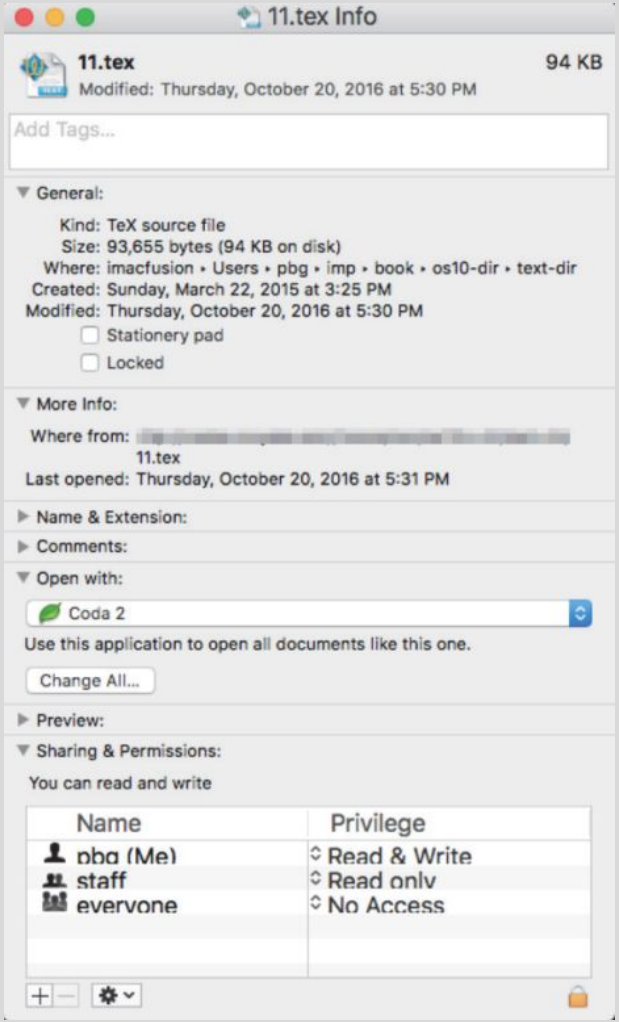
- **Atributos de un archivo**

Los atributos varían según el sistema operativo, pero suelen incluir:

- Nombre: identificador simbólico del archivo.
- Identificación: etiqueta única que lo distingue dentro del sistema.
- Tipo: distingue los distintos formatos reconocidos por el sistema.
- Ubicación: dispositivo y posición en el mismo.
- Tamaño: espacio ocupado actualmente y, en algunos casos, máximo permitido.
- Protección: control de acceso (lectura, escritura, ejecución, etc.).
- Timestamps e ID de usuario: fechas de creación, modificación y último acceso, útiles para seguridad y monitoreo.

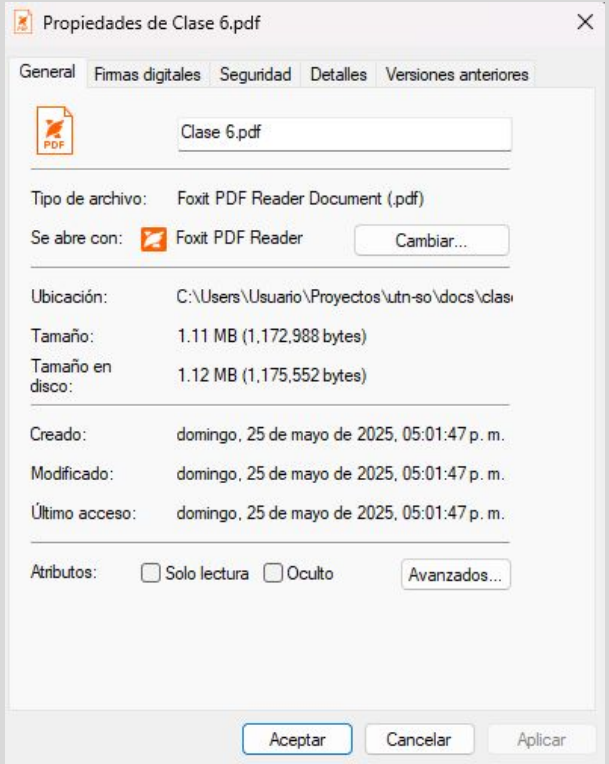
Sistemas más modernos pueden incluir atributos extendidos, como codificación de caracteres o funciones de seguridad como checksums.

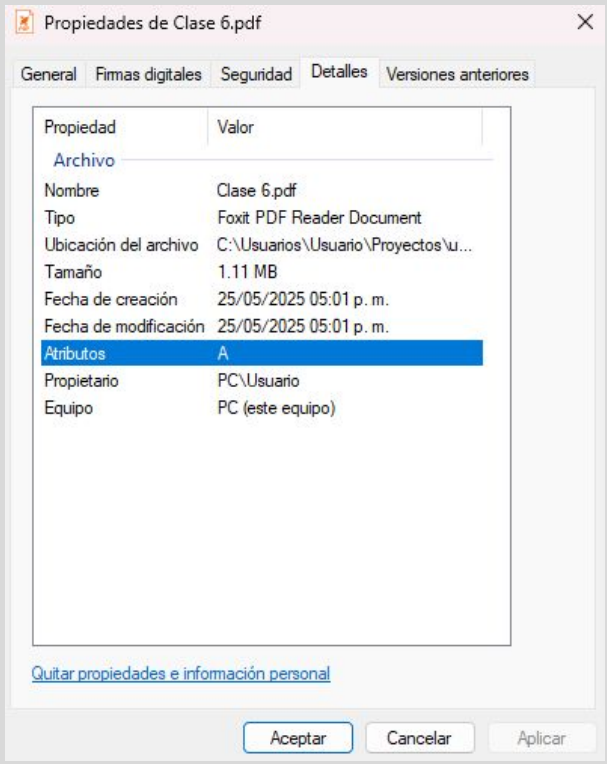
Esta información se almacena en la estructura de directorios, que reside en el mismo dispositivo físico que los archivos.



Sistemas más modernos pueden incluir atributos extendidos, como codificación de caracteres o funciones de seguridad como checksums.

Esta información se almacena en la estructura de directorios, que reside en el mismo dispositivo físico que los archivos.





CONCEPTO DE ARCHIVO

- **Operaciones sobre archivos**

Un archivo es un tipo de dato abstracto, y el sistema operativo debe definir qué operaciones se pueden realizar sobre él. Las operaciones básicas incluyen:

1. Crear: requiere encontrar espacio en el sistema de archivos y crear una entrada en el directorio.
2. Abrir: devuelve un manejador (*handler*) que se utilizará en las demás operaciones.
3. Escribir: especifica el *handler* y la información a escribir. Si es secuencial, el sistema mantiene y actualiza un puntero de escritura.
4. Leer: especifica el *handler* y la posición destino en memoria. También actualiza un puntero de lectura si es secuencial.
5. Reposicionar (*seek*): cambia la posición del puntero sin realizar E/S.
6. Eliminar: libera el espacio del archivo y borra su entrada del directorio.
7. Truncar: elimina el contenido pero conserva atributos como nombre o permisos.

CONCEPTO DE ARCHIVO

Estas siete operaciones básicas comprenden el conjunto mínimo de operaciones sobre archivos necesarias, y pueden combinarse para realizar otras operaciones de archivo. Por ejemplo, para crear la copia de un archivo, creamos uno nuevo, leemos del archivo origen y escribimos en el nuevo archivo.

Ejemplo de implementación:

- https://github.com/JereFassi/utn-so/blob/main/src/05-archivos/copy_file_example.c

CONCEPTO DE ARCHIVO

- **Gestión de Archivos Abiertos**

La mayoría de las operaciones de archivo implican la búsqueda en el directorio de la entrada asociada al archivo solicitado. Para evitar esta búsqueda constante, muchos sistemas requieren que se realice una llamada al sistema `open()` antes de usar un archivo por primera vez.

El sistema operativo mantiene una tabla de archivos abiertos (*open-file table*), que contiene información sobre todos los archivos abiertos. Cuando se solicita una operación de archivo, se especifica mediante un índice en esta tabla, evitando búsquedas repetidas.

