

SISTEMAS OPERATIVOS: SISTEMAS DE FICHEROS



Ficheros, directorios y sistema de ficheros

A recordar...

Antes de clase

Clase

Después de clase

Preparar los pre-requisitos.

Estudiar el material asociado a la **bibliografía**:
las transparencias solo no son suficiente.
Preguntar dudas (especialmente tras estudio).

Ejercitar las competencias:

- ▶ Realizar todos los **ejercicios**.
- ▶ Realizar los **cuadernos de prácticas** y las **prácticas** de forma progresiva.

Lecturas recomendadas

Base



1. Carretero 2020:
 1. Cap. 6
2. Carretero 2007:
 1. Cap. 9.1-9.5,
 2. Cap. 9.8-9.10 y 9.12

Recomendada



1. Tanenbaum 2006:
 1. (es) Cap. 6
 2. (en) Cap. 6
2. Stallings 2005:
 1. 12.1-12.8
3. Silberschatz 2006:
 1. 10.3-10.4,
 2. 11.1-11.6 y 13

Contenidos

- **Introducción**
- **Fichero**
- Directorio
- Sistema de ficheros
- Particiones/Volúmenes
- Dispositivos
- Software de sistema

Contenidos

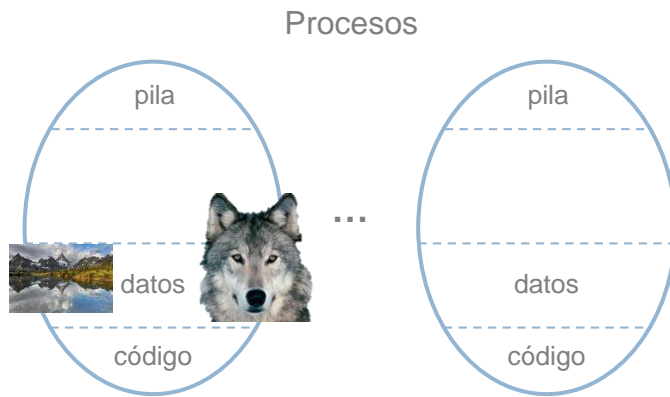
- **Introducción**
- **Fichero**
- Directorio
- Sistema de ficheros
- Particiones/Volúmenes
- Dispositivos
- Software de sistema

Introducción

6

<https://eskipaper.com/free-wolf-wallpaper-2.html>
<https://www.wallpaperbetter.com/es/hd-wallpaper-txaab>

Alejandro Calderón Mateos 



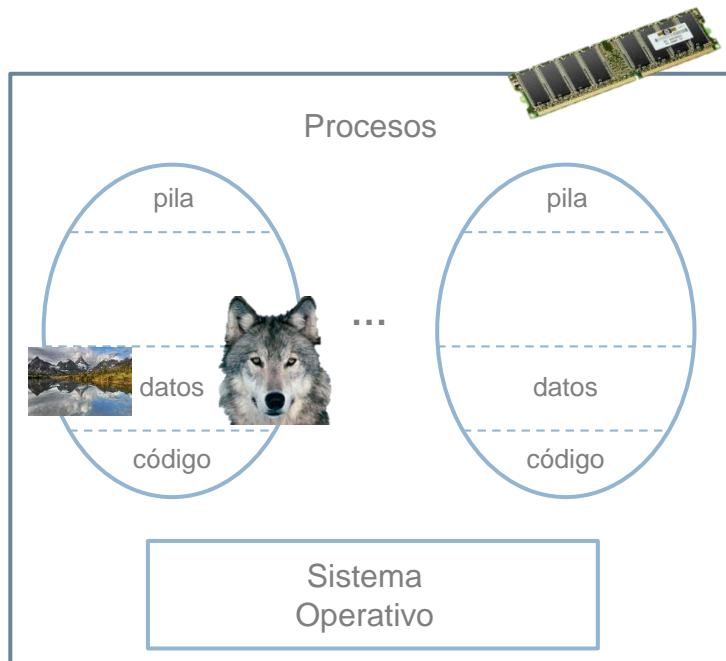
- Un proceso de edición de fotografías (por ejemplo) tiene en memoria su código y datos.
 - ▣ Cada proceso trabaja con sus datos, pudiendo generar nuevos datos.

Introducción

7

<https://eskipaper.com/free-wolf-wallpaper-2.html>
<https://www.wallpaperbetter.com/es/hd-wallpaper-txaab>

Alejandro Calderón Mateos 



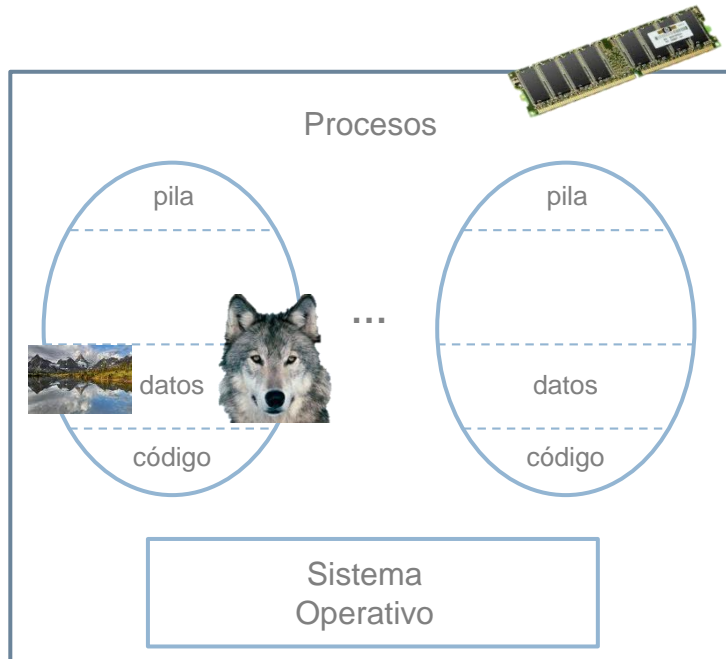
- Un proceso de edición de fotografías (por ejemplo) tiene en memoria su código y datos.
 - ▣ Cada proceso trabaja con sus datos, pudiendo generar nuevos datos.
- Puede haber varios procesos en memoria, siendo el sistema operativo el que reparte y organiza la memoria.

Introducción

8

<https://eskipaper.com/free-wolf-wallpaper-2.html>
<https://www.wallpaperbetter.com/es/hd-wallpaper-txaab>

Alejandro Calderón Mateos 



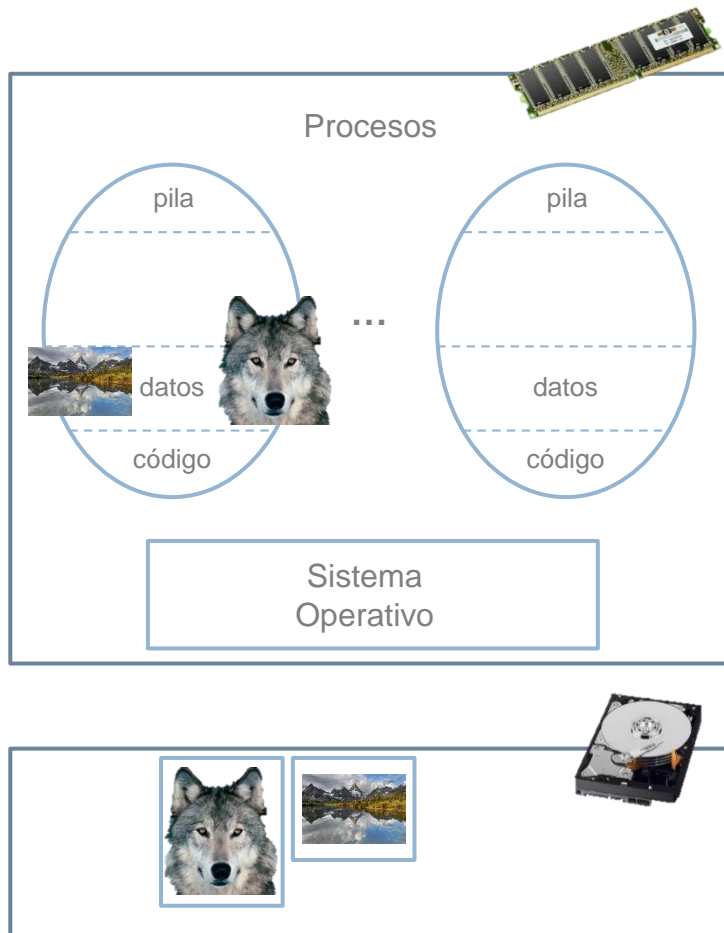
- La memoria principal en los sistemas actuales es de pequeño tamaño, acceso a palabra y volátil.
 - ▣ Los datos almacenados no son persistentes (sin electricidad).
 - ▣ Solo se usa para guardar los datos accedidos por el procesador durante un periodo.
 - ▣ Se puede acceder a cualquier palabra directamente.
- ¿Dónde guardar los datos?

