Tema 4. Gestión del sistema de ficheros

UNIVERSITAT JAUME·I

Índice

- Definición de fichero
- Atributos de un fichero
- Estructura interna de los ficheros
- Métodos de acceso
- Operaciones básicas sobre ficheros
- El sistema de directorios
- Operaciones básicas sobre directorios
- Enlaces
- Protección de un fichero
- Servicios POSIX para gestión de ficheros
- Servicios POSIX para gestión de directorios

Tema 4. Gestión del sistema de ficheros

UNIVERSITAT JAUME·I

Resultados de aprendizaje

- Explicar los mecanismos de gestión de sistemas de ficheros y resolver casos de uso y diseño
  - Explicar la abstracción de fichero y directorio
  - Describir las operaciones básicas que soportan los ficheros y directorios
  - Mostrar y explicar la evolución del contenido de las tablas que utiliza el SO para gestionar ficheros a medida que se utilizan estos
  - Utilizar servicios POSIX relacionados con la gestión de ficheros y directorios

UNIVERSITAT JAUME·I

Bibliografía

- J. Carretero et al. *Sistemas operativos: Una visión aplicada.* McGraw-Hill. 2007. Capítulo 9.
- José Manuel Badía, M. A. Castaño, Miguel Chóver, Javier Llach, R. Mayo. *Introducción Práctica al Sistema Operativo UNIX.* Col.lecciò "Material docent". Servicio de Publicaciones de la UJI. 1996. Capítulo 16

Introducción

- Los computadores tienen que almacenar información no volátil (programas y datos) → Dispositivos de almacenamiento masivo
  - Gran variedad y tecnología muy diferente (discos, cintas, CD, ..)
- El SO ofrece una visión lógica y uniforme de los dispositivos de almacenamiento secundario a través de los ficheros → Unidad de almacenamiento lógica
- Para facilitar al usuario el manejo de los ficheros, estos se organizan en directorios
- Hay que determinar qué operaciones se pueden realizar sobre estos objetos (ficheros y directorios)
- Es necesario controlar quién y cómo puede acceder a un fichero → Técnicas de protección y seguridad del SF

Introducción

- **Sistema de gestión de ficheros:**
  - Conjunto de SW del sistema que ofrece a los usuarios y aplicaciones unos servicios relativos al empleo de archivos
- **Funciones del sistema de gestión de ficheros:**
  - Ofrecer un conjunto estándar de rutinas de interfaz de E/S
  - Mostrar al usuario los diferentes dispositivos de almacenamiento masivo como sistemas de ficheros (SF)
  - Cumplir con las necesidades de gestión de datos
  - Protección del SF
  - Integridad del SF
  - Optimizar el rendimiento

Visión de usuario

Correspondencia entre visión de usuario y sistema físico de almacenamiento masivo

Muy interrelacionado con el sistema de E/S

Introducción

- El sistema de ficheros proporciona:
  - Un conjunto de **ficheros**: Objetos donde se puede almacenar información en el sistema
  - Estructura de **directorios**: Objetos que permiten organizar los ficheros que contiene el sistema
  - Conjunto de **operaciones** que se pueden realizar sobre ambos objetos

Tema 4. Gestión del sistema de ficheros

UNIVERSITAT JAUME·I

Índice

- Definición de fichero
- Atributos de un fichero
- Estructura interna de los ficheros
- Métodos de acceso
- Operaciones básicas sobre ficheros
- El sistema de directorios
- Operaciones básicas sobre directorios
- Enlaces
- Protección de un fichero
- Servicios POSIX para gestión de ficheros
- Servicios POSIX para gestión de directorios

Definición de fichero

- **¿Qué es un fichero?**
  - Conjunto de información no volátil almacenada en un dispositivo de almacenamiento masivo que tiene unas características
  - Organización lógica para el usuario
- **Datos que contiene un fichero:**
  - Datos para el usuario: Datos del usuario
  - Datos para el SO: Atributos del fichero

Tema 4. Gestión del sistema de ficheros

UNIVERSITAT JAUME·I

Índice

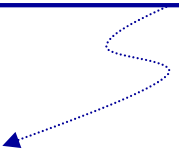
- Definición de fichero
- Atributos de un fichero
- Estructura interna de los ficheros
- Métodos de acceso
- Operaciones básicas sobre ficheros
- El sistema de directorios
- Operaciones básicas sobre directorios
- Enlaces
- Protección de un fichero
- Servicios POSIX para gestión de ficheros
- Servicios POSIX para gestión de directorios

Atributos de un fichero

- Son los datos que definen exactamente al fichero
- Dos tipos:
  - Estáticos: Aquellos que tiene el fichero aunque este no esté siendo utilizando
  - Dinámicos: Aquellos que tiene el fichero cuando algún (uno o más) usuario lo está utilizando

Atributos de un fichero

```
$ ls –l
total 636
-rw------- 1 castano uji  10354 Oct 25 18:24 assignment5.tex
-rw------- 1 castano uji   1330 Nov 11 10:04 books.db
drwx-w---- 2 castano uji      0 Nov  8 18:53 d1
drwx-w---- 2 castano uji      0 Nov  8 18:53 d2
-rwx------ 1 castano uji   5871 Sep  6 12:16 escribe_pid
-rw------- 1 castano uji   1027 Sep  6 12:16 escribe_pid.c
$
```



Atributos estáticos

Atributos del fichero

- **Estáticos:**
  - **Nombre**: Nombre simbólico mediante el cual el usuario accede al fichero
  - **Tipo**: Binario, de texto, código fuente, ejecutable, ... (sólo en caso de sistemas que reconozcan tipos de ficheros)
  - **Ubicación**: Localización en el dispositivo de almacenamiento
  - **Tamaño**: Tamaño actual del fichero, medido en bytes, bloques, registros, etc.
  - **Protección**: Información utilizada en el control de acceso al fichero
  - **Propietario**: Identificación del usuario al que pertenece el fichero
  - **Fechas**: Fecha de creación, de acceso, de modificación, etc.
  - **Enlaces**: Desde qué puntos de la jerarquía de directorios se puede acceder

Pueden existir otros atributos dependiendo del SO

Atributos del fichero

- **Dinámicos:**
  - **Puntero del fichero**: Donde se realizará la siguiente operación de lectura o escritura
  - **Contador de aperturas**: Número de usuarios que están utilizando el fichero en cada momento

Nombres de ficheros y extensiones

- **Nombres:**
  - Longitud máxima para MS-DOS 8 caracteres y 4096 para UNIX
  - Sensibilidad mayúsculas: MS-DOS y Windows no distingue entre mayúsculas, UNIX sí
- **Extensiones:**
  - Indican al SO, a las aplicaciones o a los usuarios características del contenido del fichero
  - Obligatorias o no según SO: MS-DOS sí, UNIX no

.c necesaria para compilador de lenguaje C
.z necesaria para aplicación `compress`