Cuando un archivo ya no se utiliza activamente, el proceso lo cierra (`close()`), y el sistema operativo elimina su entrada de esta tabla. Las llamadas `create()` y `delete()` solo funcionan sobre archivos cerrados.

CONCEPTO DE ARCHIVO

- **Gestión de Archivos Abiertos**

La llamada a `open()` normalmente devuelve un puntero a la entrada correspondiente en la tabla, y es este puntero —no el nombre del archivo— el que se utiliza en las operaciones de E/S, simplificando la interfaz del sistema operativo.

En resumen, la tabla de archivos abiertos mantiene:

- Puntero de archivo: indica la posición actual de lectura/escritura, único por proceso.
- Contador de archivos abiertos: lleva el registro de cuántos procesos tienen el archivo abierto; la entrada se elimina cuando este contador llega a cero.
- Ubicación del archivo: permite acceso sin tener que leer el directorio nuevamente.
- Derechos de acceso: indica el modo en que se abrió el archivo (lectura, escritura, etc.).

CONCEPTO DE ARCHIVO

- **Bloqueo de Archivos**

Los sistemas operativos en general, ofrecen funciones para bloquear archivos abiertos. Los bloqueos permiten a los procesos controlar el acceso concurrente a un archivo:

- Bloqueo compartido: varios procesos pueden leer simultáneamente.
- Bloqueo exclusivo: solo un proceso puede escribir o acceder en modo exclusivo.

Con esto, los sistemas operativos pueden ofrecer mecanismos de bloqueo obligatorios o consultivos.

- Obligatorio: impuesto por el sistema operativo, es el mismo sistema quien garantiza la integridad del bloqueo (ej. Windows).
- Consultivo: los desarrolladores de software deben garantizar que los bloqueos se adquieran y liberen correctamente (ej. UNIX).

CONCEPTO DE ARCHIVO

- **Tipos archivos**

Una técnica común para implementar tipos de archivo es incluir el tipo como parte del nombre del archivo. El nombre se divide en dos partes: un nombre y una extensión, generalmente separados por un punto. Algunos ejemplos son `resume.docx`, `server.c` y `ReaderThread.cpp`. El sistema utiliza la extensión para indicar el tipo de archivo y el tipo de operaciones que se pueden realizar en él. Por ejemplo, solo se pueden ejecutar archivos con extensión .exe o .sh.

Los programas de aplicación también utilizan extensiones para indicar los tipos de archivo que les interesan. Por ejemplo, los compiladores de Java esperan la extensión `.java`, y Microsoft Word espera la extensión `.doc` o `.docx`.

Tipos de archivos:

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

CONCEPTO DE ARCHIVO

- **Estructura lógica de un archivo**

La estructura lógica de un archivo varía según su tipo. Por ejemplo:

- Los archivos fuente u objeto tienen estructuras predecibles que se ajustan a las expectativas de los programas que los leen.
- Mientras que archivos ejecutables requieren una estructura específica que el sistema operativo pueda determinar dónde cargarlo en la memoria y dónde se ubica la primera instrucción.

Los sistemas operativos que admiten muchas estructuras se vuelven más complejos. Por eso, muchos como UNIX o Windows adoptan una estructura mínima común.

Por ejemplo, UNIX considera todos los archivos como un *streams de bytes*, donde cada byte es direccionable por su desplazamiento desde el inicio. Estos bytes se empaquetan en bloques físicos (por ejemplo, de 512 bytes), lo cual es gestionado automáticamente.

CONCEPTO DE ARCHIVO

- **Estructura Interna y Fragmentación**

La E/S en disco se realiza en bloques físicos (de tamaño fijo), pero los registros lógicos (contenido significativo para el usuario) pueden variar. Por eso, para gestionar el espacio y la organización del almacenamiento secundario, es común empaquetar varios registros lógicos en un bloque físico.

Pero esto introduce fragmentación interna: espacio no aprovechado dentro de un bloque si los datos no lo llenan completamente. Todos los sistemas sufren de fragmentación interna, y a mayor tamaño del bloque, mayor puede ser la fragmentación. Factores relevantes:

- Tamaño del registro lógico (por ej., 1 byte en UNIX).
- Tamaño del bloque físico (por ej., 512 bytes).
- Técnica de empaquetado: hecha por el sistema o por la aplicación.

Finalmente, todas las operaciones básicas de E/S se realizan en términos de bloques, lo que influye en el rendimiento y la eficiencia del sistema de archivos.

MÉTODOS DE ACCESO

MÉTODOS DE ACCESO

Dependiendo del tipo de aplicación y del dispositivo de almacenamiento, los archivos pueden ser accedidos de distintas maneras. Los dos métodos más comunes son el acceso secuencial y el acceso directo (o aleatorio).

- **Acceso Secuencial**

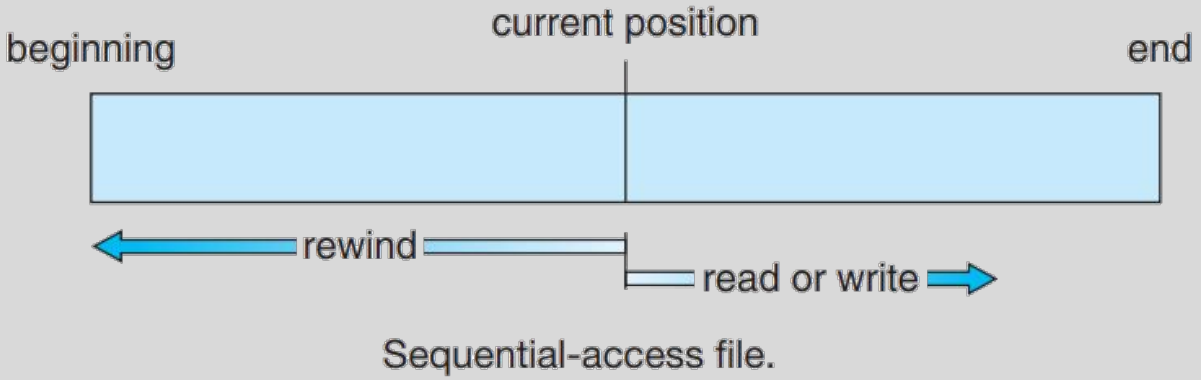
En acceso secuencial, los datos del archivo se procesan en orden, uno tras otro, como si fueran una cadena continua de registros. Este es el método más común de acceso y es utilizado por herramientas como editores de texto o compiladores.

El sistema mantiene un puntero de archivo, que rastrea la ubicación actual dentro del archivo. Cada operación de lectura (`read_next()`) recupera la porción siguiente del archivo y mueve el puntero hacia adelante automáticamente. De manera análoga, una operación de escritura (`write_next()`) agrega datos al final del archivo y actualiza el puntero al nuevo final del contenido.

MÉTODOS DE ACCESO

Aunque el acceso es secuencial, los programas pueden reiniciar el puntero al principio del archivo, o incluso saltar hacia adelante o hacia atrás una cantidad determinada de registros, si el sistema lo permite.

Este modelo está inspirado en el funcionamiento de una cinta magnética, donde los datos deben leerse o escribirse en orden. Sin embargo, funciona también en dispositivos modernos de acceso aleatorio, como discos.



Sequential-access file.

ASIGNACIÓN DE MEMORIA CONTIGUA

- **Acceso directo**

El acceso directo (también llamado aleatorio) permite que los datos se lean o escriban en cualquier orden, sin necesidad de seguir una secuencia. Para esto, el archivo debe estar compuesto por registros lógicos de tamaño fijo, lo que facilita calcular la ubicación exacta de un dato.

Este modelo está basado en la estructura de los discos, que permiten acceder directamente a cualquier bloque. En este caso, el archivo se visualiza como una secuencia numerada de bloques o registros. Por ejemplo, un programa podría leer el bloque 14, luego el 53, y después escribir en el 7, sin seguir un orden lineal.

El acceso directo es ideal para aplicaciones que manejan grandes volúmenes de información y requieren búsquedas rápidas, como las bases de datos. Al llegar una consulta, se calcula qué bloque contiene los datos necesarios y se accede directamente a ese bloque.