Introducción

9

<https://eskipaper.com/free-wolf-wallpaper-2.html>
<https://www.wallpaperbetter.com/es/hd-wallpaper-txaab>

Alejandro Calderón Mateos 



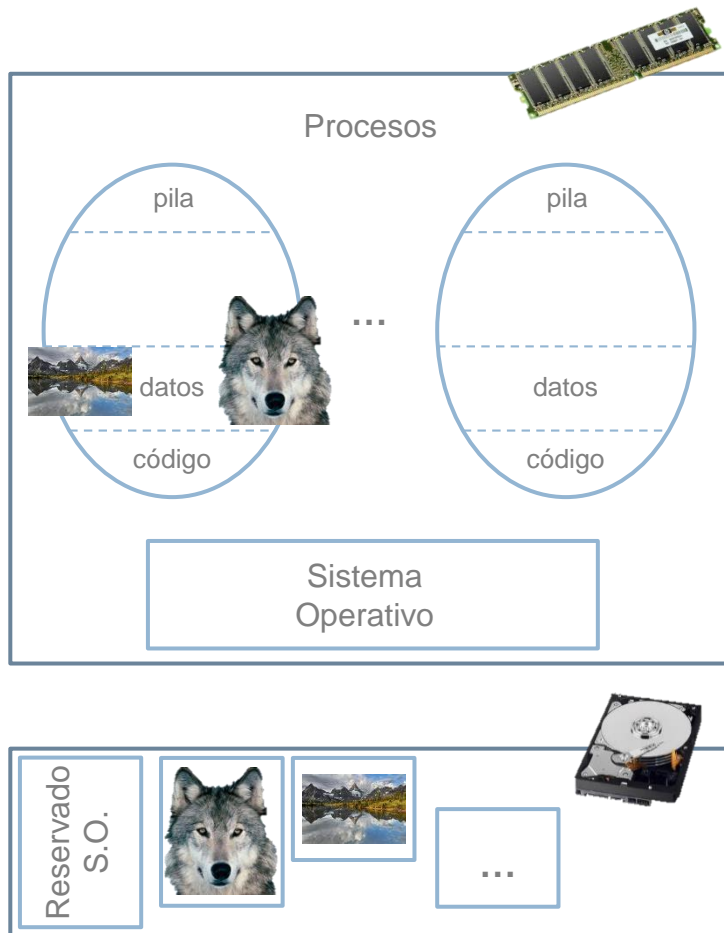
- La memoria secundaria es de mayor tamaño, acceso a bloque y no volátil.
 - ▣ Datos persistentes
 - Al proceso que lo usa, a la lectura concurrente entre procesos.
 - ▣ Permitirá guardar mayor cantidad de datos que en M.P.
 - ▣ Organizada en bloques, lo que supone tener que gestionar el uso de estos bloques.
- Los datos se guardarán en M.S.: disco duro, flash, etc..

Introducción

10

<https://eskipaper.com/free-wolf-wallpaper-2.html>
<https://www.wallpaperbetter.com/es/hd-wallpaper-txaab>

Alejandro Calderón Mateos 



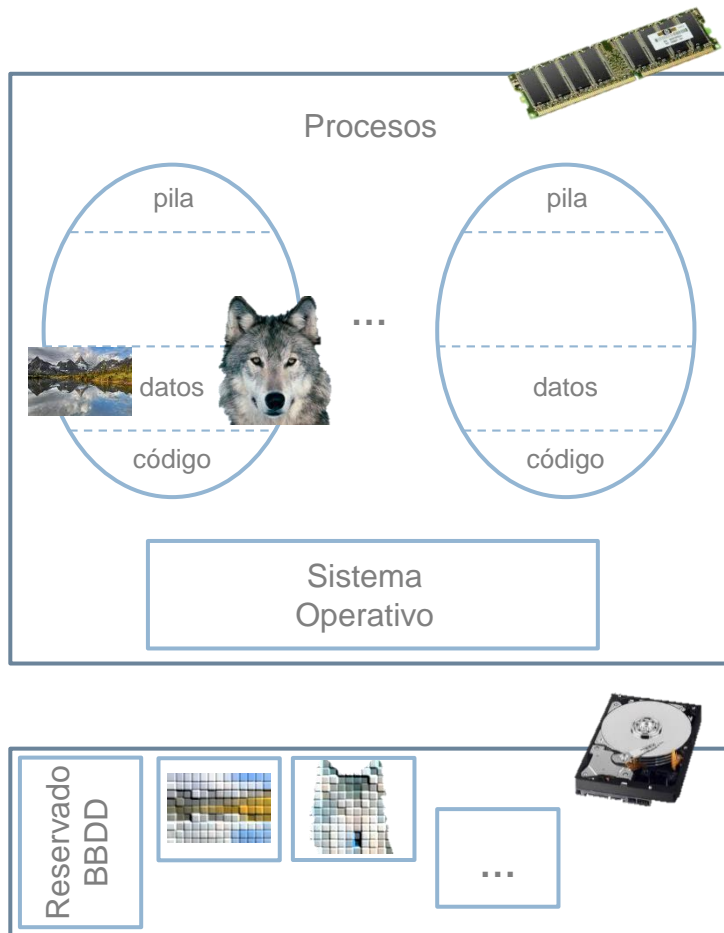
- Parte del sistema operativo que se encarga de repartir y organizar la M.S.
 - ▣ Sistema de ficheros.
- El sistema de ficheros ofrece servicios para almacenar y recuperar los datos de forma simple
 - ▣ Oculta los detalles de la organización de la M.S. mediante abstracciones: ficheros, directorios, etc.

Introducción

11

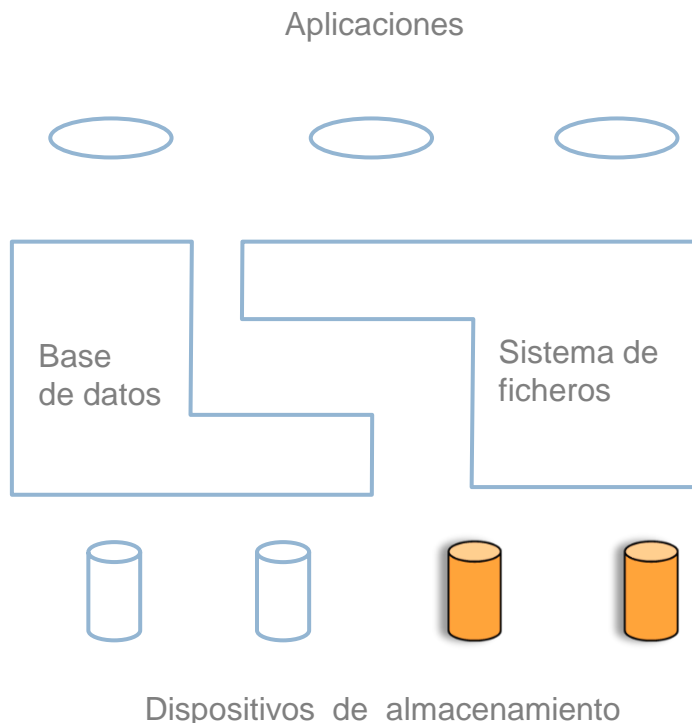
<https://eskipaper.com/free-wolf-wallpaper-2.html>
<https://www.wallpaperbetter.com/es/hd-wallpaper-txaab>

Alejandro Calderón Mateos 



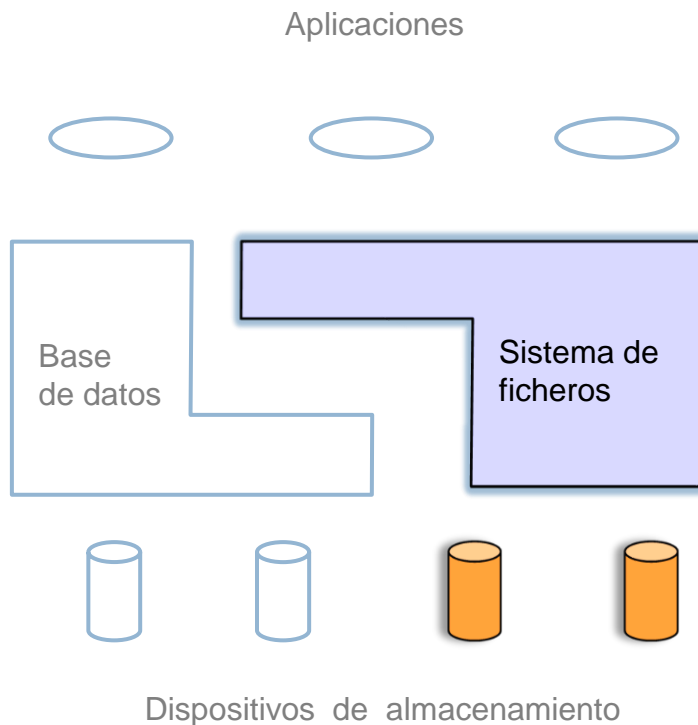
- Pero también es posible que ciertas aplicaciones organicen la M.S.:
 - ▣ Gestores de bases de datos.
- El sistema operativo ofrece acceso a todo el dispositivo.
- Es posible también una organización mixta
 - ▣ Parte el sistema operativo y parte la aplicación