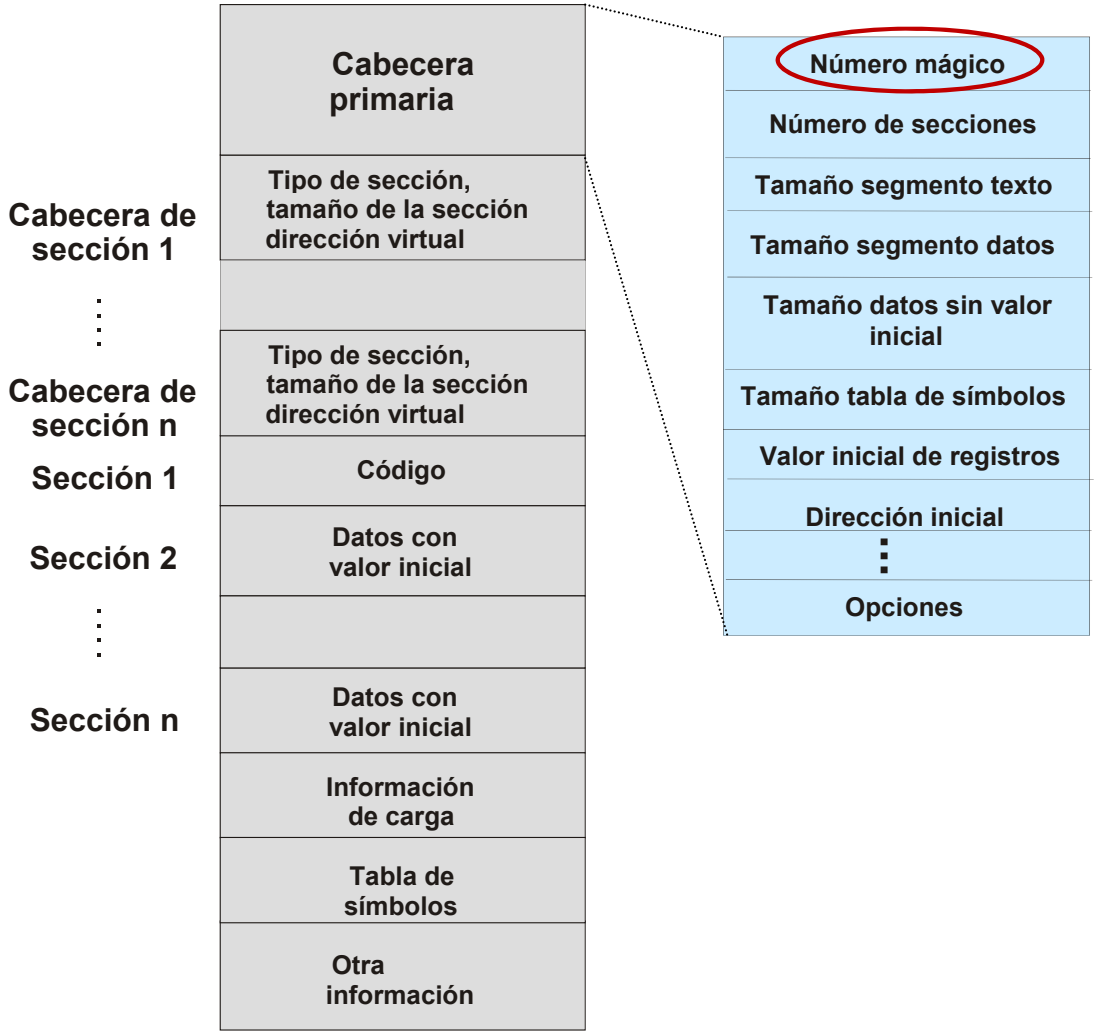
Para el SO no hay diferencia entre ambos tipos de ficheros

Tipos de ficheros

- Cambian de un SO a otros
- Todos los SO deben reconocer al menos un tipo de fichero: sus propios **ficheros ejecutables**
  - Reconocidos por la extensión o
  - Reconocidos por su contenido
    - Ejemplo: Sistemas UNIX
      - ✓ Prácticamente distinguen sólo los ficheros ejecutables
      - ✓ No almacenan entre los atributos el tipo de fichero
      - ✓ En cabecera aparece información para que el SO descifre la estructura del ejecutable
      - ✓ **Número mágico**: número que aparece en cabecera del fichero e indica el tipo de fichero

Tipos de ficheros

Estructura de fichero ejecutable en Linux



Tema 4. Gestión del sistema de ficheros

UNIVERSITAT JAUME·I

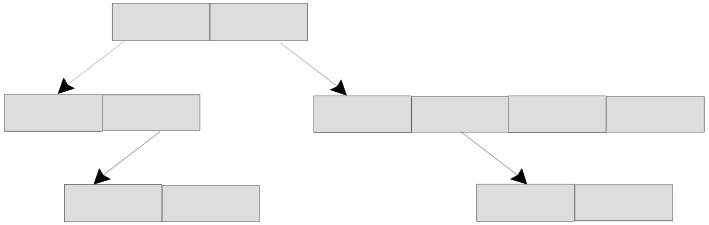
Índice

- Definición de fichero
- Atributos de un fichero
- Estructura interna de los ficheros
- Métodos de acceso
- Operaciones básicas sobre ficheros
- El sistema de directorios
- Operaciones básicas sobre directorios
- Enlaces
- Protección de un fichero
- Servicios POSIX para gestión de ficheros
- Servicios POSIX para gestión de directorios

Estructura interna del fichero

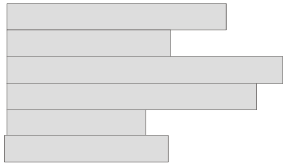
- **Sin formato:**
  - Secuencia de bytes
  - El usuario es el encargado de interpretar la información almacenada en el fichero. La información almacenada no tiene ninguna estructura lógica
- **Con formato:**
  - Estructura sencilla de registros:
    - Secuencia de registros (de longitud fija o variable)
    - La información almacenada está estructurada en registros con estructura lógica
  - Estructuras complejas:
    - Documentos con formato (HTML, postscript)
    - Fichero de carga reubicable

Estructura interna del fichero

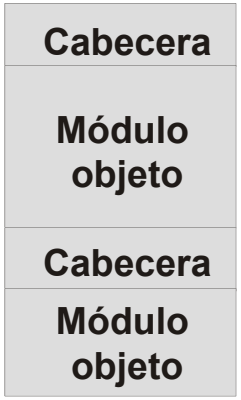


Árbol de registros

Byte o registro de longitud fija



Registros de longitud variable



Estructura con formato

Tema 4. Gestión del sistema de ficheros

UNIVERSITAT JAUME·I

Índice

- Definición de fichero
- Atributos de un fichero
- Estructura interna de los ficheros
- Métodos de acceso
- Operaciones básicas sobre ficheros
- El sistema de directorios
- Operaciones básicas sobre directorios
- Enlaces
- Protección de un fichero
- Servicios POSIX para gestión de ficheros
- Servicios POSIX para gestión de directorios

Métodos de acceso a los datos del fichero

- **Secuencial**
  - Para acceder a la posición P, hay que acceder a las P-1 posiciones previas
- **Directo**
  - Para acceder a la posición P, no hay que acceder a las P-1 posiciones previas
- **Indexado**
  - No se accede por posición sino mediante un índice y una tabla de índices

- A nivel de SO es normal proporcionar acceso directo y/o secuencial. El método indexado se suele proporcionar con herramientas de más alto nivel.

Tema 4. Gestión del sistema de ficheros

UNIVERSITAT JAUME·I

Índice

- Definición de fichero
- Atributos de un fichero
- Estructura interna de los ficheros
- Métodos de acceso
- Operaciones básicas sobre ficheros
- El sistema de directorios
- Operaciones básicas sobre directorios
- Enlaces
- Protección de un fichero
- Servicios POSIX para gestión de ficheros
- Servicios POSIX para gestión de directorios

Operaciones básicas sobre ficheros

- **Crear:** Crear un fichero
- **Escribir**: Introducir datos en el fichero
- **Leer**: Recuperar datos del fichero
- **Reposicionar**: Modificar la posición del puntero de lectura/escritura del fichero donde se producirá la siguiente operación
- **Borrar**: Eliminar el fichero
- **Truncar**: Eliminar todos o parte de los datos de un fichero (desde un punto hasta el final)
- **Modificar atributos:** Nombre, permisos, etc.

Algunas de estas operaciones pueden ser consideradas como operaciones básicas sobre el sistema de directorios

Operaciones básicas sobre ficheros

- Un fichero debe ser abierto antes de usarlo
- **¿Qué quiere decir abrir un fichero?**
  - Acceder al directorio donde se encuentra y obtener la información (**entrada de directorio**) que permitirá después acceder a los datos

    Esta información se lleva a memoria central; así, acceder a los datos no implica buscar de nuevo la información del fichero en el directorio
  - Comprobar que el usuario tiene permisos de acceso
  - Proporcionar un identificativo temporal para manipularlo

O sea, comprobar que existe

Operaciones básicas sobre ficheros

- **¿Qué quiere decir cerrar un fichero?**
  - Eliminar el identificativo temporal para manipularlo
  - Liberar los recursos de memoria que ocupa el fichero
  - Actualizar información en disco

Tema 4. Gestión del sistema de ficheros

UNIVERSITAT JAUME·I

Índice

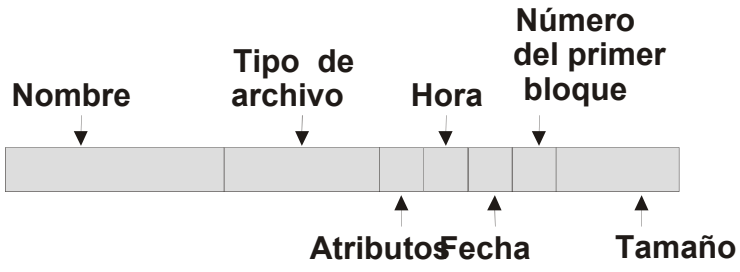
- Definición de fichero
- Atributos de un fichero
- Estructura interna de los ficheros
- Métodos de acceso
- Operaciones básicas sobre ficheros
- El sistema de directorios
- Operaciones básicas sobre directorios
- Enlaces
- Protección de un fichero
- Servicios POSIX para gestión de ficheros
- Servicios POSIX para gestión de directorios