ASIGNACIÓN DE MEMORIA CONTIGUA

- **Acceso directo**

Para utilizar este método, las operaciones del esquema secuencial deben ser adaptadas. En lugar de `read_next()`, se usa `read(n)`, donde **n** es el número de bloque; y en lugar de `write_next()`, se utiliza `write(n)`.

Una alternativa consiste en mantener las operaciones `read_next()` y `write_next()` sin modificar, y agregar la función `position_file(n)`, que reposiciona el puntero del archivo en el bloque deseado antes de realizar la operación.

El número de bloque que proporciona el usuario es relativo al inicio del archivo, es decir, el bloque 0 representa el primero, el 1 el segundo, y así sucesivamente. Este número relativo permite que el sistema operativo maneje las direcciones físicas reales del disco (por ejemplo, el bloque relativo 0 podría estar almacenado en el sector físico 14.703 del disco). Este enfoque protege la integridad del sistema, ya que el usuario nunca accede directamente a ubicaciones físicas que podrían pertenecer a otros archivos o al sistema mismo.

ASIGNACIÓN DE MEMORIA CONTIGUA

- **Acceso directo**

Es importante tener en cuenta que algunos sistemas operativos solo permiten uno de los métodos (secuencial o directo), mientras que otros permiten ambos. En algunos casos, el tipo de acceso debe declararse cuando se crea el archivo, y solo se puede usar de acuerdo con esa declaración.

Por último, simular un acceso secuencial sobre un archivo de acceso directo es posible, manteniendo una variable que indica la posición actual del puntero (por ejemplo, `cp`). Esto permite avanzar bloque a bloque como si se tratara de acceso secuencial. Sin embargo, hacer lo inverso —simular acceso directo sobre un archivo secuencial— resulta extremadamente ineficiente y complejo, ya que implicaría recorrer todo el archivo desde el principio hasta la posición deseada cada vez que se requiere acceso a un punto específico.

sequential access	implementation for direct access
reset	cp = 0;
read_next	read cp; cp = cp + 1;
write_next	write cp; cp = cp + 1;

Simulation of sequential access on a direct-access file.

ESTRUCTURA DE DIRECTORIO

ESTRUCTURA DE DIRECTORIO

Un directorio (o carpeta) puede verse como una tabla que traduce nombres simbólicos de archivos a sus ubicaciones físicas en el almacenamiento (normalmente apuntando a sus bloques de control). Además de esta función básica, el directorio debe permitir una serie de operaciones esenciales:

- Buscar archivos: localizar un archivo dentro del directorio a partir de su nombre. A veces se busca por coincidencias parciales (por ejemplo, todos los que empiecen con “doc”).
- Crear archivos: agregar nuevas entradas al directorio.
- Eliminar archivos: quitar la entrada del archivo del directorio (y posiblemente liberar espacio en disco). Esto puede implicar tareas como reorganizar o desfragmentar el directorio.
- Listar archivos: mostrar el contenido del directorio, es decir, todos los archivos y subdirectorios que contiene.
- Renombrar archivos: cambiar el nombre simbólico del archivo. A veces esto también implica cambiar su ubicación en la estructura.
- Recorrer el sistema: navegar por todo el sistema de archivos, accediendo a múltiples directorios y subdirectorios.

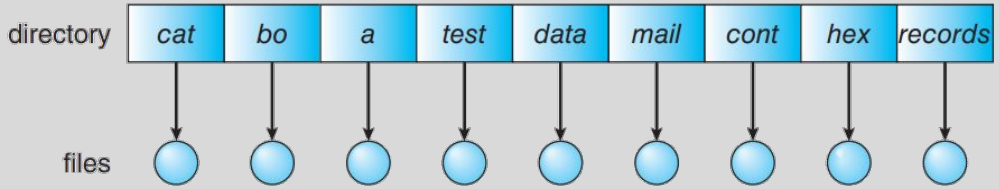
ESTRUCTURA DE DIRECTORIO

- **Directorio de un solo nivel**

Es la forma más simple de organización: todos los archivos están contenidos en un único directorio común. Es fácil de implementar, pero presenta varias limitaciones:

- Todos los archivos deben tener un nombre único, ya que no hay subdirectorios que los separen.
- No es adecuado para múltiples usuarios, porque los archivos de todos terminan mezclados en un solo lugar.

Aunque los sistemas modernos permiten nombres largos (hasta 255 caracteres) y resultaría relativamente sencillo generar nombres únicos por usuario, esto no resuelve completamente el problema cuando se trabaja con muchos archivos o usuarios.

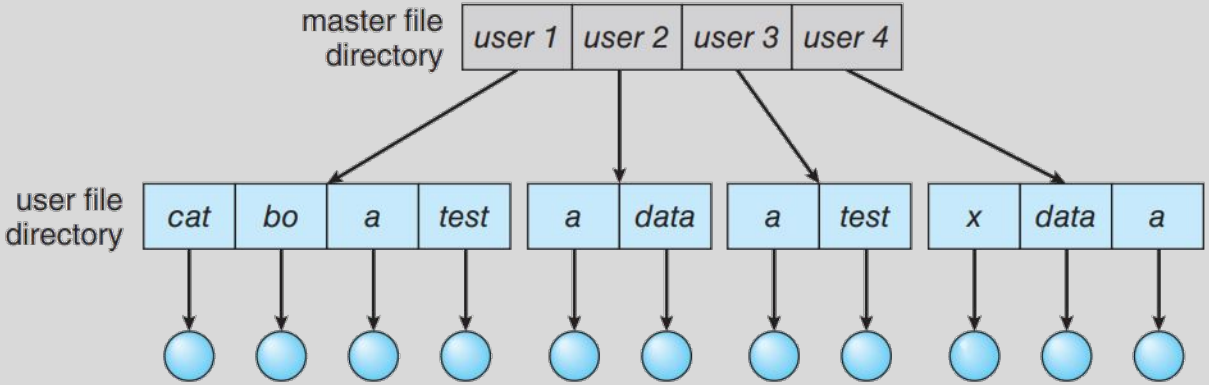


Single-level directory.

- **Estructura de dos niveles**

Para mejorar la organización y el soporte a múltiples usuarios, se puede usar una estructura de dos niveles donde:

- Cada usuario tiene su propio directorio personal (User File Directory o UFD).
- Todos los UFD se gestionan desde un directorio maestro (Master File Directory o MFD), que contiene referencias a los directorios individuales.

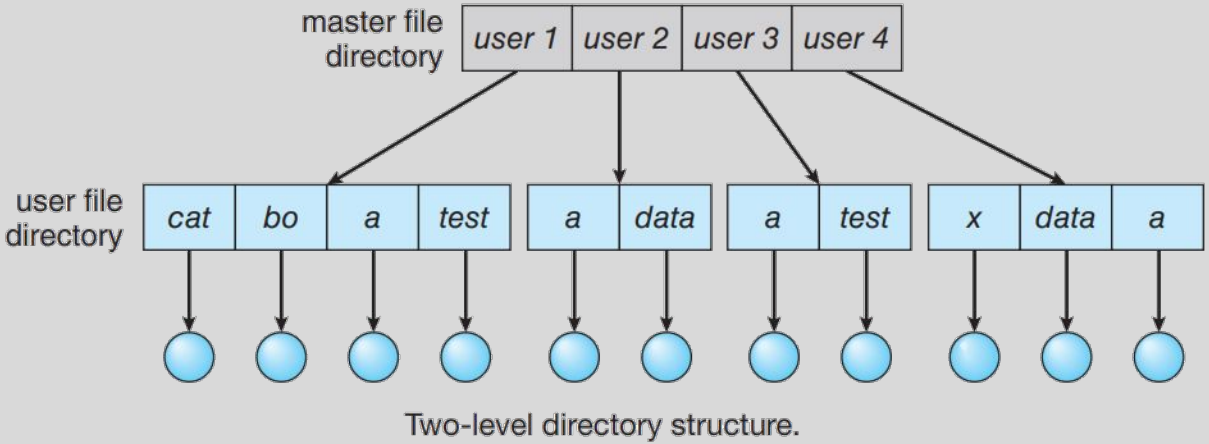


Two-level directory structure.

- **Estructura de dos niveles**

Con este enfoque:

- Cada usuario puede tener archivos con los mismos nombres que los de otros usuarios, sin conflictos.
- Las búsquedas, creaciones y eliminaciones se limitan al UFD del usuario activo.
- Se mejora la seguridad, ya que un usuario no puede borrar por accidente archivos de otro.