Resumen: arquitectura



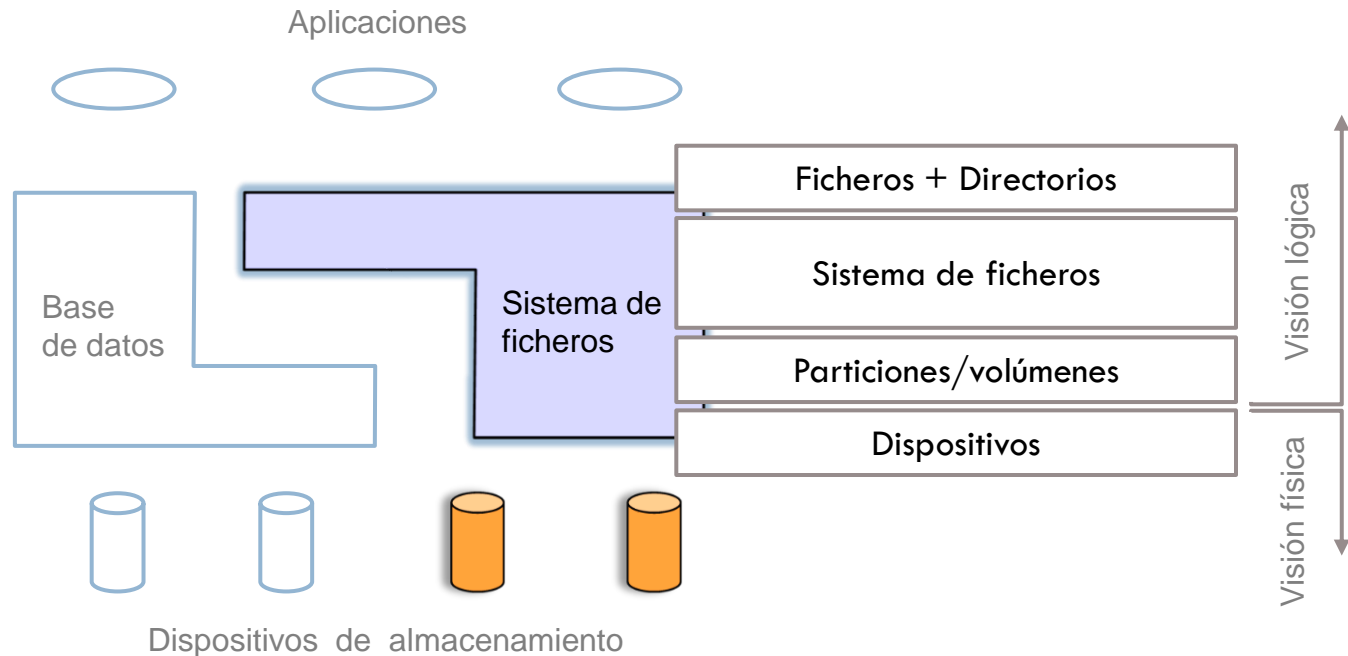
- Tenemos ambas posibilidades en ilustración que propone la SNIA:
 - ▣ *Storage Networking Industry Association*
 - ▣ <http://www.snia.org>
- Las aplicaciones acceden a los datos guardados en dispositivos de almacenamiento usando BBDD y/o sistemas de ficheros.

Resumen: arquitectura



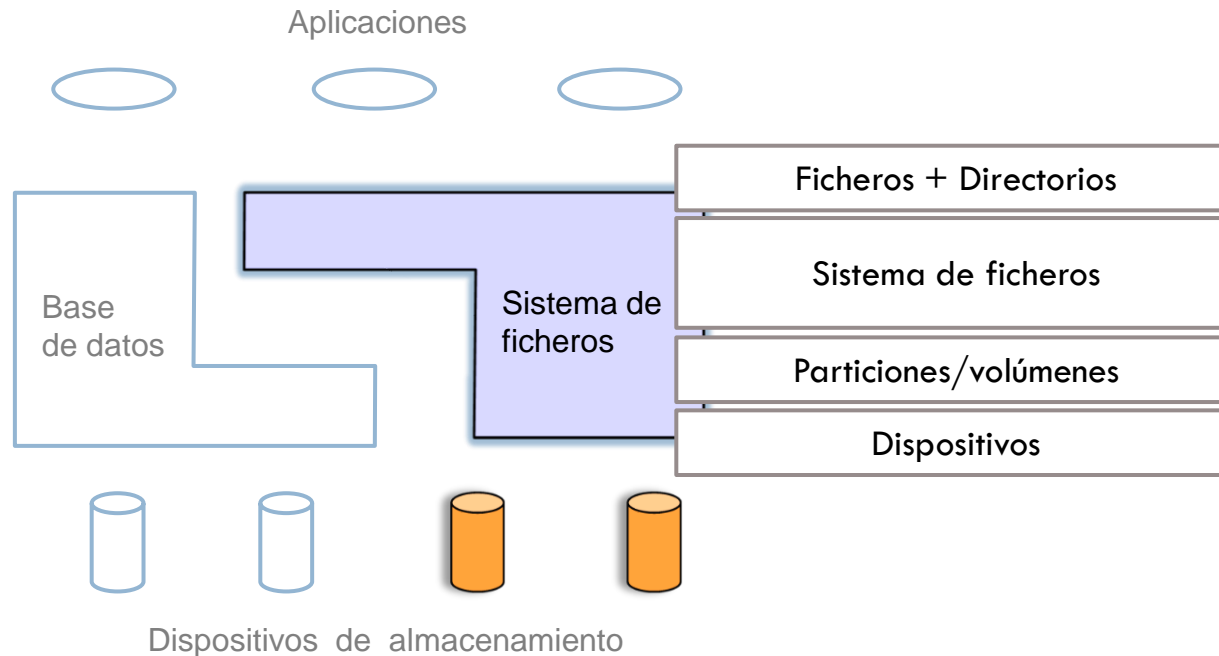
- En este tema nos centraremos en la gestión mediante el S.O. a través del sistema de ficheros:
 - ▣ Organización
 - ▣ Almacenamiento
 - ▣ Recuperación
 - ▣ Gestión de nombres
 - ▣ Implementación de la semántica de utilización
 - ▣ Protección

Resumen: abstracciones



- A estudiar: ficheros, directorios, sistema de ficheros, volúmenes y dispositivos
Visión lógica Visión física

Resumen: abstracciones



- ❑ Cuidado con el término “**sistema de ficheros**” que es usado para nombrar tanto al **software gestor** como para las **estructuras de datos en disco**

Introducción

resumen

importante

□ Sistema de ficheros:

- Es la parte del SO encargada de repartir y organizar la M.S.
- Proporciona una abstracción (basada en ficheros, directorios, etc.) que oculta los detalles de la organización de la M.S.
 - Oculta detalles sobre almacenamiento/distribución de datos en los periféricos.
- Funciones principales:
 - (1) Organización, (2) Gestión de nombres, (3) Almacenamiento, (4) Recuperación, (5) Implementación de la semántica de coutilización, (6) Protección

□ Sistema de ficheros también:

- Es la capa de software entre dispositivos y usuarios/as.
- Simplifica el manejo de periféricos tratándolos como ficheros
 - Establece una correspondencia entre dispositivos lógicos y ficheros.
 - Facilita protección y visión lógica (lo más independiente de detalles físicos).