El sistema de directorios

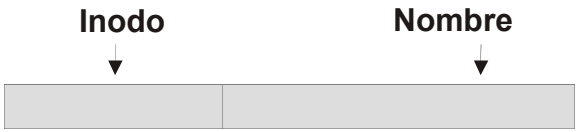
- No puede existir un sistema de ficheros sin un sistema de directorios
- Los **directorios** son **ficheros especiales en los cuales la información almacenada pertenece al SO** y no a los usuarios
- Puesto que un directorio es un fichero (especial), tendrá las mismas características que las indicadas para los ficheros

El sistema de directorios

- **Estructura de un directorio:**
  - La forma más sencilla de ver un directorio es como un fichero organizado en registros, donde cada registro contiene la información relativa a un fichero o subdirectorio que contiene



Directorio de MS-DOS



Directorio de UNIX

El sistema de directorios

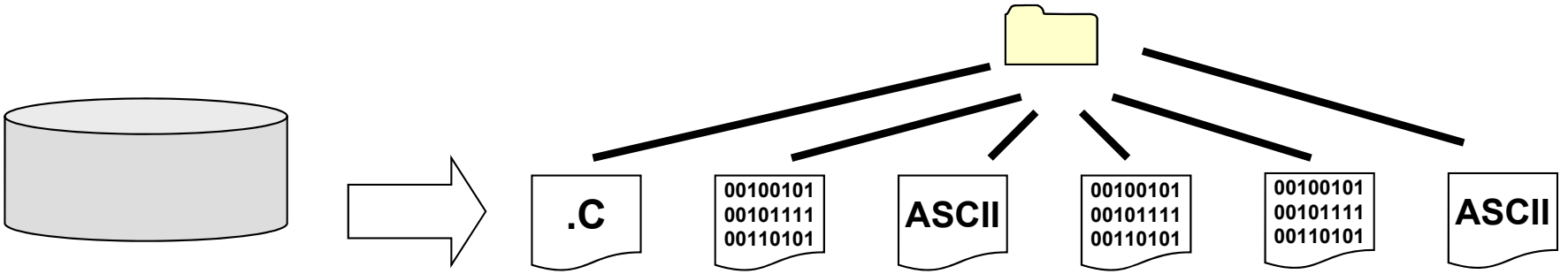
- **Finalidad doble de un directorio:**
  - Cara al usuario:
    - Permite organizar los ficheros de este
  - Cara al SO:
    - Permite transformar un nombre de fichero en las estructuras de datos necesarias para acceder a los datos del fichero

Organización de directorios

- **Directorio de un nivel**
- **Árbol de directorios**
- **Grafo de directorios:**
  - Permite compartición de ficheros y subdirectorios
  - Forma habitual de compartir ficheros: creación de enlaces

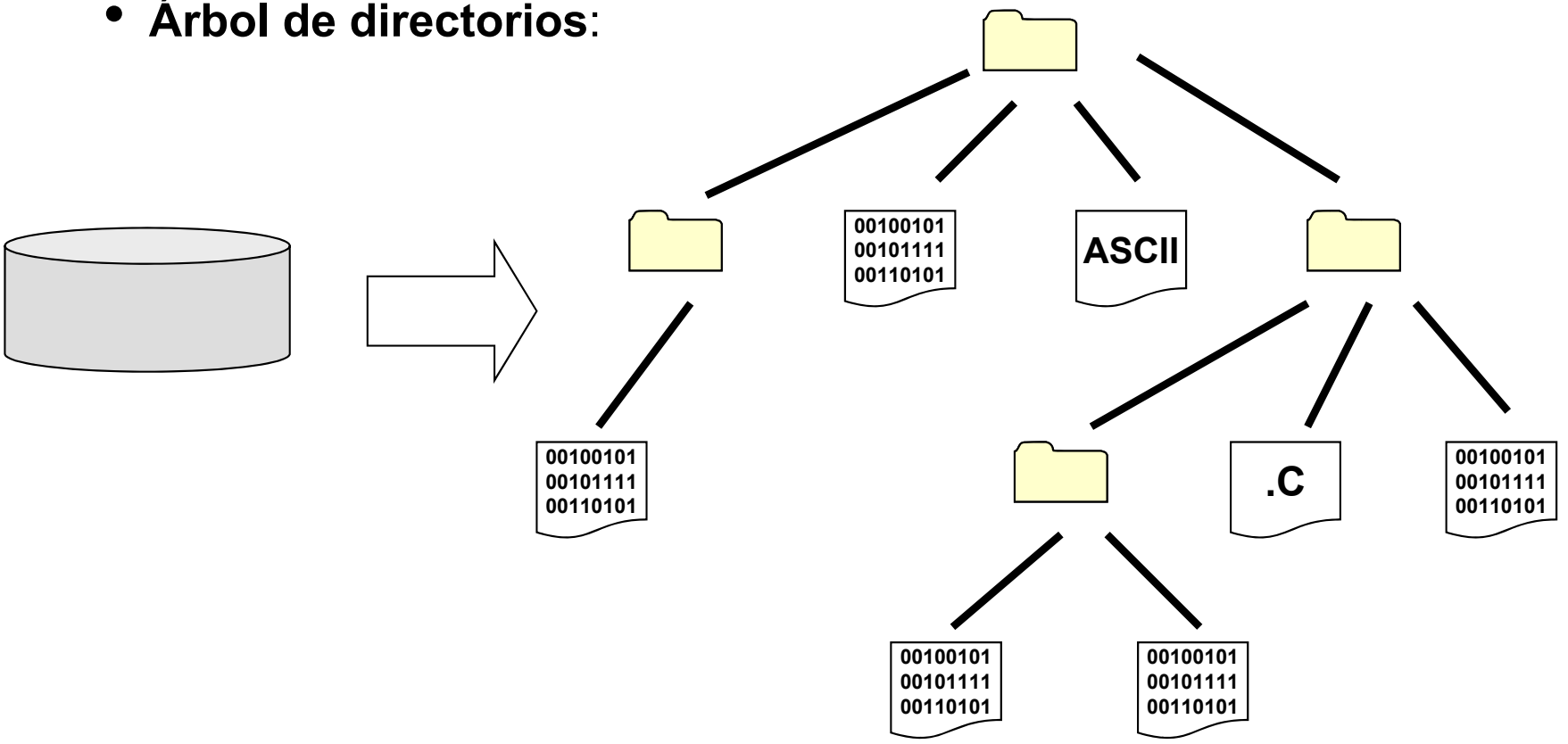
Organización de directorios

- **Un nivel**: Todos los ficheros en un único directorio.