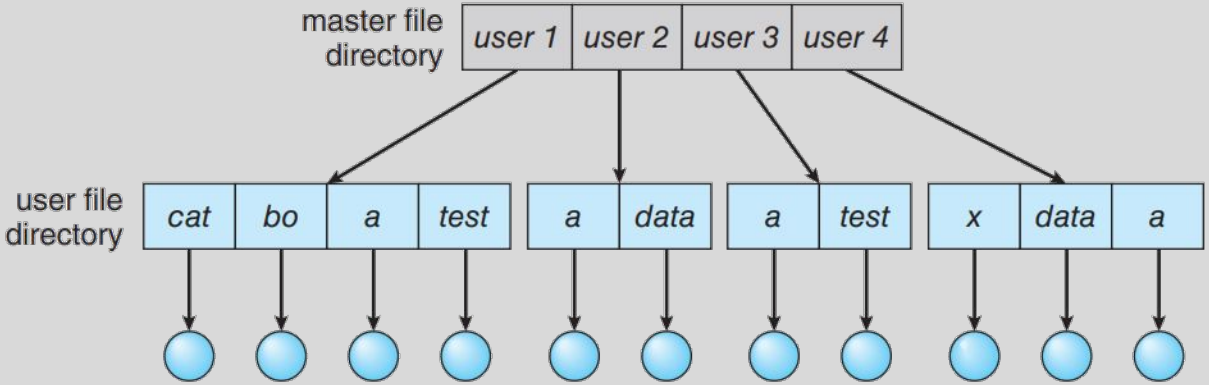


Two-level directory structure.

- **Estructura de dos niveles**

Sin embargo, hay una limitación: no hay posibilidad de compartir archivos entre usuarios fácilmente. Esto complica, por ejemplo, el acceso a herramientas del sistema (como compiladores o librerías), ya que estos archivos deberían copiarse a cada UFD, desperdiciando espacio.

Para resolverlo, se introduce una ruta de búsqueda: el sistema operativo primero busca el archivo en el UFD del usuario y, si no lo encuentra, lo busca en un directorio común (por ejemplo, uno asociado al “usuario 0”) donde están almacenados los archivos del sistema.

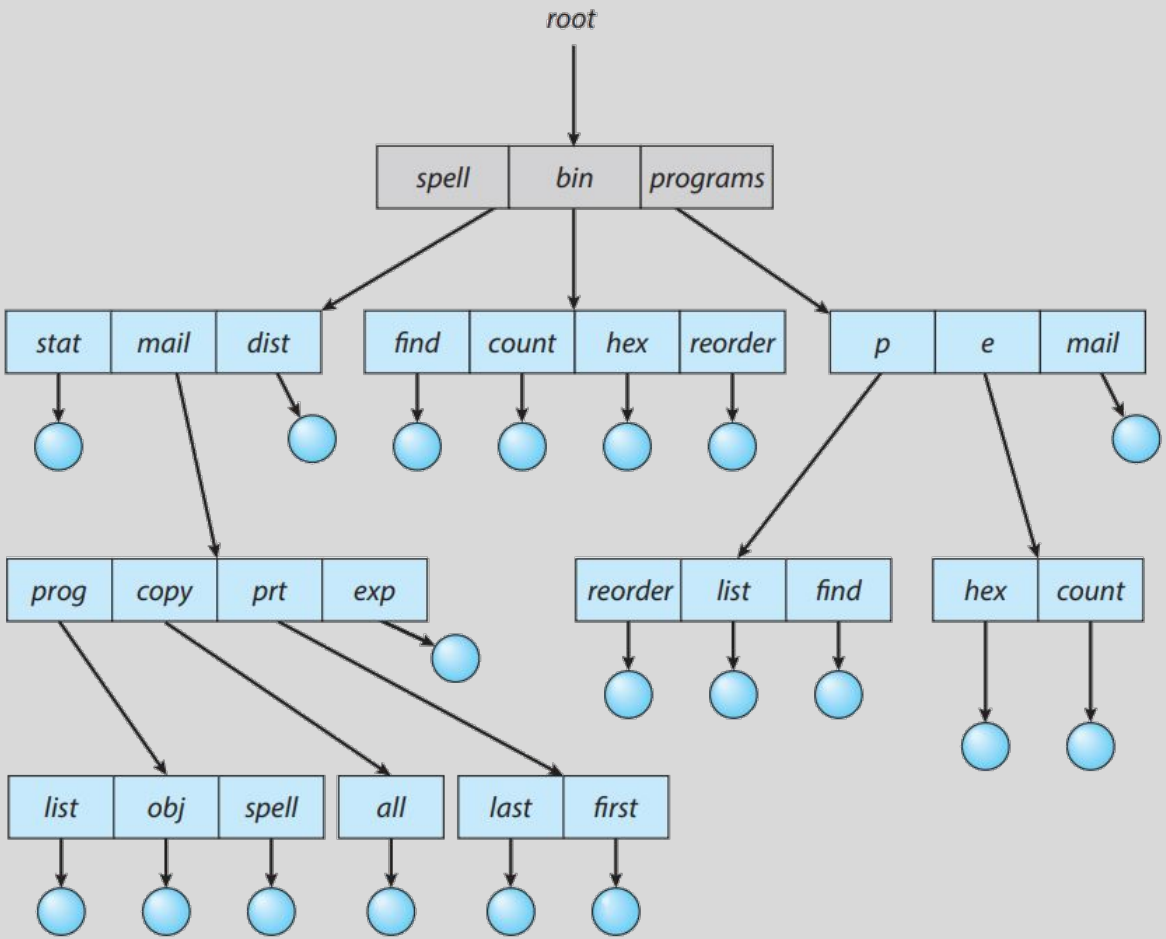


Two-level directory structure.

- **Estructura de árbol**

Una evolución natural de la estructura anterior es la estructura de árbol, que es la más común hoy en día:

- Se permite que los usuarios creen subdirectorios dentro de sus propios directorios, y así organizar jerárquicamente sus archivos.
- Existe un directorio raíz desde donde se ramifican todos los demás directorios y archivos.
- Cada archivo tiene una ruta única que permite localizarlo.