Introducción

resumen

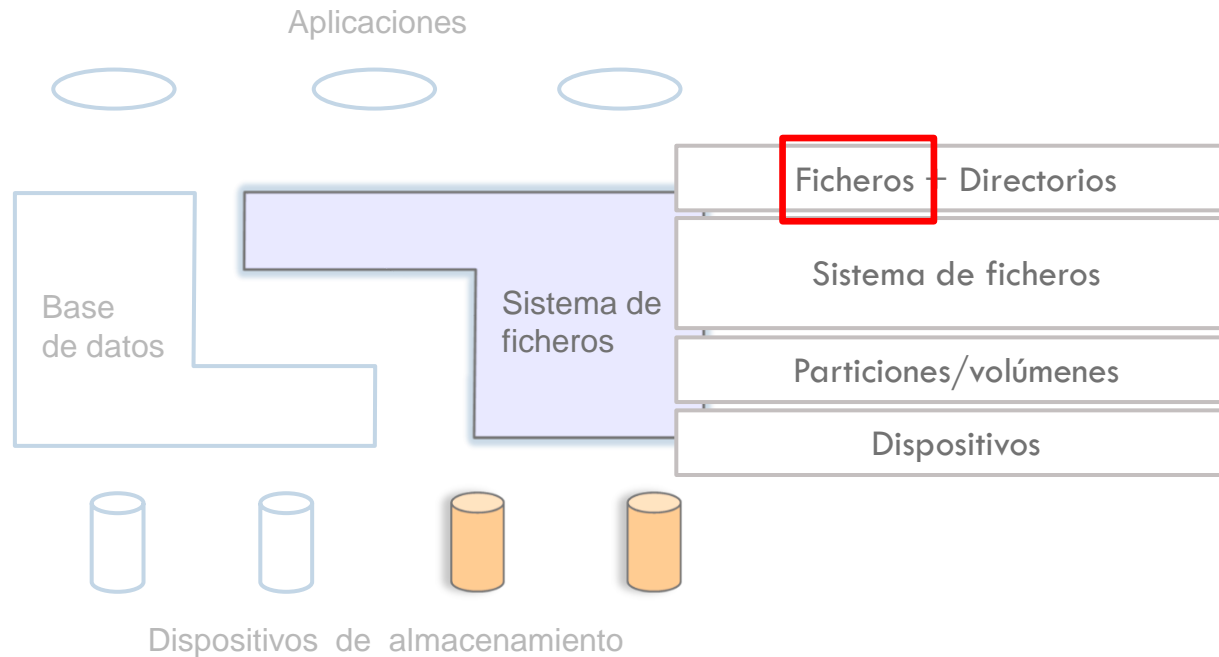
importante

- Sistema de ficheros para el usuario:
 - Almacenamiento permanentes de información:
 - No desaparece aunque se apague el computador.
 - Abstracción lógica para facilitar el manejo de información:
 - Conjunto de información estructurada de forma lógica según criterios de aplicación.
 - Nombres lógicos y estructurados.
 - No están ligados al ciclo de vida de una aplicación particular.
 - Abstraen los dispositivos de almacenamiento físico.
 - Acceso a los servicios ofrecidos a través de un API:
 - Se acceden a través de llamadas al sistema operativo o de bibliotecas de utilidades.
 - Se puede trabajar con varios sistemas de ficheros a la vez en un SO:
 - Ej: Linux admite a la vez ext2, btrfs, fat32, etc.

Contenidos

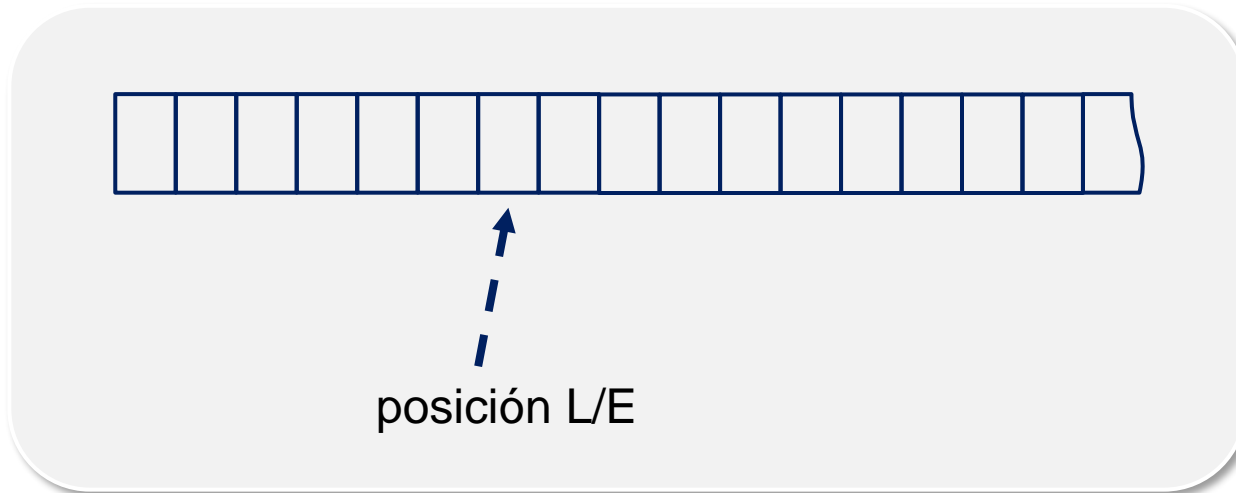
- Introducción
- **Fichero**
 - ▣ Metadatos
 - ▣ Interfaz
 - ▣ Métodos de acceso
 - ▣ Semántica de compartición
- Directorio
- Sistema de ficheros
- Particiones/Volúmenes
- Dispositivos
- Software de sistema

Fichero o archivo



Fichero o archivo (visión lógica)

- Conjunto de información relacionada que ha sido definida por su creador.
- Habitualmente el contenido es representado por una secuencia o tira de bytes (UNIX, POSIX):



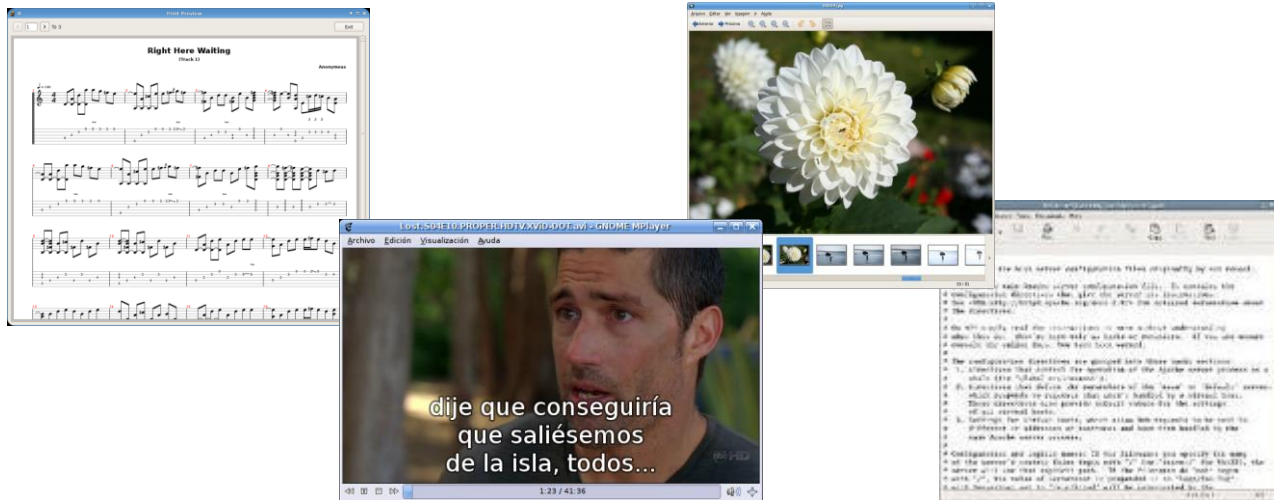
Fichero o archivo

21

Alejandro Calderón Mateos



□ Diferentes tipos de información:



Fichero o archivo (estructura)

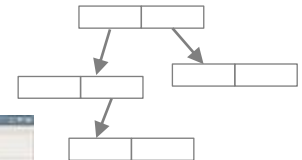
22

Alejandro Calderón Mateos

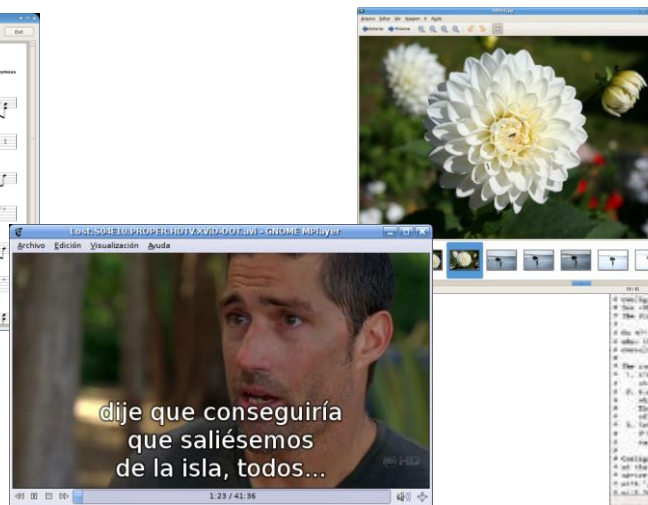
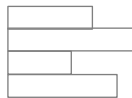