Organización de directorios

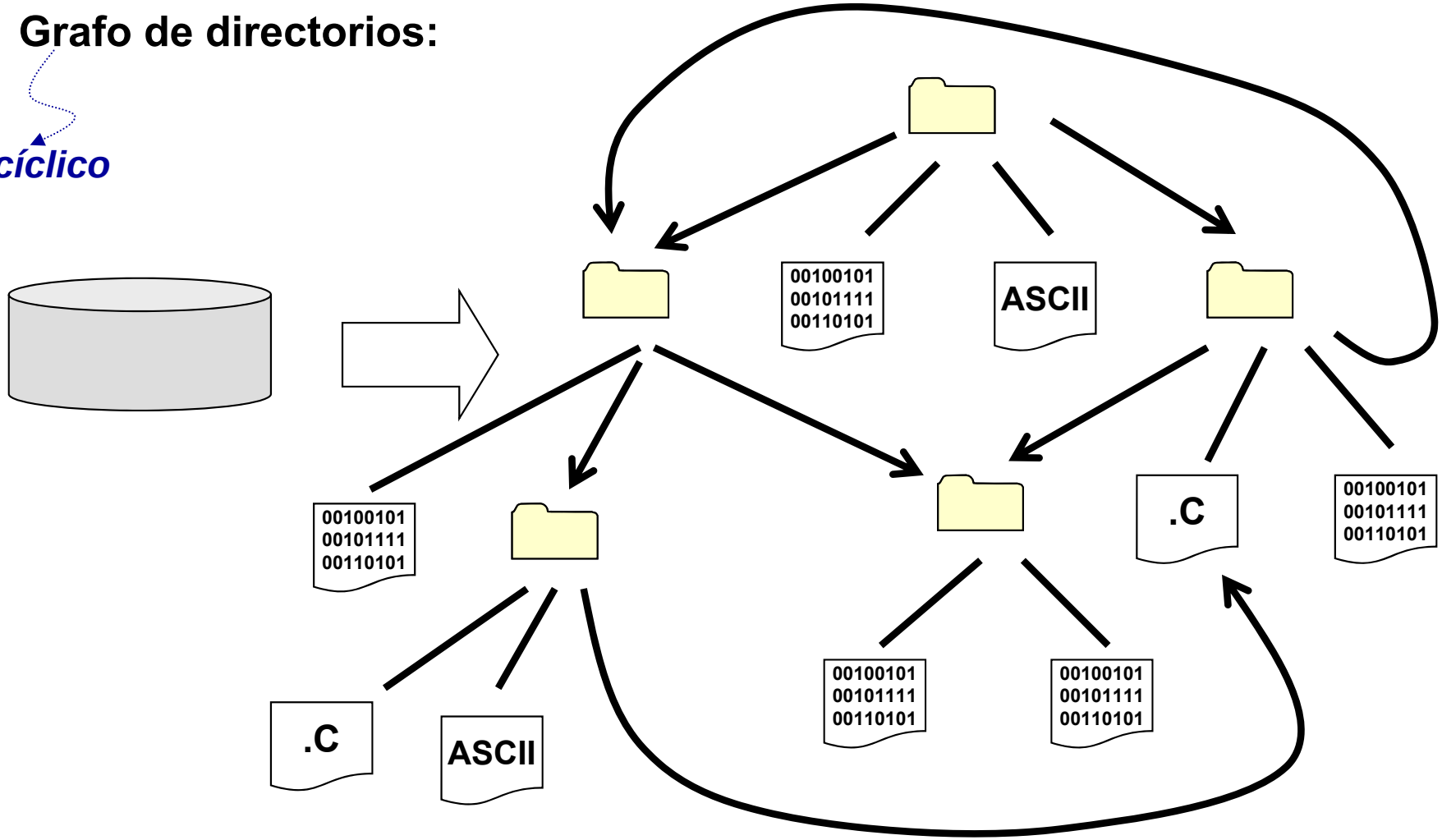
- **Árbol de directorios**:



Organización de directorios

- **Grafo de directorios:**

Acíclico



Organización de directorios

UNIVERSITAT JAUME·I

- **Rutas de acceso:**
  - La entradas `.` y `..` pueden utilizarse para formar rutas de acceso
    - . se refiere al directorio de trabajo
    - .. se refire al directorio que lo contiene (directorio padre)

Directorio actual en el que nos encontramos

- **Tipos de rutas:**
  - Nombre completo o absoluto (dado a partir del directorio raíz—empieza por `/`)
    - `/usr/include/stdio.h`
  - Nombre relativo (al directorio de trabajo)
    - `ejemplo.pdf` está en el directorio actual
    - `../ejemplo2.pdf` está en el directorio que contiene al directorio actual

Tema 4. Gestión del sistema de ficheros

UNIVERSITAT JAUME·I

Índice

- Definición de fichero
- Atributos de un fichero
- Estructura interna de los ficheros
- Métodos de acceso
- Operaciones básicas sobre ficheros
- El sistema de directorios
- Operaciones básicas sobre directorios
- Enlaces
- Protección de un fichero
- Servicios POSIX para gestión de ficheros
- Servicios POSIX para gestión de directorios

Operaciones básicas sobre directorios

- **Crear un directorio:**
  - Añadir un registro (entrada) con información sobre el nuevo directorio y asignar espacio para el nuevo directorio
  - Para POSIX y Win32: Además, añadir en el nuevo directorio dos registros para los directorios “.” y “..”
- **Borrar un directorio**:
  - Eliminar el registro con información sobre el directorio y liberar el espacio que ocupa

    Habitualmente sólo se puede borrar un directorio vacío

    Directorio vacío para POSIX y FAT: Fichero con sólo dos entradas, para los directorios “.” y “..”

Operaciones básicas sobre directorios

- **Abrir**: Abrir el fichero que contiene los datos del directorio
  Para poder manipular el contenido de un fichero directorio es necesario abrirlo
- **Cerrar**: Cerrar el fichero que contiene los datos del directorio
- **Leer** : Extraer la siguiente entrada del directorio
- **Cambiar de directorio:** Cambiar el directorio de trabajo de un usuario

Operaciones relacionadas con el sistema de directorios

UNIVERSITAT JAUME·I

- Borrar un fichero (de usuario o directorio)
- Crear un fichero (de usuario o directorio)
- Listar un fichero (de usuario o directorio)
- Cambiar el nombre de un fichero (de usuario o directorio)
- Recorrer el sistema de ficheros

Tema 4. Gestión del sistema de ficheros

UNIVERSITAT JAUME·I

Índice

- Definición de fichero
- Atributos de un fichero
- Estructura interna de los ficheros
- Métodos de acceso
- Operaciones básicas sobre ficheros
- El sistema de directorios
- Operaciones básicas sobre directorios
- Enlaces
- Protección de un fichero
- Servicios POSIX para gestión de ficheros
- Servicios POSIX para gestión de directorios

Enlaces

- **Tipos de enlaces:**
  - Enlaces simbólicos:
    - Fichero nuevo con el nombre del fichero destino dentro
    - Acceso directo de Windows o “soft link” de UNIX
  - Enlaces físicos:
    - Un único fichero con contador de enlaces
    - “Hard link” de UNIX

Enlaces

- **Enlaces físicos:**
  - Los enlaces se contabilizan a través de un **contador de enlaces** que aparece como atributo en el inodo

    En POSIX: Campo `st_nlink` de la estructura que devuelve `stat`

  - Crear de un enlace $\Rightarrow$ Incrementar contador de enlaces
  - Borrar un fichero $\Rightarrow$ Decrementar contador de enlaces
    - Puede hacerse por varios caminos
    - Sólo se borran los datos de un fichero si el contador de enlaces es cero tras el decremento

Enlaces

- **Enlaces lógicos:**
  - En POSIX se identifican a través de:

    Campo `st_mode` de la estructura que devuelve `stat`

Enlaces

```
$pwd
/home/castano
$ln f f.hard
$ln –s f f.soft
```

Enlace blando

Inodo nº 40 (f.soft)

st_mode	
Otros campos	
800	NIL
NIL	NIL
NIL	NIL
NIL	NIL

Bloque 800

f
FREE

Bloque de /home/castano

.	10
..	30
f	20
f.hard	20
f.soft	40
...	...

Inodo nº 20 (f y f.hard)

Otros campos	
st_nlink 2	
600	NIL
NIL	NIL
NIL	NIL
NIL	NIL

N° enlaces duros

Bloque 600

Datos
FREE

Enlaces

- **Peligro de los enlaces:**
  - Existencia de bucles en el árbol de directorios (ciclos en el grafo)
    - No permiten recorrer el árbol de directorios completo
    - Pueden hacer que una traducción de nombre no acabe nunca
  - Algunas soluciones:
    - Permitir sólo enlaces a ficheros, no a subdirectorios
    - Algoritmo de búsqueda de bucle cuando se crea un enlace
  - Limitaciones de implementación en UNIX:
    - No se pueden establecer enlaces duros entre SF diferentes
    - No se pueden establecer enlaces duros a un directorio

Tema 4. Gestión del sistema de ficheros

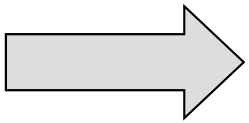
UNIVERSITAT JAUME·I

Índice

- Definición de fichero
- Atributos de un fichero
- Estructura interna de los ficheros
- Métodos de acceso
- Operaciones básicas sobre ficheros
- El sistema de directorios
- Operaciones básicas sobre directorios
- Enlaces
- Protección de un fichero
- Servicios POSIX para gestión de ficheros
- Servicios POSIX para gestión de directorios