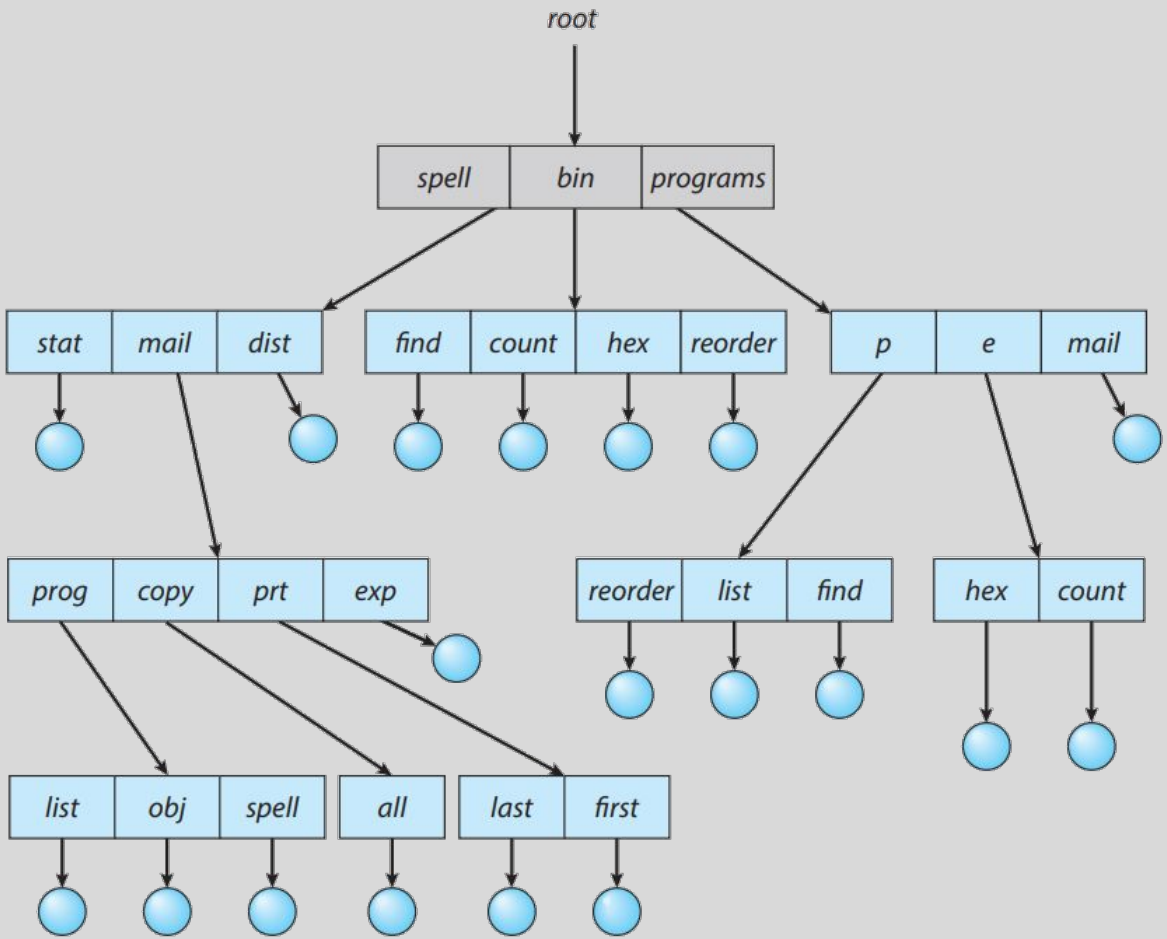


Tree-structured directory structure.

- **Estructura de árbol**

Características clave de esta estructura:

- Un directorio puede contener archivos y otros subdirectorios.
- Técnicamente, un directorio es un archivo especial, con formato interno específico. Cada entrada indica si se trata de un archivo o un subdirectorio (por ejemplo, con un bit que vale 0 para archivo y 1 para subdirectorio).
- Cada proceso tiene un directorio actual, que es donde se realizan por defecto las búsquedas de archivos. Si se desea acceder a otro archivo fuera del directorio actual, se debe especificar su ruta.

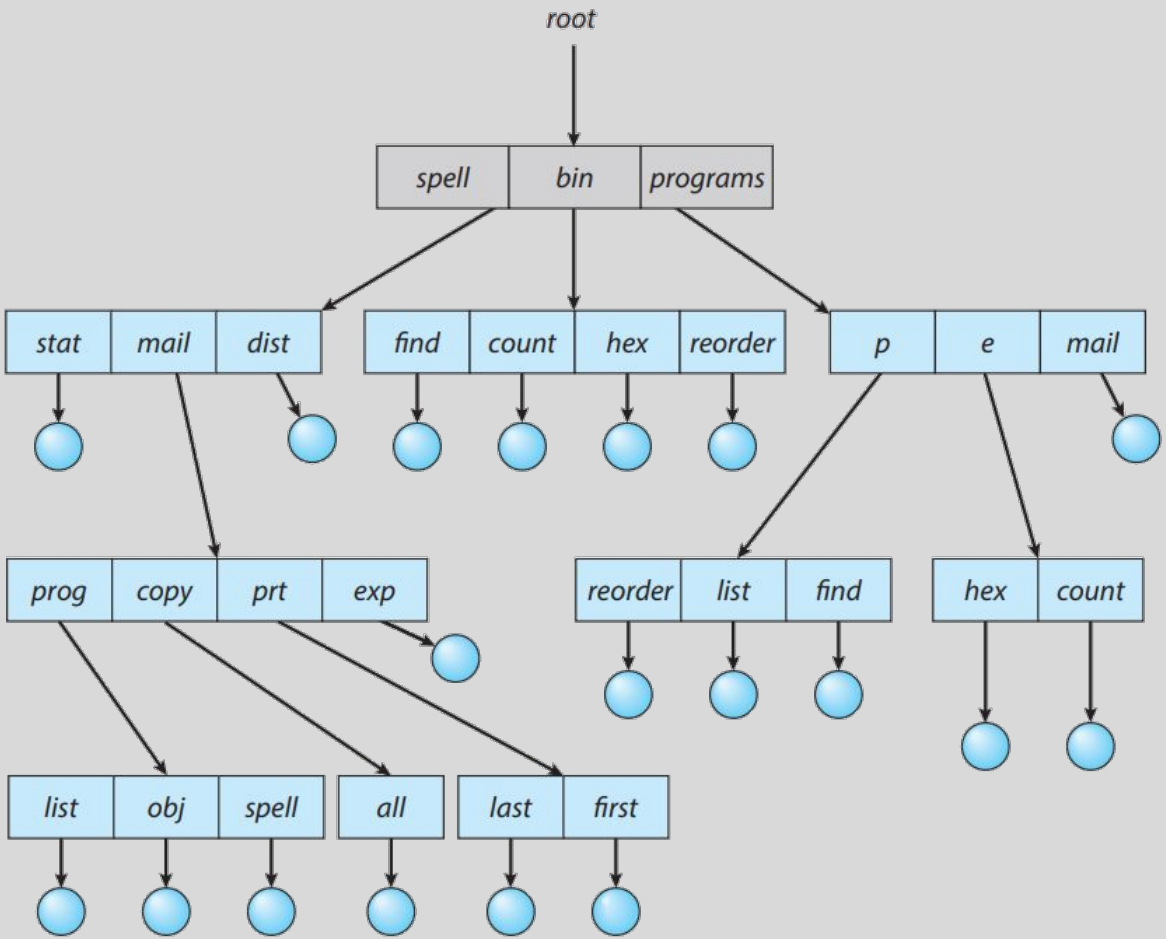


Tree-structured directory structure.

- **Estructura de árbol**

Hay dos tipos de rutas:

- Ruta absoluta: comienza desde la raíz (por ejemplo, `/usuario/docs/proyecto.txt`).
- Ruta relativa: parte del directorio actual (si estamos en `/usuario/docs`, entonces `proyecto.txt` o `../otros/proyecto2.txt` serían rutas relativas).