□ Diferentes tipos de estructuras de esa información:

- Complejos
 - Formato (XML, etc.)
 - Reubicables



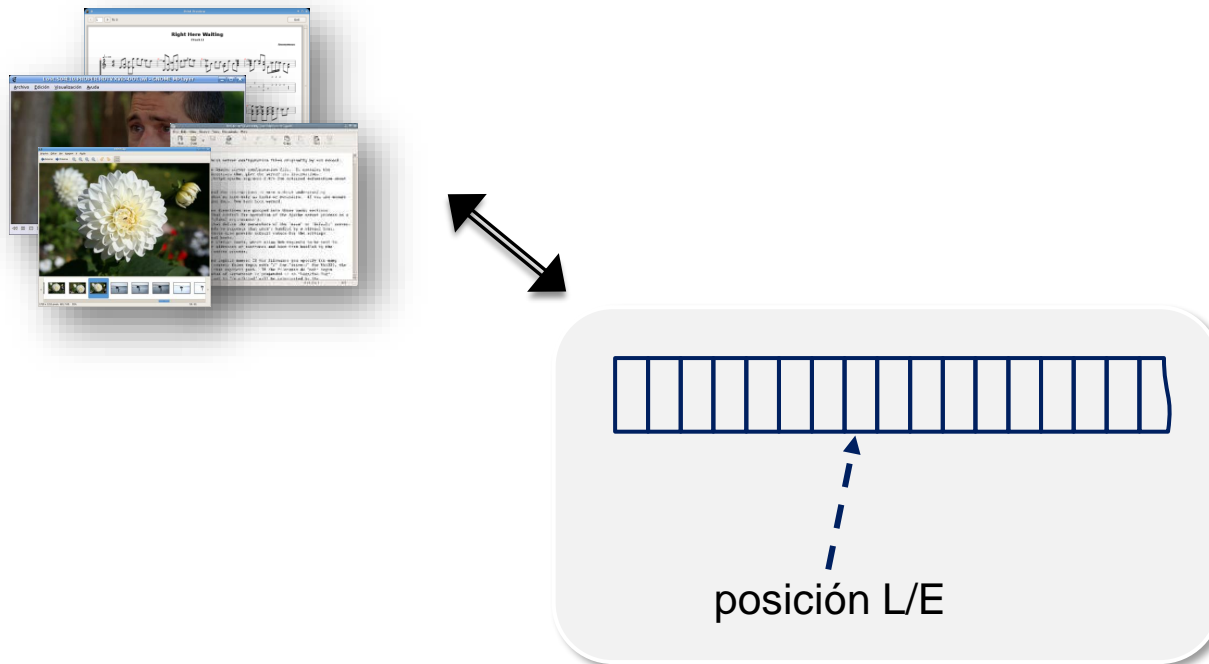
- Registros
 - Longitud fija
 - Longitud variable



- Secuencia de palabras

Fichero o archivo

- Las aplicaciones convierten y almacenan como una **secuencia o tira de bytes**:



Contenidos

- Introducción
- **Fichero**
 - ▣ **Metadatos**
 - ▣ Interfaz
 - ▣ Métodos de acceso
 - ▣ Semántica de compartición
- Directorio
- Sistema de ficheros
- Particiones/Volúmenes
- Dispositivos
- Software de sistema

Fichero o archivo

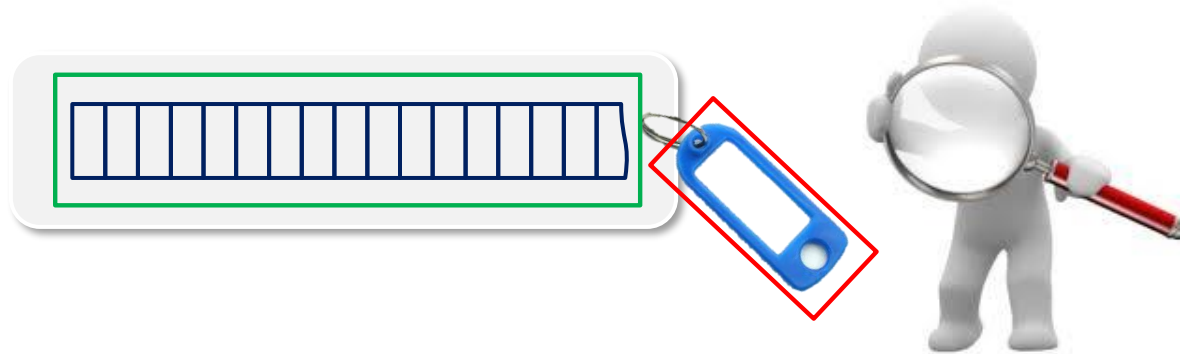
□ Información de un archivo:

□ Datos

- Información que almacena el archivo.

□ Metadatos

- Información sobre el archivo.
- Distintos **atributos** sobre el archivo (+ información usada por el S.O.)
- Dependiente del sistema de ficheros.



Fichero o archivo

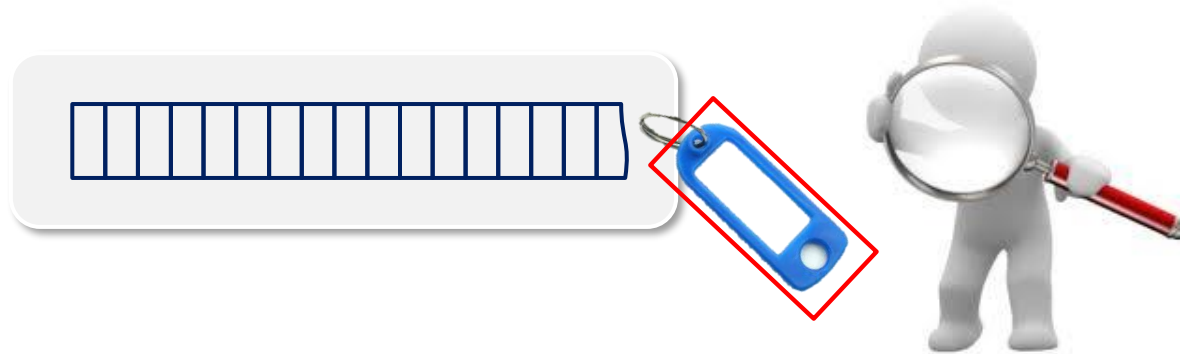
□ Información de un archivo:

□ Datos

- Información que almacena el archivo.

□ Metadatos

- Información sobre el archivo.
- Distintos **atributos** sobre el archivo (+ información usada por el S.O.)
- Dependiente del sistema de ficheros.





Fichero o archivo: atributos

□ Atributos típicos de un fichero:

- **Nombre:** identificador para los usuarios del fichero.
- **Identificador:** etiqueta unívoca del archivo (numérico) usada por el S.O.
- **Tipo:** tipo de archivo (para los sistemas que lo necesiten)
 - Ej.: extensión (.exe, .pdf, etc.)
- **Ubicación:** identificador que ayuda a la localización de los bloques del dispositivo que pertenecen al archivo.
- **Tamaño:** tamaño actual del fichero (en bytes o bloques de disco).
- **Protección:** control de acceso y operaciones qué usuario puede hacer.
- **Información temporal:** instante de tiempo de último acceso, de creación, etc. que permite la monitorización del uso del archivo.
- **Identificación de usuario:** identificador del creador, dueño del archivo, etc.

Nombre de fichero (y extensión)

- Se utiliza tiras de caracteres:
 - ▣ Permite a los usuarios organizarse mejor.
 - ▣ Los usuarios no recuerdan nombres del tipo 00112233.
 - ▣ Directorios relacionan nombre con su identificador interno.
- Es característico de cada sistema de ficheros:
 - ▣ Longitud del nombre: fijo (MS-DOS) o variable (UNIX)
 - ▣ Sensibles a mayúsculas/minúsculas (Unix) o no (MS-DOS)
 - INMA e inma
 - ▣ Necesario extensión: si y fija (MS-DOS), no (UNIX)
 -  remain.zip .zip -> identifica el tipo de fichero (y la aplicación a usar)
 -  /bin/ls file nombre -> identifica por contenido (número mágico)

Control de acceso

- Listas de control de acceso (ACL):
 - ▣ ACL es una lista asociada a un fichero que está formada por entradas ACE en la forma de (usuario/grupo, permiso).
 - ▣ Ej.: NTFS, UFS de Solaris, HFS de HP-UX, etc.
 - ▣ En Linux: `setfacl -d -m g:desarrollo:rw /home/devs`
- Permisos:
 - ▣ Versión condensada usada en UNIX tradicional.
 - ▣ 3 categorías: usuario, grupo, otros.
 - ▣ 3 tipos de acceso por categoría: `read`, `write`, `execute`.