Esquema de protección de datos

Dispositivo compartido por todos los usuarios

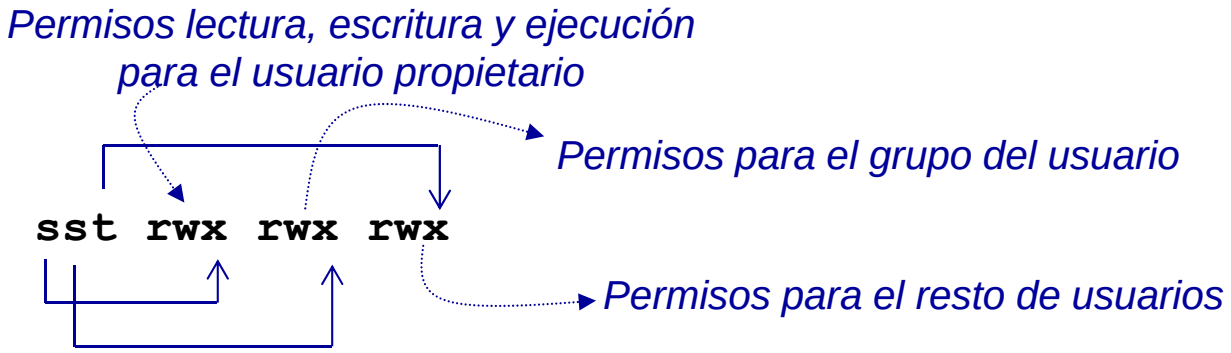


Sistema de protección de datos
Control de acceso a los ficheros por parte de los usuarios

- **Capacidades:**
  - Cada fichero (y directorio) tiene asignadas unas capacidades (permisos) que indican qué usuarios y qué tipo de acceso pueden hacer
- **Listas de control de acceso:**
  - Cada fichero (y directorio) tiene una lista de los usuarios que pueden acceder y qué tipo de acceso pueden hacer

Permisos de acceso en UNIX

- **Permisos básicos:**
  - Se representa mediante una máscara de 9 bits y se suele expresar en octal mediante un número de 3 dígitos
  - Formato en octal:



    - t: bit “sticky” (pegajoso)
    - ss: bits “setuid” y “setguid”

Los veremos en “Administración de SO”

  - Ejemplo: 0644 (110 100 100)

Tema 4. Gestión del sistema de ficheros

UNIVERSITAT JAUME·I

Índice

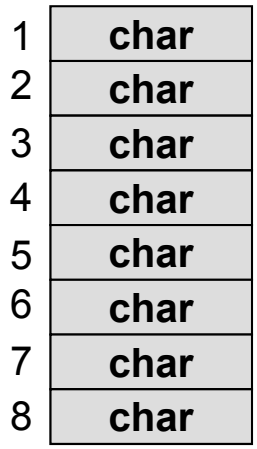
- Definición de fichero
- Atributos de un fichero
- Estructura interna de los ficheros
- Métodos de acceso
- Operaciones básicas sobre ficheros
- El sistema de directorios
- Operaciones básicas sobre directorios
- Enlaces
- Protección de un fichero
- Servicios POSIX para gestión de ficheros
- Servicios POSIX para gestión de directorios

Servicios POSIX para gestión de ficheros

- **Crear un fichero:** `creat`
- **Borrar un fichero:** `remove`
- **Abrir un fichero:** `open`
- **Cerrar un fichero:** `close`
- **Leer datos de un fichero:** `read`
- **Escribir datos en un fichero:** `write`
- **Duplicar un descriptor de fichero:** `dup`
- **Cambiar puntero de un fichero:** `lseek`
- **Consultar atributos de un fichero:** `stat, fstat, lstat`
- **Cambiar nombre de un fichero:** `rename`

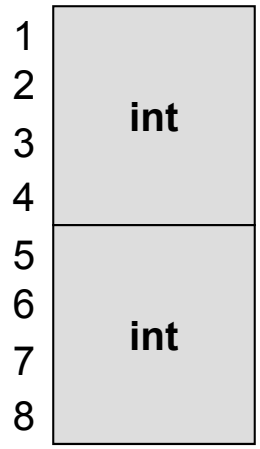
Conceptos previos

- **Visión del usuario de un fichero Unix:**
  - Fichero de acceso directo sin formato
  - Secuencia de bytes a los que se puede acceder de forma directa
  - Interpretación de los datos del fichero dependiente del programa que lo utilice



Secuencia de 8 caracteres

Secuencia de 4 caracteres y 1 entero



Secuencia de 2 enteros

Recordando...

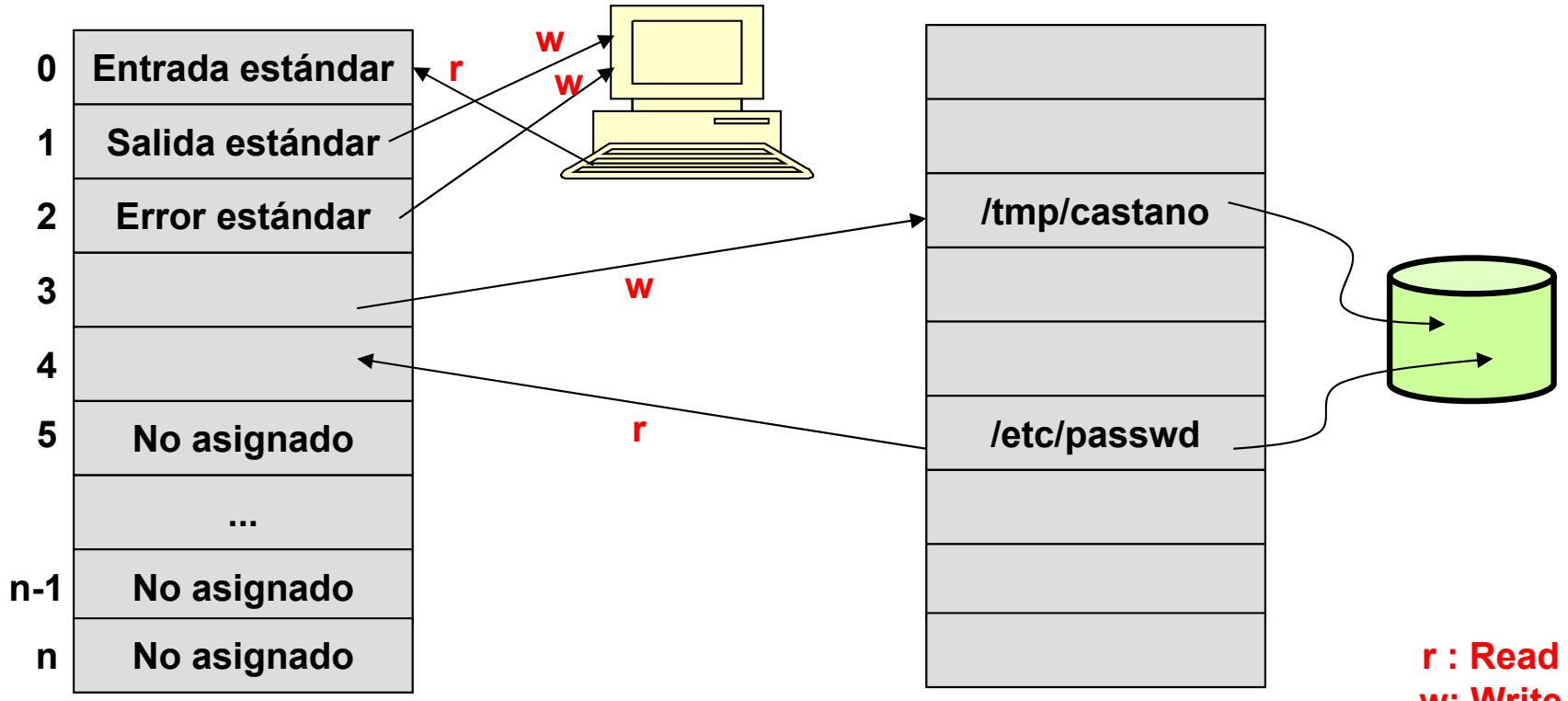
- **Tabla de descriptores de ficheros (tabla de ficheros abiertos por el proceso):**
  - Vector que apunta a los ficheros utilizados por un proceso
  - Cada proceso tiene su propia tabla de descriptores de ficheros
  - Se crea al crearse el proceso
  - Se almacena en el BCP del proceos
  - Al crear un proceso hay 3 descriptores de fichero abiertos por defecto:
    - Entrada estándar: Descriptor 0
    - Salida estándar: Descriptor 1
    - Error estándar: Descriptor 2

Recordando...

- **Tabla de descriptores de ficheros (tabla de ficheros abiertos por el proceso):**
  - Cuando se crea un nuevo proceso (`fork`)
    - El hijo hereda la tabla de ficheros abiertos del padre
  - Cuando se cambia el código de un proceso (`exec`)
    - La tabla de ficheros abiertos por el proceso se mantiene

Recordando...