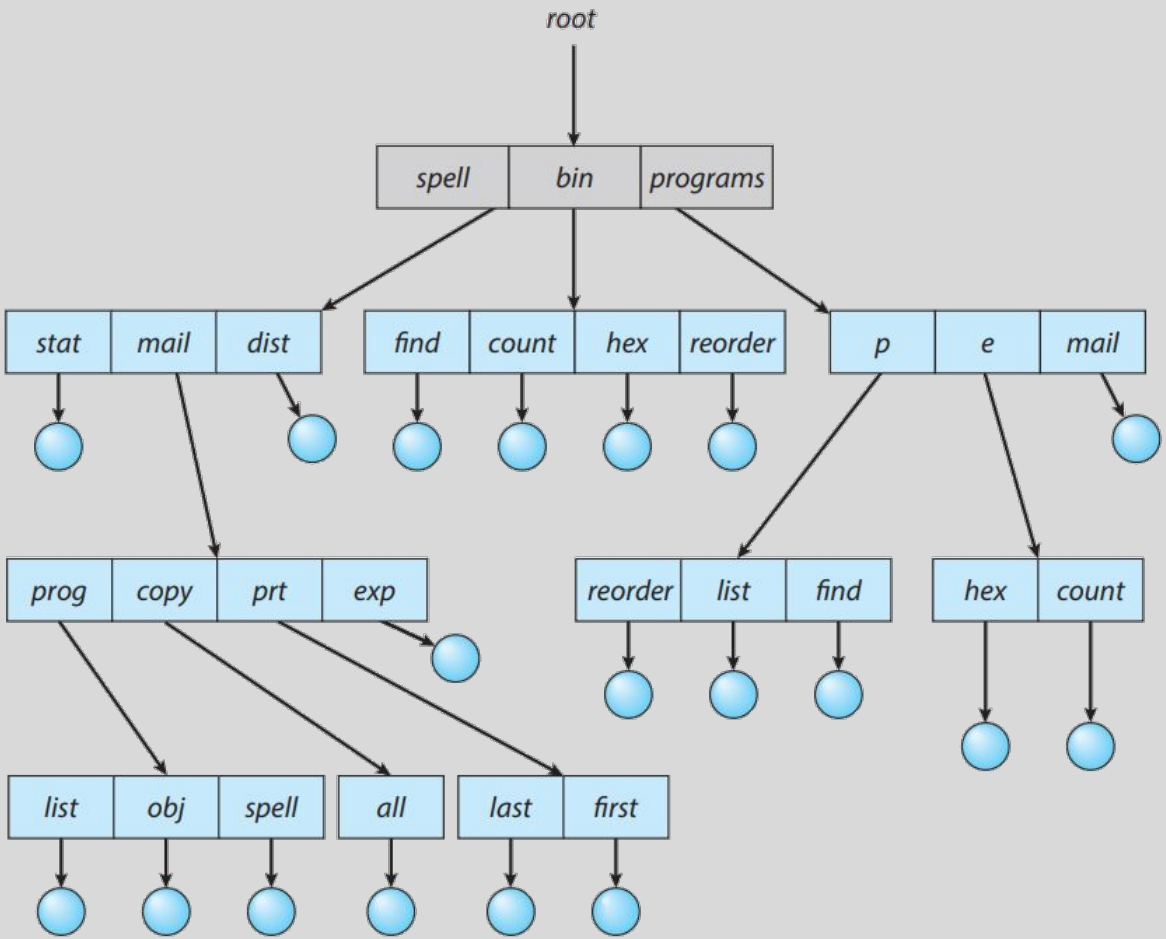


Tree-structured directory structure.

- **Estructura de árbol**

Eliminar directorios puede ser más complejo que eliminar archivos. Hay que tomar una decisión si el mismo no está vacío:

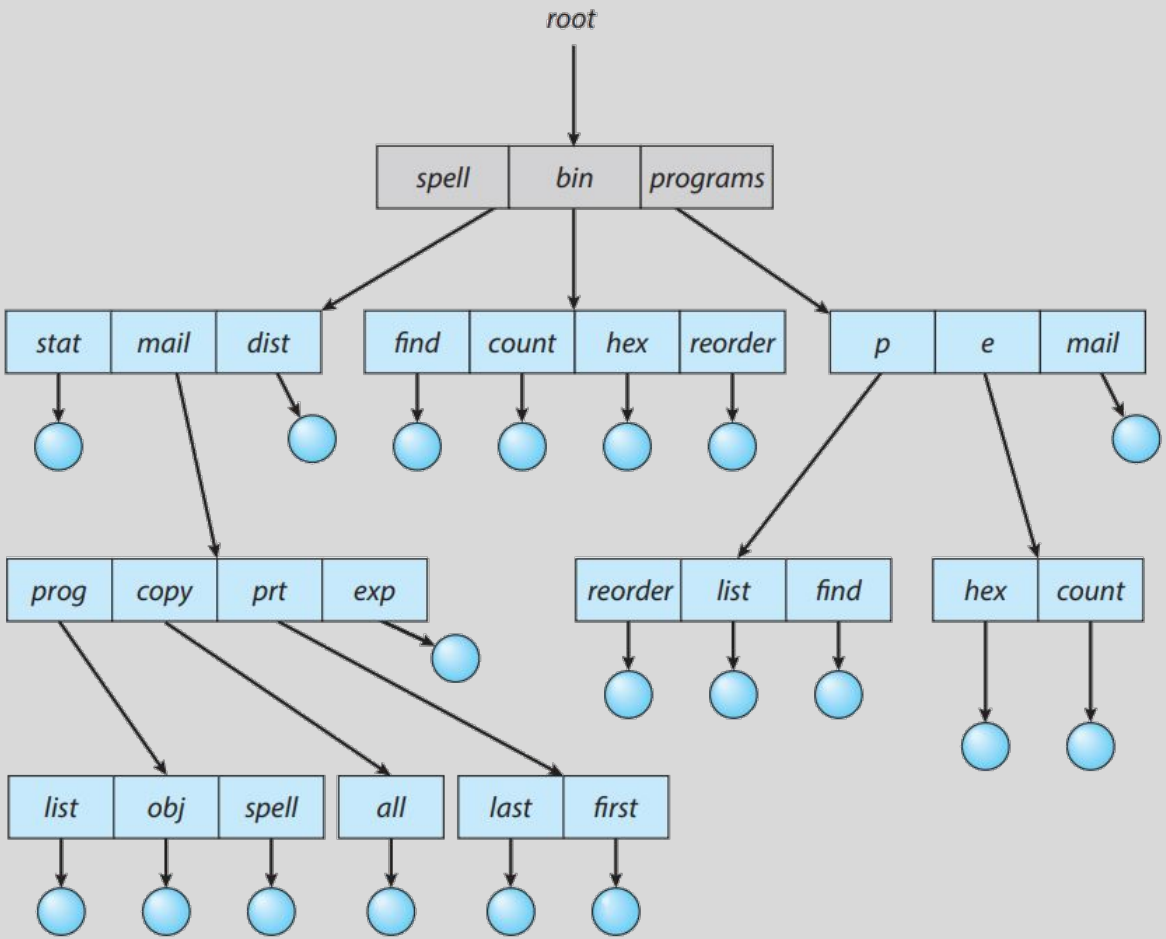
- Si el directorio está vacío, simplemente se elimina su entrada.
- Si contiene archivos o subdirectorios, algunos sistemas no permiten su eliminación hasta vaciarlo. Otros, como UNIX con el comando `` `rm -r` ``, permiten eliminar todo el contenido recursivamente, aunque esto conlleva el riesgo de perder muchos datos si se ejecuta por error (salvo que existan copias de seguridad).



Tree-structured directory structure.

- **Estructura de árbol**

Finalmente, una ventaja importante del modelo en árbol es que los usuarios pueden acceder a archivos de otros usuarios si tienen permiso, simplemente indicando la ruta correspondiente (absoluta o relativa). Esto habilita la cooperación y el acceso compartido sin perder estructura ni control de acceso.



Tree-structured directory structure.

PROTECCIÓN

PROTECCIÓN DE ARCHIVOS

La protección en los sistemas de archivos es necesaria porque, al poder acceder a los archivos, también se corre el riesgo de que ese acceso sea indebido o no autorizado. Por eso, es importante controlar qué usuarios pueden realizar qué acciones sobre cada archivo.

PROTECCIÓN DE ARCHIVOS

- **Tipos de Acceso**

El sistema debe controlar distintos tipos de operaciones posibles sobre un archivo. Los principales tipos de acceso que pueden ser permitidos o denegados son:

- Lectura: permite ver el contenido del archivo.
- Escritura: permite modificar o sobrescribir el contenido existente.
- Ejecución: permite ejecutar el archivo si contiene código.
- Agregar (*append*): permite añadir información nueva al final del archivo, sin modificar lo ya escrito.
- Eliminar: permite borrar el archivo del sistema y liberar el espacio que ocupaba.
- Listar: permite ver el nombre del archivo y sus atributos.
- Modificar atributos: permite cambiar propiedades del archivo (por ejemplo, su nombre, permisos, etc.).

Este control de acceso permite que los archivos estén protegidos y que no cualquiera pueda usarlos de manera inapropiada.

PROTECCIÓN DE ARCHIVOS

- **Control de Acceso**

El enfoque más común para controlar el acceso es asociarlo a la identidad del usuario. Es decir, los permisos que un usuario tiene sobre un archivo dependen de quién es.

Una forma de implementar esto es mediante una Lista de Control de Acceso o *Access Control List* (ACL). Donde cada archivo o directorio tiene su propia ACL, que indica qué usuarios pueden realizar qué acciones. Cuando alguien intenta acceder a un archivo, el sistema consulta su ACL: si el usuario tiene permiso para la operación deseada, se le concede; si no, se le niega.

- Problema con las ACL:

Cuando hay muchos usuarios, estas listas pueden volverse muy largas y difíciles de mantener. Por ejemplo, si queremos que todos puedan leer un archivo, deberíamos agregar a cada uno de los usuarios con permiso de lectura, lo que no es práctico ni escalable, especialmente si no conocemos todos los usuarios de antemano.

PROTECCIÓN DE ARCHIVOS

- **Clasificación de Usuarios: Propietario, Grupo y Otros**

Para simplificar este problema, muchos sistemas agrupan a los usuarios en tres categorías:

- Propietario: quién creó el archivo, tiene el control más directo sobre él.
- Grupo: conjunto de usuarios con intereses o funciones similares, que necesitan permisos parecidos sobre el archivo.
- Otros: todos los demás usuarios del sistema que no pertenecen al grupo ni son propietarios.