Contenidos

- Introducción
- **Fichero**
 - Metadatos
 - **Interfaz**
 - Métodos de acceso
 - Semántica de compartición
- Directorio
- Sistema de ficheros
- Particiones/Volúmenes
- Dispositivos
- Software de sistema

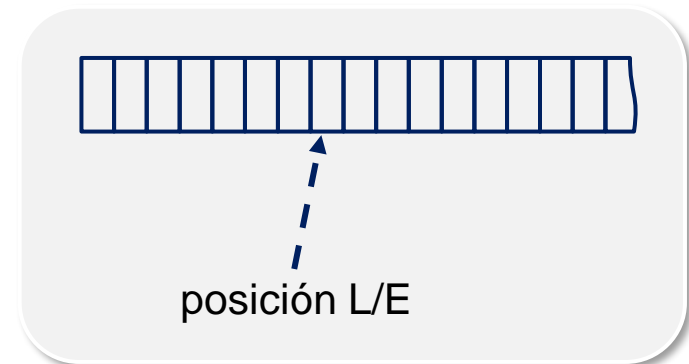
Fichero o archivo: interfaz

importante

31

Alejandro Calderón Mateos 

- **Interfaz genérica** para acceder a la información:
 - ▣ `descriptor` ← `open` (nombre, flags, modo)
 - ▣ `close` (descriptor)
 - ▣ `read` (descriptor, puntero, tamaño)
 - ▣ `write` (descriptor, puntero, tamaño)
 - ▣ `lseek` (descriptor, desplazamiento, origen)
 - ▣ `ioctl` (descriptor, operación, puntero_a_parámetros)



OPEN – Apertura de fichero

Servicio	<pre>#include <sys/types.h> #include <sys/stat.h> #include <fcntl.h> int open(char *pathname, int flags[, mode_t mode]);</pre>
Argumentos	<ul style="list-style-type: none">□ pathname nombre del fichero (puntero al primer caracter).□ flags opciones de apertura:<ul style="list-style-type: none">■ O_RDONLY Sólo lectura■ O_WRONLY Sólo escritura■ O_RDWR Lectura y escritura■ O_APPEND Posicionar el puntero de acceso al final del fichero abierto■ O_CREAT Si existe no tiene efecto. Si no existe lo crea■ O_TRUNC Trunca si se abre para escritura□ mode permisos:<ul style="list-style-type: none">■ S_I{RWX}{USR GRP,OTH} Lectura, Escritura, Ejecución x usuario, grupo, otros
Devuelve	Un descriptor de fichero ó -1 si hay error.
Descripción	Apertura de fichero (o creación si se usa O_CREAT).

CLOSE – Cierre de fichero

Servicio	<pre>#include <unistd.h> int close(int fd);</pre>
Argumentos	fd descriptor de fichero.
Devuelve	Devuelve 0 ó -1 si error.
Descripción	El proceso cierra la session de trabajo con el fichero, y el descriptor pasa a estar libre.

READ – Lectura de fichero

Servicio	<pre>#include <sys/types.h> ssize_t read(int fd, void *buf, size_t n_bytes);</pre>
Argumentos	<ul style="list-style-type: none">▣ <code>fd</code> descriptor de fichero▣ <code>buf</code> zona donde almacenar los datos▣ <code>n_bytes</code> número de bytes a leer
Devuelve	Número de bytes realmente leídos, 0 si fin de fichero (EOF) y -1 si error.
Descripción	<ul style="list-style-type: none">▣ Intenta leer <code>n_bytes</code>. Puede leer menos datos de los solicitados (ej.: si se rebasa el fin de fichero o se interrumpe por una señal).▣ Después de la lectura se actualiza el puntero de posición del fichero (file pointer) con el número de bytes realmente leídos.

WRITE – Escritura de fichero

Servicio	<pre>#include <sys/types.h> ssize_t write(int fd, void *buf, size_t n_bytes);</pre>
Argumentos	<ul style="list-style-type: none">❑ <code>fd</code> descriptor de fichero❑ <code>buf</code> zona de datos a escribir❑ <code>n_bytes</code> número de bytes a escribir
Devuelve	Número de bytes realmente escritos ó -1 si error.
Descripción	<ul style="list-style-type: none">❑ Intenta escribir <code>n_bytes</code>. Puede escribir menos datos de los solicitados (ej.: si se rebasa el tamaño máximo de un fichero o se interrumpe por una señal).❑ Después de la escritura se actualiza el puntero de posición del fichero (file pointer) con el número de bytes realmente escritos.❑ Si se rebasa el fin de fichero el fichero aumenta de tamaño.

LSEEK – Movimiento del puntero de posición

Servicio	<pre>#include <sys/types.h> #include <unistd.h> off_t lseek(int fd, off_t offset, int whence);</pre>
Argumentos	<ul style="list-style-type: none">❑ <code>fd</code> descriptor de fichero❑ <code>offset</code> desplazamiento (en bytes, positivo o negativo)❑ <code>whence</code> base del desplazamiento
Devuelve	<ul style="list-style-type: none">❑ La nueva posición del puntero ó -1 si error.❑ Ejemplo: <code>lseek(fd, 55, SEEK_SET)</code> devolvería 55 si no hay error.
Descripción	<ul style="list-style-type: none">❑ Modifica el puntero de lectura/escritura asociado a <code>fd</code>❑ La nueva posición se calcula:<ul style="list-style-type: none">■ <code>SEEK_SET</code> -> posición = offset■ <code>SEEK_CUR</code> -> posición = posición actual + offset■ <code>SEEK_END</code> -> posición = tamaño del fichero + offset❑ Saltar más allá del final solo se consolida si se escribe.

LINK – Creación de enlace

Servicio	<pre>#include <unistd.h> int link (const char* oldpath, const char* newpath);</pre>
Argumentos	<ul style="list-style-type: none">❑ <code>oldpath</code> nombre del fichero existente a enlazar.❑ <code>newpath</code> nombre del enlace a crear.
Devuelve	Devuelve 0 ó -1 si error.
Descripción	<ul style="list-style-type: none">❑ Crea un enlace duro (<i>hard link</i>) de un entrada (archivo o directorio) existente en la misma partición de la entrada enlazada.❑ Incrementa el contador de enlaces del fichero.

UNLINK – Borrado de fichero

Servicio	<pre>#include <unistd.h> int unlink (const char* path);</pre>
Argumentos	path nombre del fichero
Devuelve	Devuelve 0 si todo correcto ó -1 si error.
Descripción	<ul style="list-style-type: none">▣ Decrementa el contador de enlaces del fichero.▣ Si el contador es 0 entonces:<ul style="list-style-type: none">▣ Si no está abierto entonces borra el fichero y libera sus recursos.▣ Si está abierto, al cerrar se borrará y liberará sus recursos.

Fichero o archivo: interfaz POSIX

importante

39

Alejandro Calderón Mateos 

escritura

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>

int main ( int argc, char *argv[] )
{
    int fd1 ;
    char str1[10] ;
    int nb ;

    fd1 = open ("/tmp/txt1",
                O_CREAT|O_RDWR, S_IRWXU);
    if (-1 == fd1) {
        perror("open:");
        exit(-1);
    }

    strcpy(str1,"hola");
    nb = write (fd1,str1,strlen(str1));
    printf("bytes escritos = %d\n",nb);