- Tabla de descriptores de ficheros (tabla de ficheros abiertos por el proceso):



T. ficheros abiertos por proceso

T. ficheros abiertos en el sistema

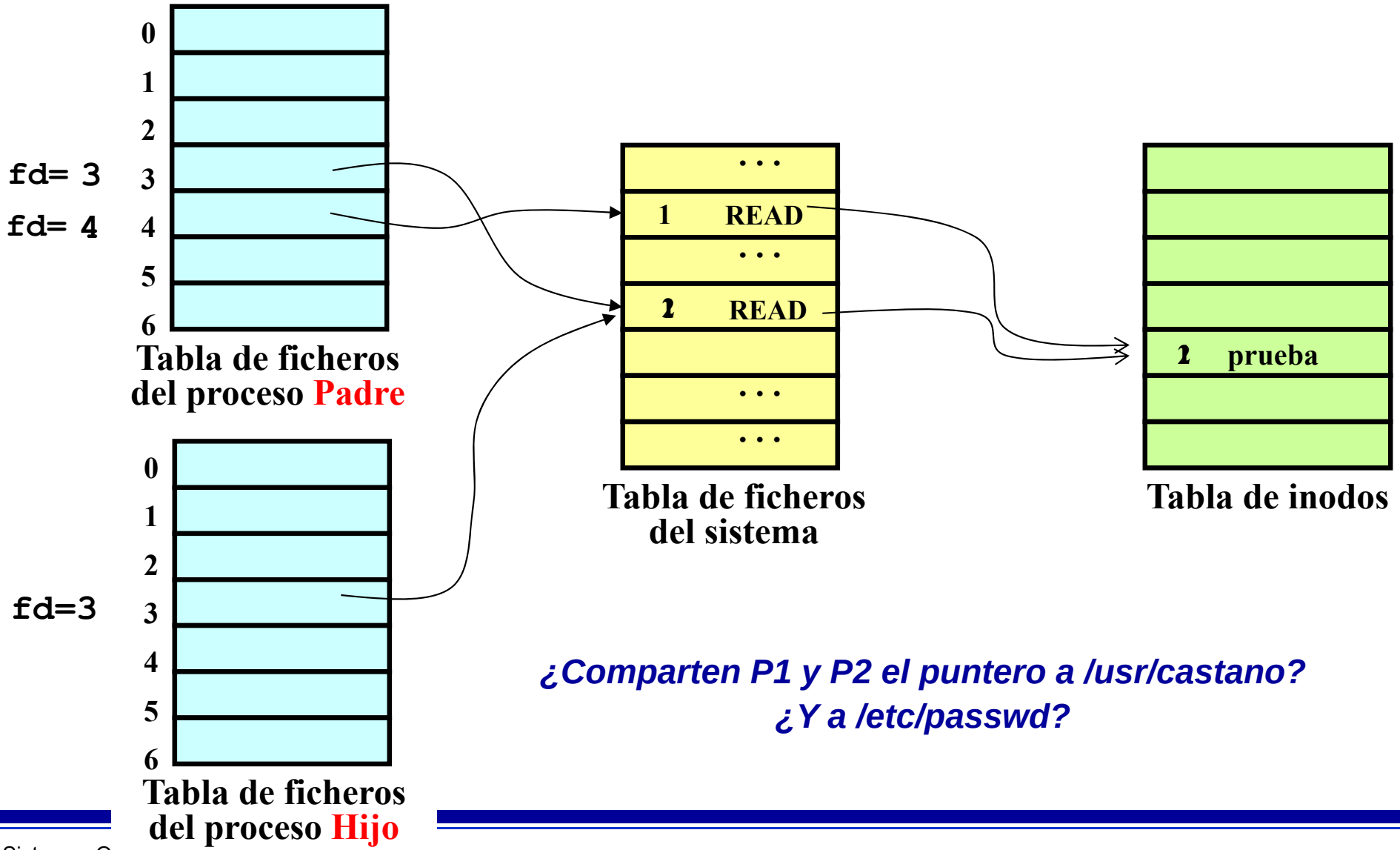
Recordando...

- **Apertura de un fichero:**
  - Localizar fichero
  - Obtener atributos
  - Comprobar que el usuario tiene permisos
  - Crear una entrada en la tabla de ficheros abiertos por el proceso
  - Devolver un **descriptor de fichero** (índice entero a la tabla de descriptores de ficheros) que identifica al fichero
  - Crear una entrada en la tabla de ficheros abiertos en el sistema
  - Cargar su inodo en la tabla de inodos del sistema. Si existía ya, añadir contador de aperturas en la entrada

Recordando...

- **Puntero de lecutra/escritura de un fichero:**
  - Característica de un fichero, no de un proceso
  - Se guarda en el tabla de ficheros abiertos en el sistema
  - Padre e hijo comparten el puntero a un fichero si este estaba abierto cuando el padre creó al hijo
  - Cuando un proceso ejecuta una función `exec` se mantienen los punteros a los ficheros abiertos por el proceso antes del `exec`

Recordando...



¿Comparten P1 y P2 el puntero a /usr/castano?
¿Y a /etc/passwd?

Funciones de gestión de ficheros

UNIVERSITAT JAUME·I

- **Funciones POSIX de gestión básica de ficheros:**

```
int open(const char *nombre, int modo, mode_t permisos);
int close(int desc_fich);
int creat(const char *nombre, mode_t permisos);
int remove(const char *nombre);
int read(int desc_fich, void *dato, size_t n_bytes);
int write(int desc_fich, const void *dato, size_t n_bytes);
int dup(int desc_fich);
```

Ejemplo

- **Ejemplo 1:**

```
$ cat dup1.c
#include<unistd.h>
#include<fcntl.h>
#include<stdio.h>
#include <stdlib.h>

int main()
{
   int fichero, duplicado, i, dato, salir;
                                          /* (a) */
   fichero = creat("prueba",0644);        /* (b) */
   for (i=0;i<10;i++) write(fichero,&i,sizeof(int));
   close(fichero);                        /* (c) */

   fichero = open("prueba",O_RDONLY);     /* (d) */
   printf("Abierto el fichero con descriptor %d\n",fichero);
   duplicado=dup(fichero);                /* (e) */
   printf("Duplicado sobre descriptor %d\n",duplicado);
```

Ejemplo

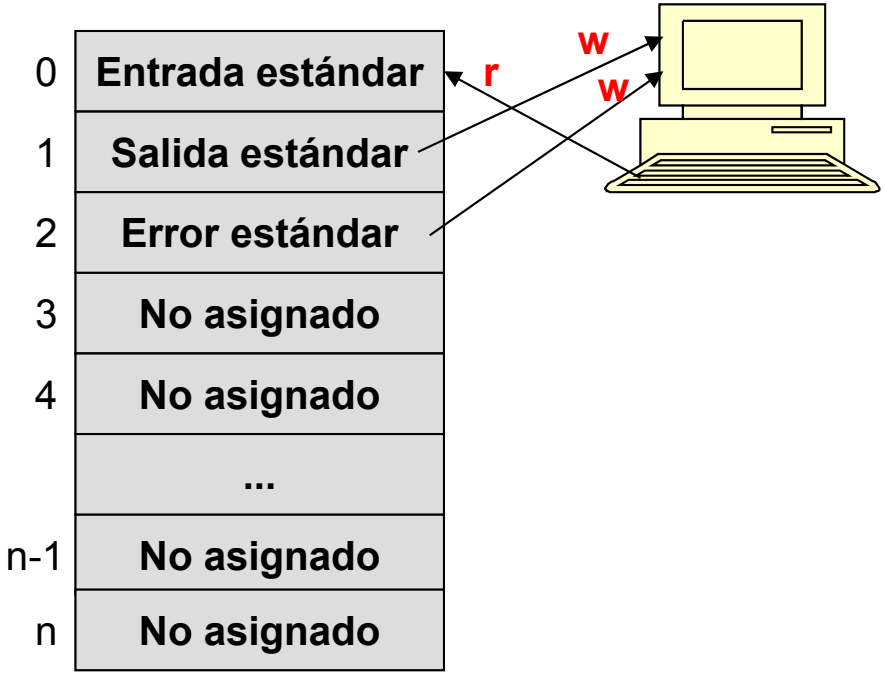
- **Ejemplo 1 (cont.):**

```
        salir=0;
        while (!salir)
        {
          salir = (read(fichero,&dato,sizeof(int))==0);
          if (!salir)
          {
            printf("Dato leido mediante fichero = %d\n",dato);
            salir = (read(duplicado,&dato,sizeof(int))==0);
            if (!salir)
            {   printf("Dato leido mediante duplicado = %d\n",dato);
            } else {
                printf("Final de fichero encontrado en duplicado\n");
            }
          } else {printf("Final de fichero encontrado en fichero\n");}
      }
      close(fichero);
      close(duplicado);
      exit(0);
    }
```