En lugar de listar usuario por usuario, se definen los permisos para cada una de estas tres categorías, lo que simplifica enormemente la gestión de accesos.

PROTECCIÓN DE ARCHIVOS

- **Ejemplo: Sistema de Permisos en UNIX**

El sistema UNIX (y otros similares) utiliza una combinación de ACLs y la clasificación anterior. Define los permisos mediante tres campos de tres bits cada uno, que se expresan como:

- `r` (read): permiso de lectura.
- `w` (write): permiso de escritura.
- `x` (execute): permiso de ejecución.

Cada archivo tiene entonces nueve bits de permisos: tres para el propietario, tres para el grupo, y tres para otros.

```
-rw-rw-r--   1 pbg   staff    31200  Sep 3 08:30   intro.ps
drwx------   5 pbg   staff      512  Jul 8 09.33   private/
drwxrwxr-x   2 pbg   staff      512  Jul 8 09:35   doc/
drwxrwx---   2 jwg   student    512  Aug 3 14:13   student-proj/
-rw-r--r--   1 pbg   staff     9423  Feb 24 2017   program.c
-rwxr-xr-x   1 pbg   staff    20471  Feb 24 2017   program
drwx--x--x   4 tag   faculty    512  Jul 31 10:31  lib/
drwx------   3 pbg   staff     1024  Aug 29 06:52  mail/
drwxrwxrwx   3 pbg   staff      512  Jul 8 09:35   test/
```

PROTECCIÓN DE ARCHIVOS

- **Ejemplo: Sistema de Permisos en UNIX**

Ejemplos:

- `rw-r--r--` significa que el propietario puede leer y escribir, el grupo solo puede leer, y el resto también solo puede leer.
- Para que un usuario pueda listar el contenido de un subdirectorio, necesita que el bit `r` esté activo en su categoría.
- Para entrar a un directorio (cambiarse a él con `cd`), necesita tener el bit `x` habilitado para su categoría.

Este esquema es compacto, eficiente y fácil de usar, y permite controlar el acceso de forma bastante precisa sin necesidad de listas extensas.

-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx------	5 pbg	staff	512	Jul 8 09.33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 jwg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2017	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2017	program
drwx--x--x	4 tag	faculty	512	Jul 31 10:31	lib/
drwx------	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/

ARCHIVOS MAPEADOS EN MEMORIA

ARCHIVOS MAPEADOS EN MEMORIA

Existe una forma alternativa y muy eficiente de trabajar con archivos, conocida como mapeo de archivos en memoria (*memory-mapped files*). Esta técnica permite que el acceso a un archivo se realice como si fuera acceso a memoria común, eliminando la necesidad de llamadas explícitas al sistema como `read()` o `write()` en cada operación.

La idea es la siguiente: normalmente, cuando un programa necesita leer o escribir en un archivo, se utilizan funciones del sistema operativo como `open()`, `read()` y `write()`. Cada una de estas funciones implica una llamada al sistema (lo cual tiene un costo) y un acceso físico al disco, que como sabemos, es muy lento comparado con el acceso a memoria.

Con los archivos mapeados en memoria, en cambio, se utiliza la memoria virtual para vincular directamente el archivo a una parte del espacio de direcciones del proceso. Esto significa que el programa puede trabajar con el archivo como si fuera una variable o un arreglo en RAM.

ARCHIVOS MAPEADOS EN MEMORIA

¿Qué es el mapeo?

Cuando se mapea un archivo, el sistema operativo asocia bloques del archivo con páginas de memoria virtual. Al intentar acceder por primera vez a una parte del archivo, se produce un fallo de página (porque aún no está en memoria). El sistema operativo responde cargando desde disco la parte del archivo necesaria y asignándola a una página física en memoria.

Después de esto, cualquier lectura o escritura al archivo se realiza directamente en memoria —es mucho más rápido que acceder al disco constantemente.

Ejemplo: si accedemos a la posición 1000 del archivo, y esa posición está mapeada en la página 3 de la memoria virtual, el acceso se comporta como si estuvieras leyendo o escribiendo en una variable en RAM. Así, las ventajas del mapeo son:

- Alto rendimiento: se eliminan muchas llamadas al sistema y accesos al disco.
- Simplicidad: se puede tratar al archivo como si fuera una estructura de datos en memoria.
- Integración con la memoria virtual: permite paginar el contenido del archivo de forma automática.

ARCHIVOS MAPEADOS EN MEMORIA

- Escrituras diferidas y sincronización

Cuando escribimos sobre un archivo mapeado en memoria, los cambios no se escriben inmediatamente en el archivo real del disco. Los sistemas suelen mantener los cambios en memoria hasta que pase alguno de estos eventos:

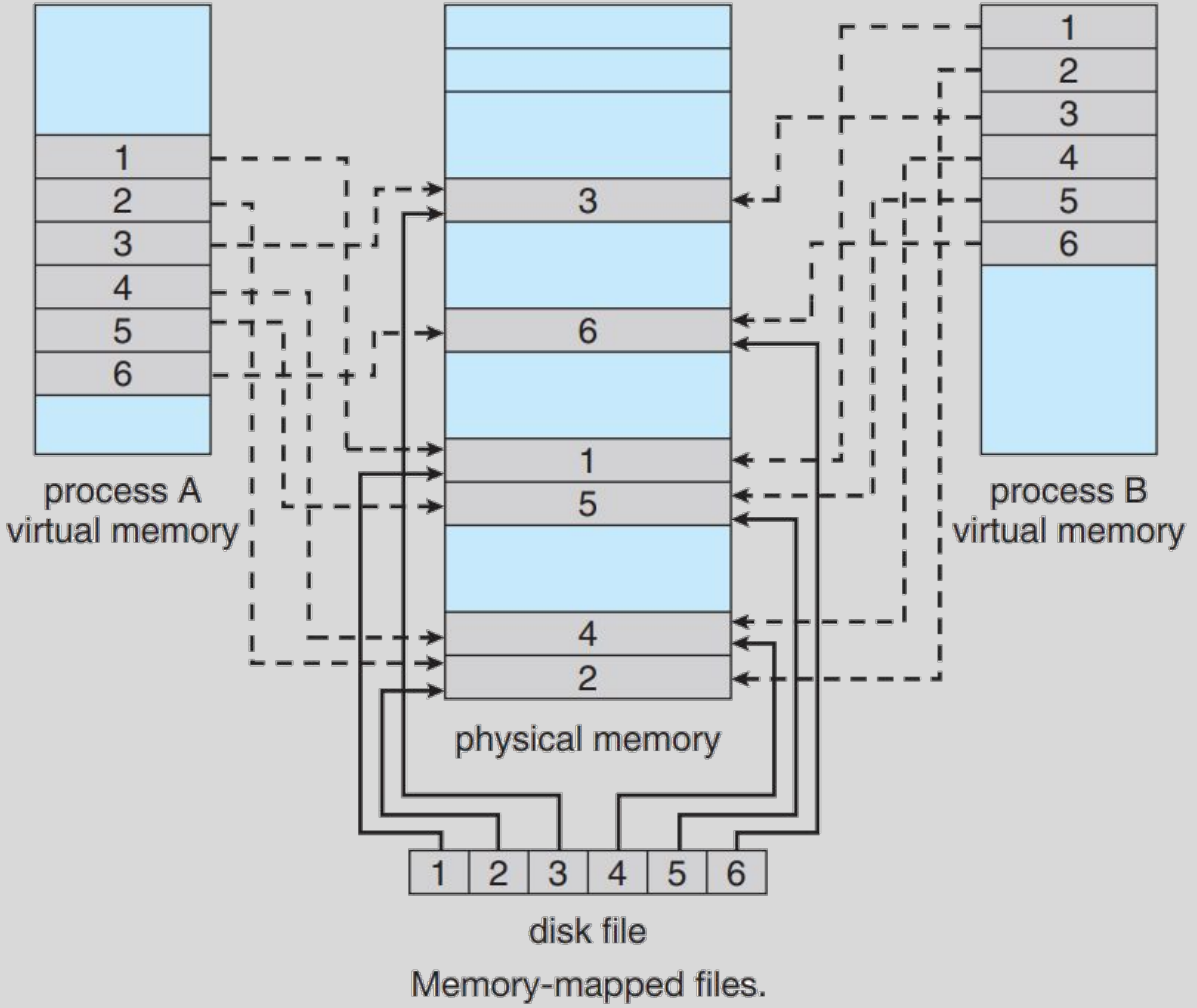
- Se cierra el archivo, momento en el cual los datos se sincronizan con el archivo en disco.
- O bien, si hay mucha carga de memoria, los datos se guardan en el espacio de intercambio (*swap*) para liberar RAM.