    close (fd1);
    return (0) ;
}
```

lectura

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>

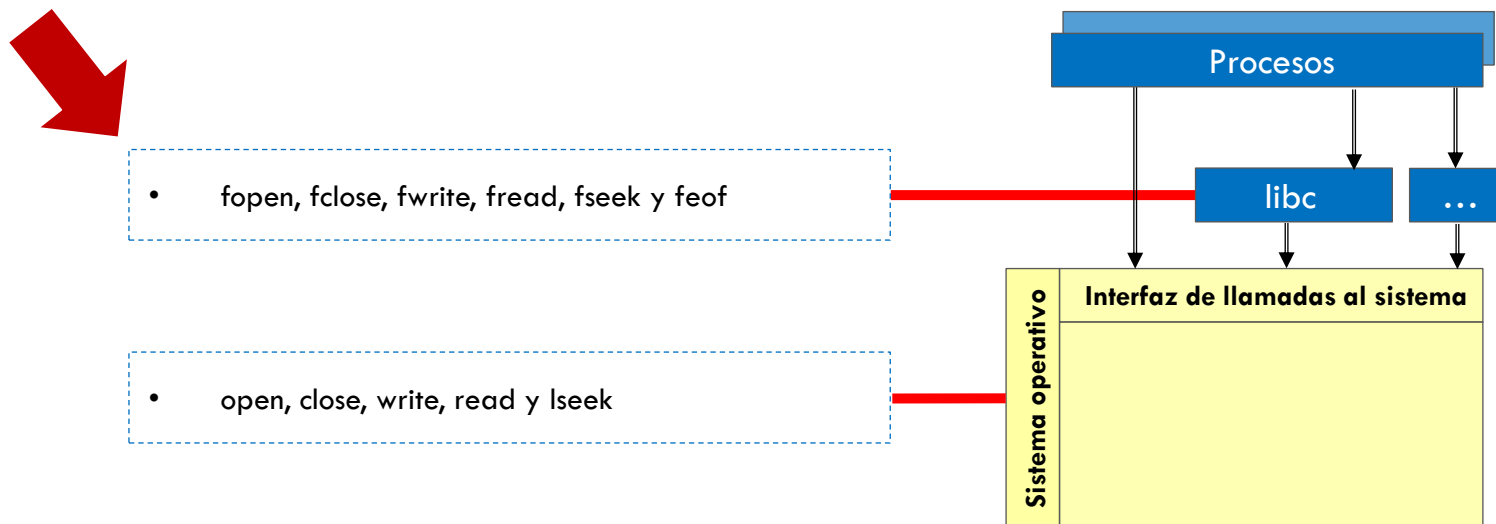
int main ( int argc, char *argv[] )
{
    int fd1 ;
    char str1[10] ;
    int nb, i ;

    fd1 = open ("/tmp/txt1",O_RDONLY);
    if (-1 == fd1) {
        perror("open:");
        exit(-1);
    }

    i=0;
    do {
        nb = read (fd1,&(str1[i]),1); i++;
    } while (nb != 0) ;
    str1[i] = '\0';
    printf("%s\n",str1);

    close (fd1);
    return (0);
}
```

Fichero o archivo: interfaz C99



Fichero o archivo: interfaz C99

41

Alejandro Calderón Mateos 

escritura

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main ( int argc, char *argv[] )
{
    FILE *fd1 ;
    char str1[10] ;
    int nb ;

    fd1 = fopen ("/tmp/txt2","w+");
    if (NULL == fd1) {
        printf("fopen: error\n");
        exit(-1) ;
    }

    strcpy(str1,"mundo");
    nb = fwrite (str1,strlen(str1),1,fd1);
    printf("items escritos = %d\n",nb);

    fclose (fd1) ;
    return (0) ;
}
```

lectura

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main ( int argc, char *argv[] )
{
    FILE *fd1 ;
    char str1[10] ;
    int nb, i ;

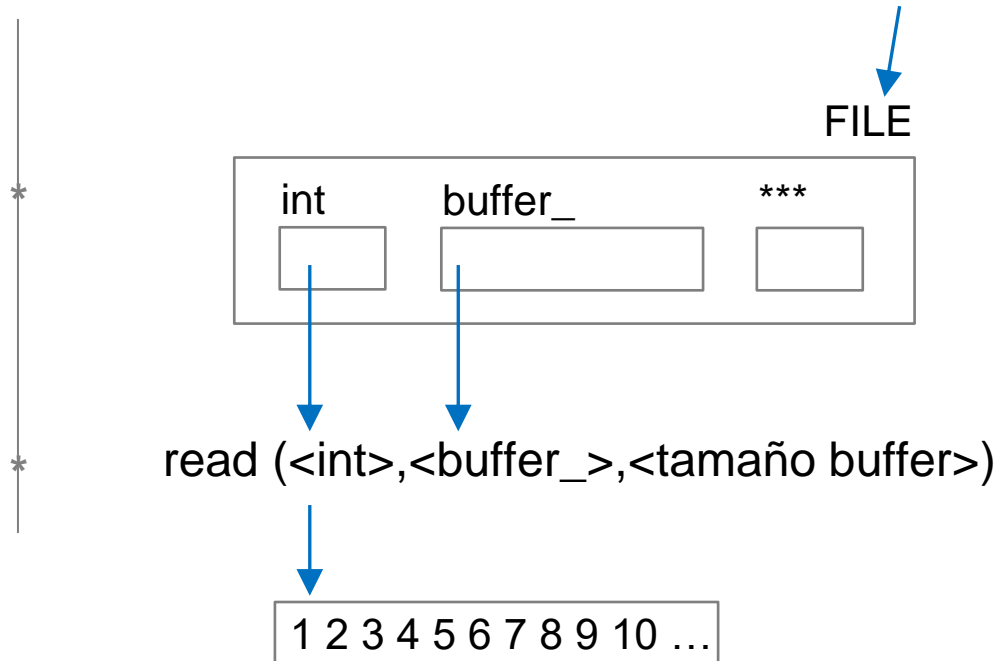
    fd1 = fopen ("/tmp/txt2","r");
    if (NULL == fd1) {
        printf("fopen: error\n");
        exit(-1) ;
    }

    i=0;
    do {
        nb = fread (&(str1[i]),1,1,fd1) ;
        i++ ;
    } while (nb != 0) ; /* feof() */
    str1[i] = '\0' ;
    printf("%s\n",str1);

    fclose (fd1);
    return (0);
}
```

Fichero o archivo: interfaz C99

fread (<buffer>,<tamaño 1 elto>,<nº eltos>,<FILE *>)

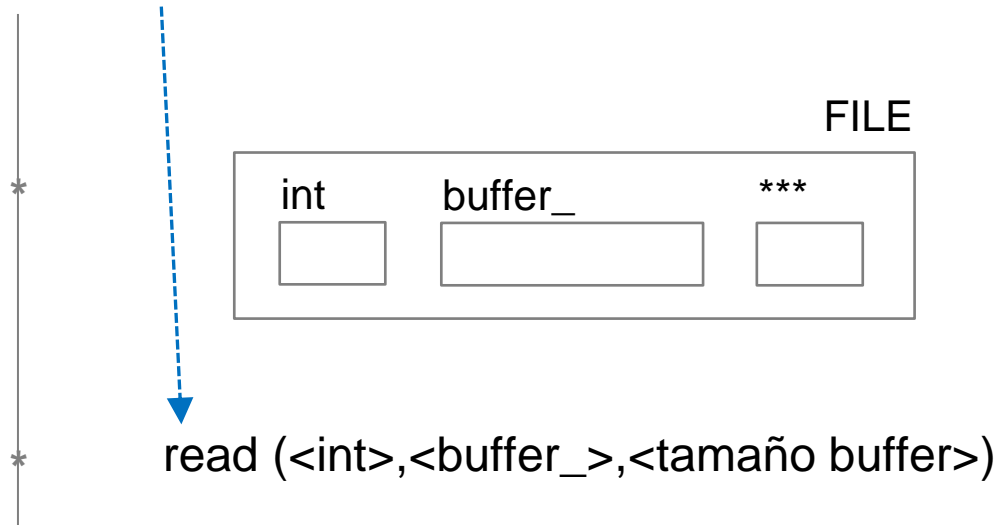


Fichero o archivo: interfaz C99

43

Alejandro Calderón Mateos 

`fread (<buffer>,<tamaño 1 elto>,<nº eltos>,<FILE *>)`



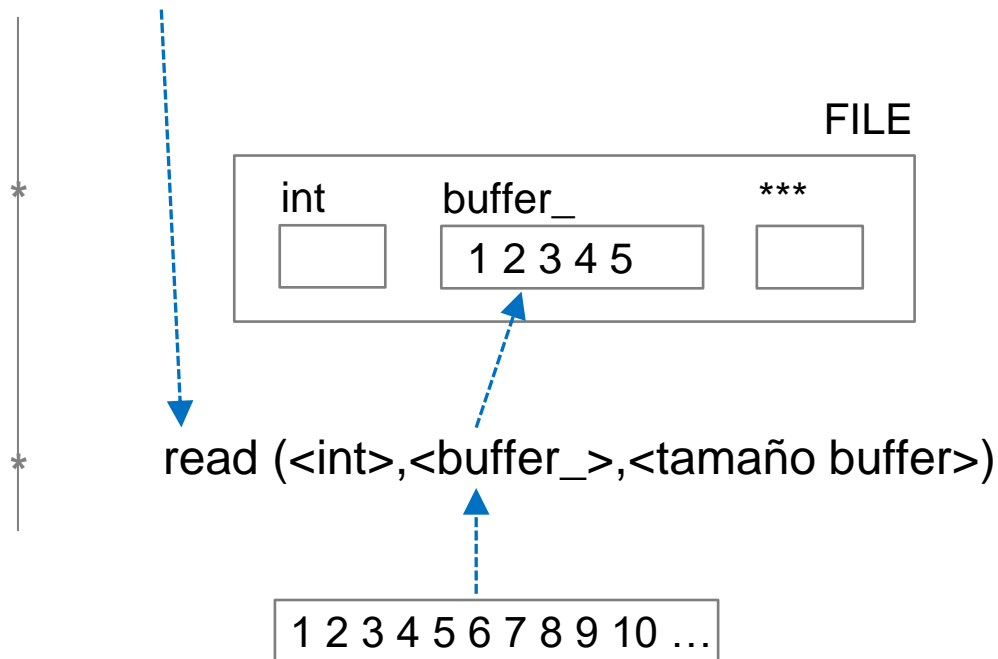
1 2 3 4 5 6 7 8 9 10 ...