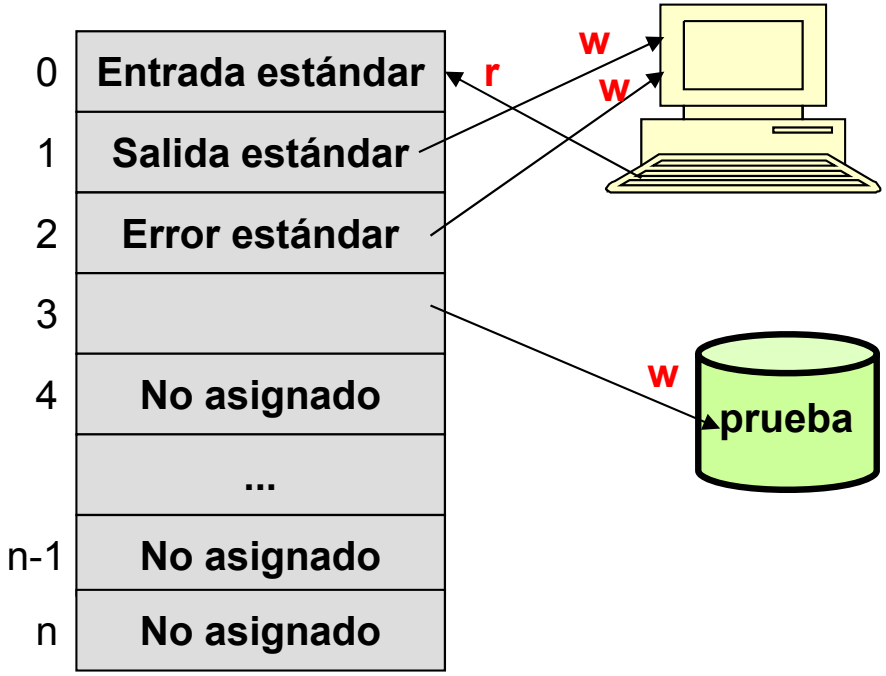
Ejemplo

- **Ejemplo 1 (cont.):**

Evolución de la tabla de descriptores de ficheros



(a)



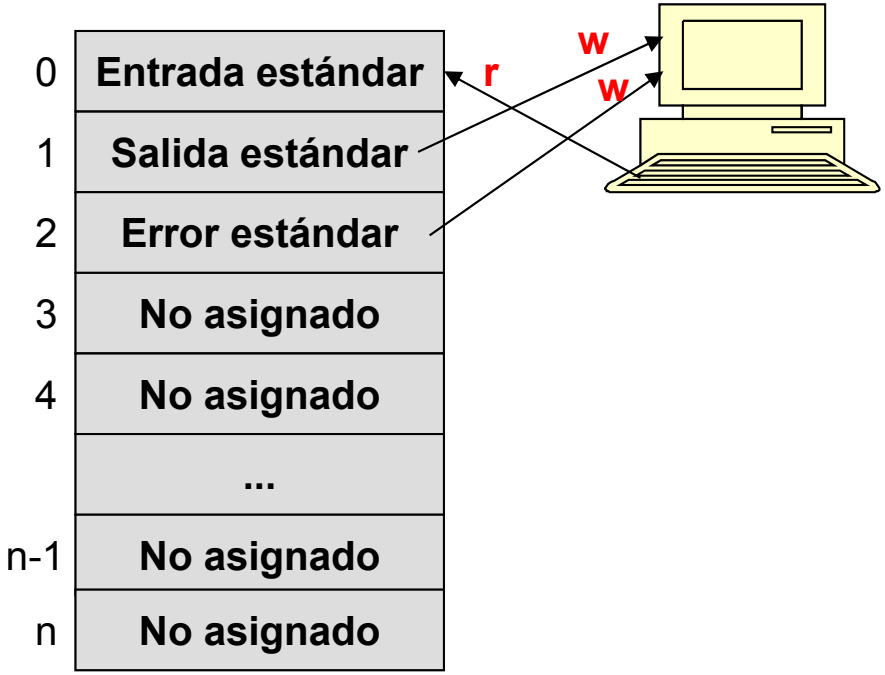
(b)

fichero=3

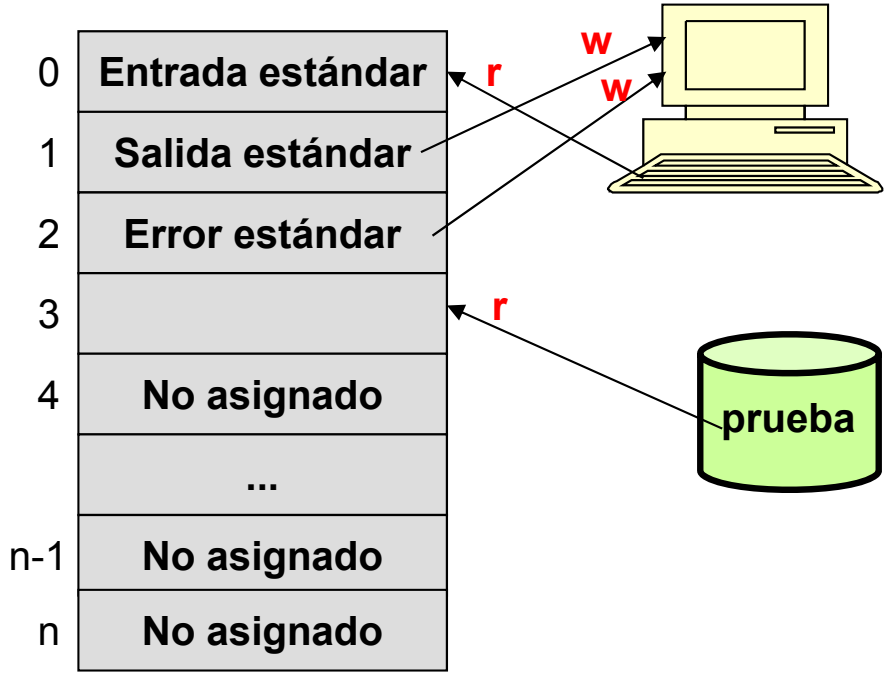
Ejemplo

- **Ejemplo 1 (cont.):**

Evolución de la tabla de descriptores de ficheros (cont.)



(c)



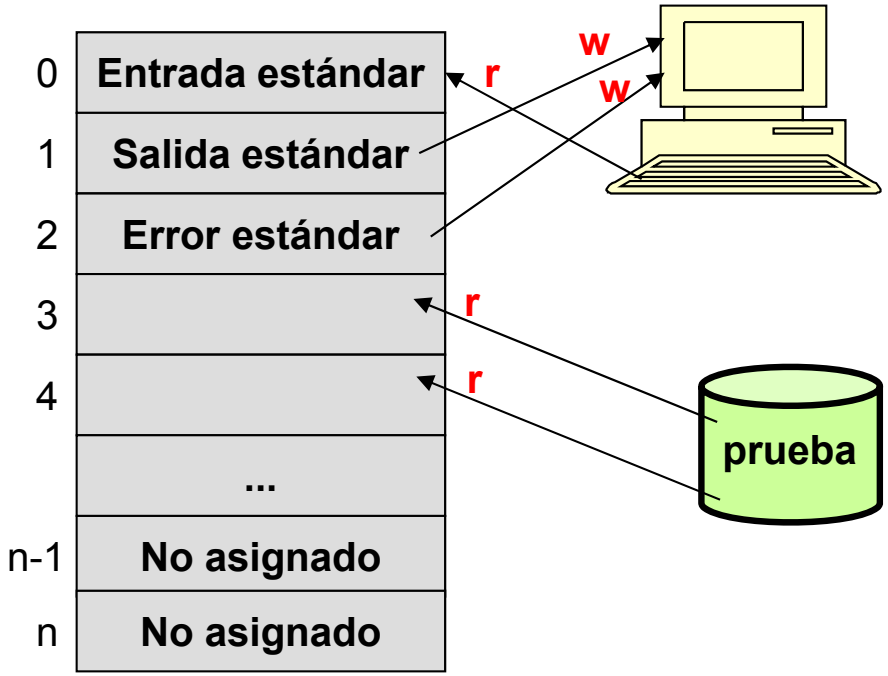
(d)

fichero=3

Ejemplo

- **Ejemplo 1 (cont.):**

Evolución de la tabla de descriptores de ficheros



(d)
fichero=3
duplicado=4

Ejemplo

- **Ejemplo 1 (cont.):**

```
$ make dup1
$ dup1
Ya se ha creado el fichero
Abierto el fichero con descriptor 3
Duplicado sobre descriptor 4
Dato leido mediante fichero = 0
Dato leido mediante duplicado = 1
Dato leido mediante fichero = 2
Dato leido mediante duplicado = 3
Dato leido mediante fichero = 4
Dato leido mediante duplicado = 5
Dato leido mediante fichero = 6
Dato leido mediante duplicado = 7
Dato leido mediante fichero = 8
Dato leido mediante duplicado = 9
Final de fichero encontrado en fichero
$
```