Esto significa que hay una ventana de tiempo en la que los cambios existen solo en la memoria. Si el sistema falla en ese momento, los cambios podrían perderse, a menos que se use una sincronización explícita (`msync()` en UNIX, por ejemplo).

- Archivos mapeados y múltiples procesos

Una ventaja clave de esta técnica es que varios procesos pueden mapear el mismo archivo al mismo tiempo y así **compartir datos**. En este caso, las modificaciones que hace un proceso en la memoria pueden ser vistas por los otros procesos, porque todos están apuntando a la misma página física, en memoria.

Esto funciona como una forma de memoria compartida, y es una de las formas más eficientes de implementar comunicación entre procesos (IPC).

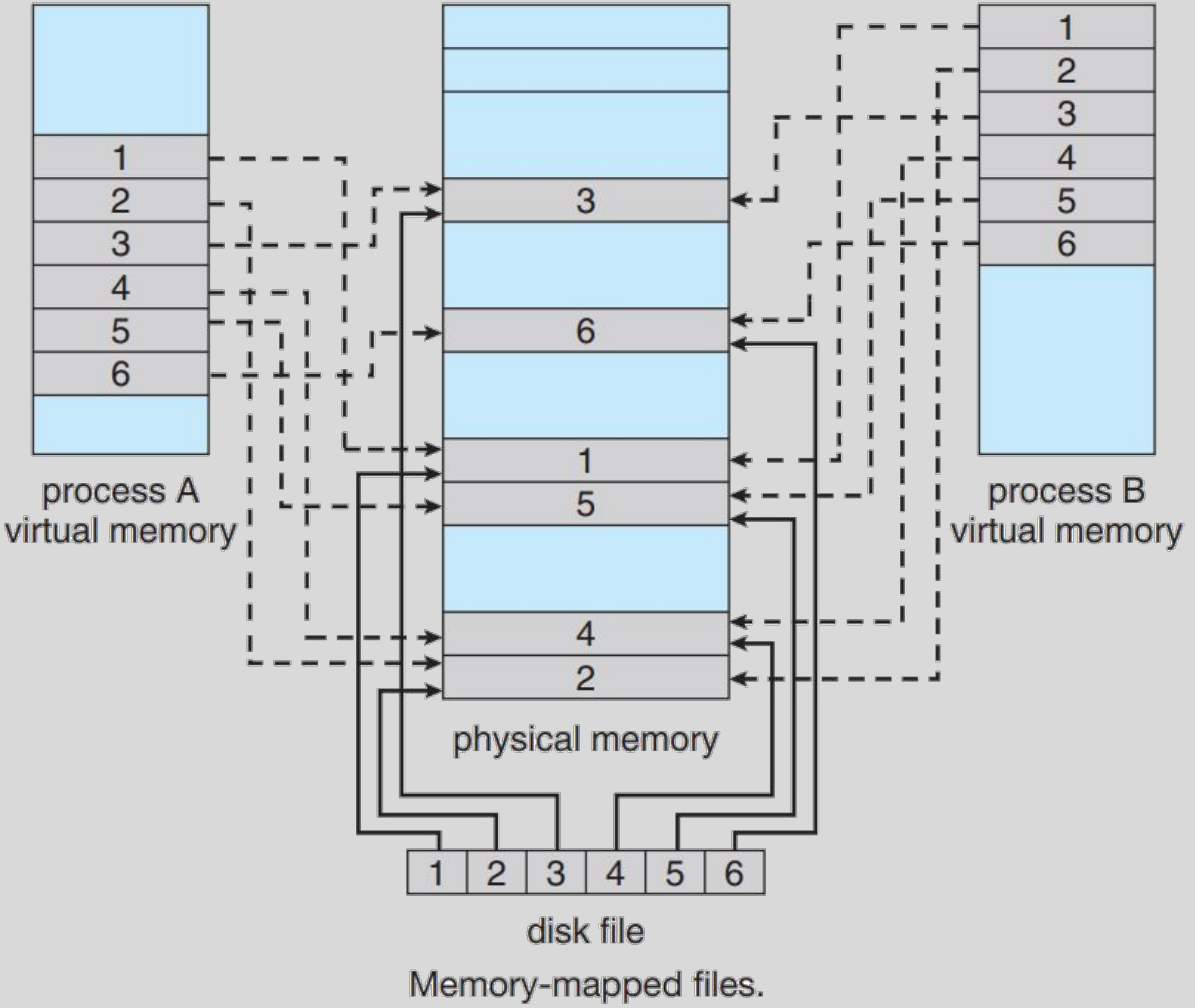


Memory-mapped files.

- Copia en escritura (Copy-on-write)

A veces, se permite que múltiples procesos accedan al mismo archivo en modo solo lectura, pero que puedan modificar datos de forma independiente. Esto se logra con una técnica llamada *copy-on-write*.

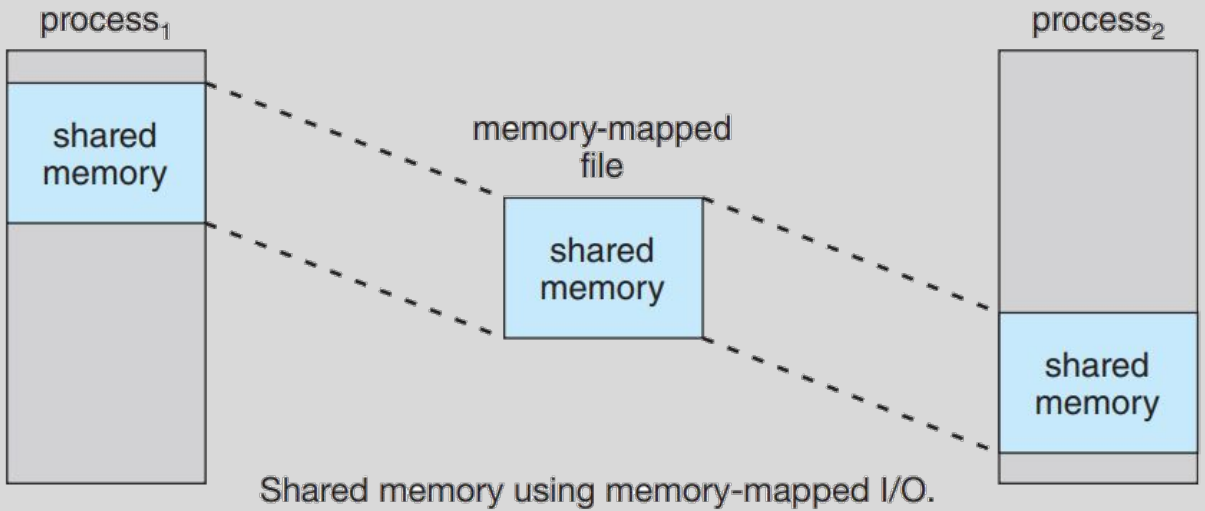
Cuando un proceso intenta escribir, se le da una copia privada de la página para que no afecte a los demás. Así se protege la integridad de los datos compartidos mientras se permite cierta flexibilidad.



Memory-mapped files.

Muchos sistemas operativos usan directamente esta técnica para implementar las regiones de memoria compartida entre procesos. Por ejemplo, si dos procesos mapean el mismo archivo en sus espacios de direcciones virtuales, pueden comunicarse escribiendo y leyendo desde esa región común de memoria.

Esto permite una forma simple y eficiente de comunicación entre procesos sin necesidad de pasar por el sistema de archivos constantemente.



Shared memory using memory-mapped I/O.

Muchas Gracias

Jeremías Fassi

Javier E. Kinter