Fichero o archivo: interfaz C99

44

Alejandro Calderón Mateos 

`fread (<buffer>,<tamaño 1 elto>,<nº eltos>,<FILE *>)`

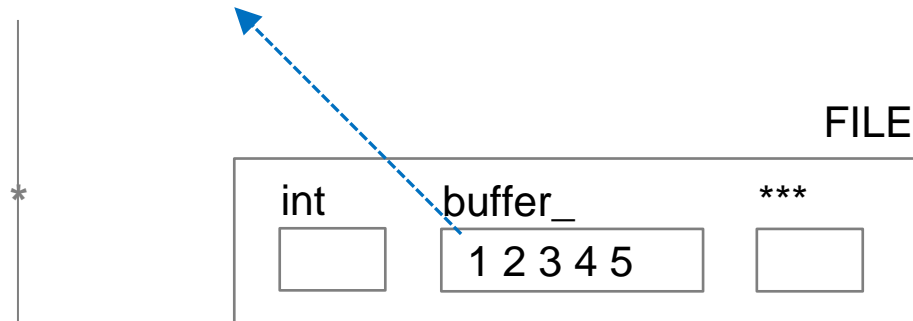


Fichero o archivo: interfaz C99

45

Alejandro Calderón Mateos 

`fread (<buffer>,<tamaño 1 elto>,<nº eltos>,<FILE *>)`



`read (<int>,<buffer_>,<tamaño buffer>)`

`1 2 3 4 5 6 7 8 9 10 ...`

Fichero o archivo: interfaz C99

46

Alejandro Calderón Mateos 

escritura

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>

#define BSIZE 1024

int main ( int argc, char *argv[] )
{
    FILE *fd1 ; int i; double tiempo ;
    char buffer1[BSIZE] ;
    struct timeval ti, tf;

    gettimeofday(&ti, NULL);
    fd1 = fopen ("/tmp/txt2","w+");
    if (NULL == fd1) {
        printf("fopen: error\n");
        exit(-1) ;
    }
    setbuffer(fd1,buffer1,BSIZE) ;
    for (i=0; i<8*1024; i++)
        fprintf(fd1,"%d",i);
    fclose (fd1) ;

    gettimeofday(&tf, NULL);
    tiempo= (tf.tv_sec - ti.tv_sec)*1000 +
            (tf.tv_usec - ti.tv_usec)/1000.0;
    printf("%g milisegundos\n", tiempo);
    return (0) ;
}
```

- Compilar (gcc -o b b.c)
y ejecutar con
 - BSIZE=1024
 - BSIZE=0

Fichero o archivo: interfaz C99

47

Alejandro Calderón Mateos 

escritura

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>

#define BSIZE 1024

int main ( int argc, char *argv[] )
{
    FILE *fd1 ; int i; double tiempo ;
    char buffer1[BSIZE] ;
    struct timeval ti, tf;

    gettimeofday(&ti, NULL);
    fd1 = fopen ("/tmp/txt2", "w+");
    if (NULL == fd1) {
        printf("fopen: error\n");
        exit(-1) ;
    }
    setbuffer(fd1,buffer1,BSIZE) ;
    for (i=0; i<8*1024; i++)
        fprintf(fd1,"%d",i);
    fclose (fd1) ;

    gettimeofday(&tf, NULL);
    tiempo= (tf.tv_sec - ti.tv_sec)*1000 +
            (tf.tv_usec - ti.tv_usec)/1000.0;
    printf("%g milisegundos\n", tiempo);
    return (0) ;
}
```

□ Compilar (gcc -o b b.c)
y ejecutar con

□ BSIZE=1024

□ BSIZE=0

□ Resultados:

□ BSIZE=1024

■ T=0.902 milisegundos

□ BSIZE=0

■ T=14.866 milisegundos

x 15

Contenidos

- Introducción
- **Fichero**
 - ▣ Metadatos
 - ▣ Interfaz
 - ▣ **Métodos de acceso**
 - ▣ Semántica de compartición
- Directorio
- Sistema de ficheros
- Particiones/Volúmenes
- Dispositivos
- Software de sistema

Fichero o archivo: método de acceso *importante*

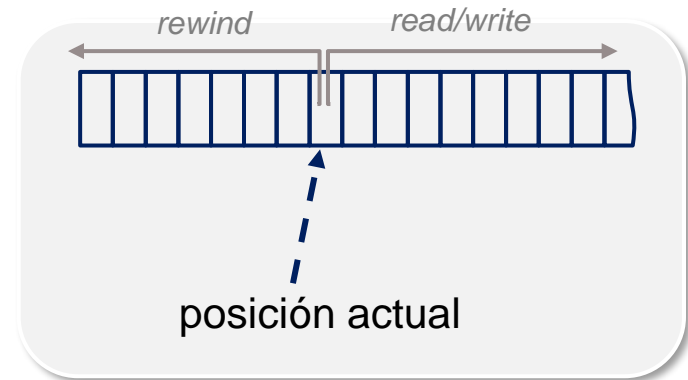
49

Alejandro Calderón Mateos



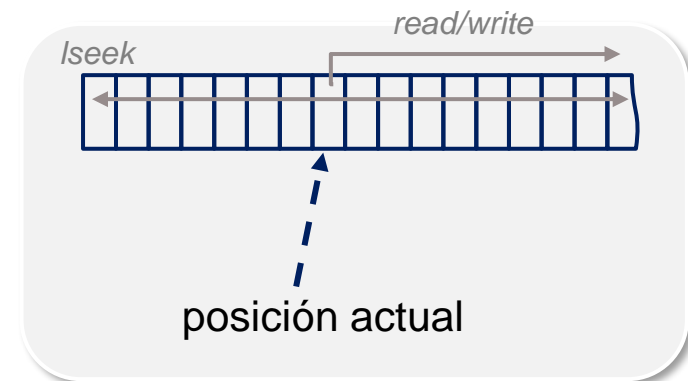
□ Acceso secuencial:

- Dispositivos de acceso secuencial: cintas magnéticas.
- Solo es posible posicionarse (*rewind*) al principio del fichero.



□ Acceso directo:

- Dispositivos de acceso aleatorio: discos duros.
- Es posible posicionarse (*lseek*) en cualquier posición del fichero.
 - Permite construir sobre él otros métodos de acceso (ej.: indexado)



Contenidos

- Introducción
- **Fichero**
 - ▣ Metadatos
 - ▣ Interfaz
 - ▣ Métodos de acceso
 - ▣ **Semántica de compartición**
- Directorio
- Sistema de ficheros
- Particiones/Volúmenes
- Dispositivos
- Software de sistema

Fichero o archivo: semántica de compartición

- Varios procesos pueden acceder simultáneamente a un fichero.
- Es necesario definir una semántica de coherencia:
 - ▣ ¿Cuándo son observables por otros procesos las modificaciones a un fichero?
- Opciones:
 - ▣ Semántica **UNIX**.
 - ▣ Semántica de **sesión**.
 - ▣ Semántica de **versiones**.
 - ▣ Semántica de archivos **inmutables**.

Fichero o archivo: semántica de compartición

importante

52

Alejandro Calderón Mateos 

Semántica Unix	Semántica de sesión	Semántica de versiones	Semántica inmutable
Las escrituras en un archivo son visibles inmediatamente a todos los procesos (y el nuevo puntero de L/E)	Las escrituras en un archivo no son visibles por otros procesos: al cerrar se hace visible .	Las escrituras se hacen sobre copias con número de versión: son visibles al consolidar versiones .	Si se declara compartido un archivo, no se puede modificar
Una vez abierto (<i>open</i>), la familia de procesos creado (<i>fork</i>) comparte su imagen .	Una vez cerrado el fichero, los siguientes procesos que lo abran ven las modificaciones .	Usar sincronización explícita para actualizaciones inmediatas.	Hasta no liberar el cerrojo , ni nombre ni contenido pueden modificarse.
Contención por acceso exclusivo a la imagen única del fichero.	Un fichero puede estar asociado a varias imágenes . No contención.	Tendrá varias imágenes y coste de consolidar .	No hay concurrencia .
Ext3, ufs, etc.	AFS (<i>Andrew File System</i>)	CODA	

SISTEMAS OPERATIVOS: SISTEMAS DE FICHEROS



Ficheros, directorios y sistema de ficheros