Función *lseek*

- **Sintaxis:**

```
off_t lseek(int desc_fich, int despl, off_t tipo_despl);
```

devuelve:

  - La nueva posición del puntero con respecto al inicio del fichero
  - Si error: -1

- **Descripción:**
  - Modifica el puntero de lectura/escritura del fichero con descriptor `desc_fich`:
    - Si `tipo_despl=SEEK_SET(0)`: el puntero avanza `despl` bytes con respecto al inicio del fichero
    - Si `tipo_despl=SEEK_CUR(1)`: el puntero avanza `despl` bytes con respecto a su posición actual
    - Si `tipo_despl=SEEK_END(2)`: el puntero avanza `despl` bytes con respecto al final del fichero
  - Si `despl` es negativo se produce un retroceso del puntero

Ejemplo

- **Ejemplo 2:**

```
$ cat ej_lseek.c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>

int main(int argc, char * argv[])
{
  int i, j, fd;

  if ( argc != 2 )
  {  printf(" numero de argumentos incorrecto\n");
     exit(-1);
  }

  fd=creat(argv[1],0755);
  for (i=1; i<9; i++) write(fd,&i,sizeof(int));
  close(fd);
```

Ejemplo

- **Ejemplo 2 (cont.):**

```
printf("Leyendo números impares...\n");
  fd=open(argv[1],0);
  while (read(fd,&i,sizeof(int)) > 0)
  {   printf("Leido: %d\n",i);
      /* O bien:  write(1,&i,sizeof(int));
                  write(1,"\n",1);          */
      lseek(fd,4,1);
  }
  close(fd);

  printf("\nLeyendo números pares...\n");
  fd=open(argv[1],0);
  lseek(fd,4,1);
  while (read(fd,&i,sizeof(int)) > 0)
  {   printf("Leido: %d\n",i);
      lseek(fd,4,1);
  }
  close(fd);
```

Asumiendo enteros de 4 bytes

Ejemplo

- **Ejemplo 2 (cont.):**

```
    printf("\nLeyendo el fichero al revés...\n");
    fd=open(argv[1],0);
    lseek(fd,-4,2);
    for (j=1; j<9; j++)
    {   read(fd,&i,sizeof(int));
        printf("Leido: %d\n",i);
        lseek(fd,-8,1);
    }

    close(fd);

    exit(0);
}
```

Ejemplo

- **Ejemplo 2 (cont.):**

```
$ make ej_lseek
$ ej_lseek f
Leyendo numeros impares...
Leido: 1
Leido: 3
Leido: 5
Leido: 7

Leyendo numeros pares...
Leido: 2
Leido: 4
Leido: 6
Leido: 8

Leyendo el fichero al reves...
Leido: 8
Leido: 7
Leido: 6
Leido: 5
Leido: 4
Leido: 3
Leido: 2
Leido: 1
$
```

Función *stat*

- **Sintaxis:**

```
int stat (const char *nombre, struct stat *datos);
```

  devuelve:

  - Si todo ha ido bien: 0
  - Si error: -1

- **Descripción:**
  - Proporciona en una estructura de tipo `stat` apuntada por `datos` información sobre los atributos del fichero `nombre`
- **Ejemplos:**
  - `stat (“prueba”, &datos);`
  - `stat (“/usr/castano/prueba”, &datos);`

Función *stat*

- **Estructura *stat*:**

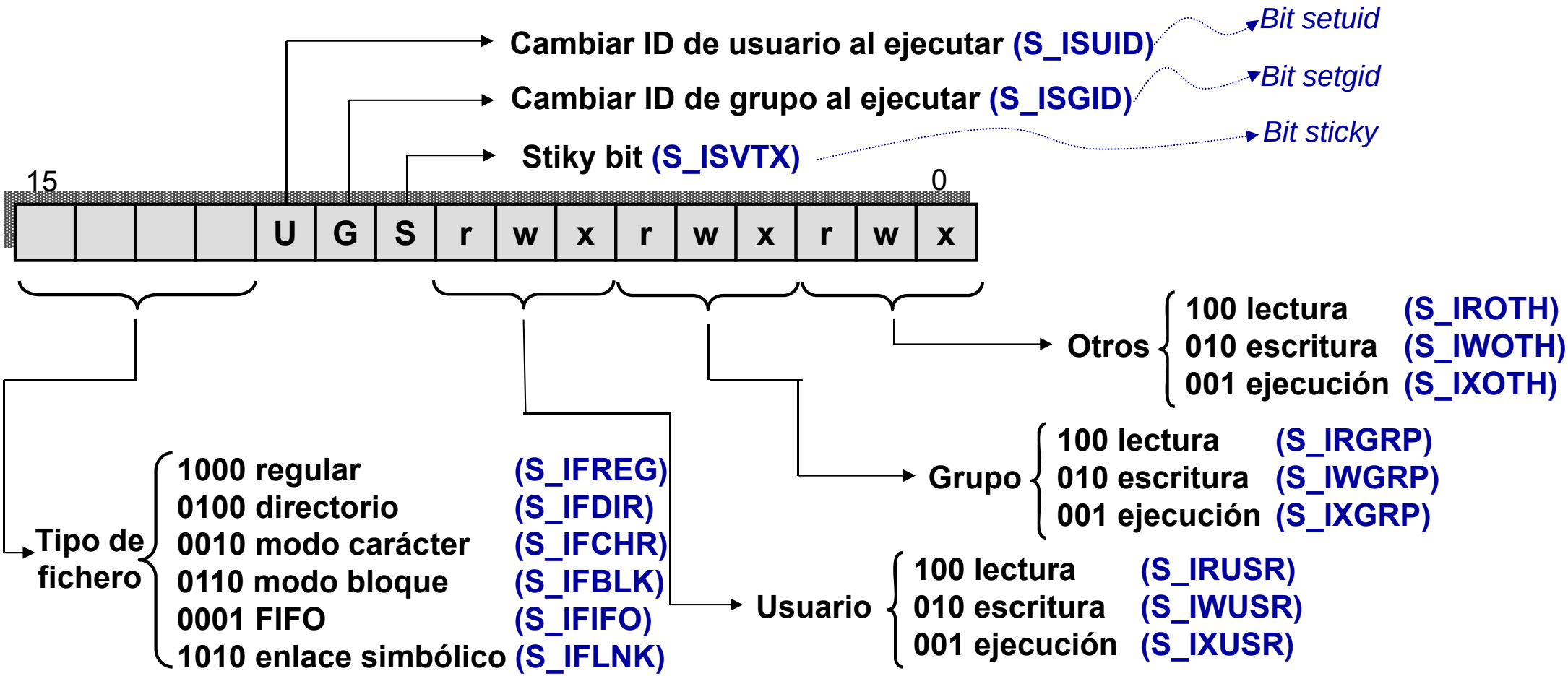
```
struct stat {
   dev_t   st_dev;   /* Número de dispositivo que contiene el inodo */
   ino_t   st_ino;   /* Número de inodo */
   mode_t  st_mode;  /* 16 bits que codifican el modo (tipo y
                        permisos) del fichero */
   nlink_t st_nlink; /* Número de enlaces duros del fichero */
   uid_t   st_uid;   /* UID del propietario */
   gid_t   st_gid;   /* GID del propietario */
   off_t   st_size;  /* Tamaño del fichero en bytes */
   time_t  st_atime; /* Tiempo del último acceso */
   time_t  st_mtime; /* Tiempo de la última modificación */
   time_t  st_ctime; /* Tiempo del último cambio en el inodo */
}
```

Definida en `<sys/stat.h>`

Acceso al manual con
`$ man 2 stat`

Función *stat*

- **Máscara de modo de un fichero (**campo `st_mode` de la estructura `stat`**):**



Función *stat*

- **Especificación del tipo de fichero:**
  - Mediante la máscara `S_IFMT`

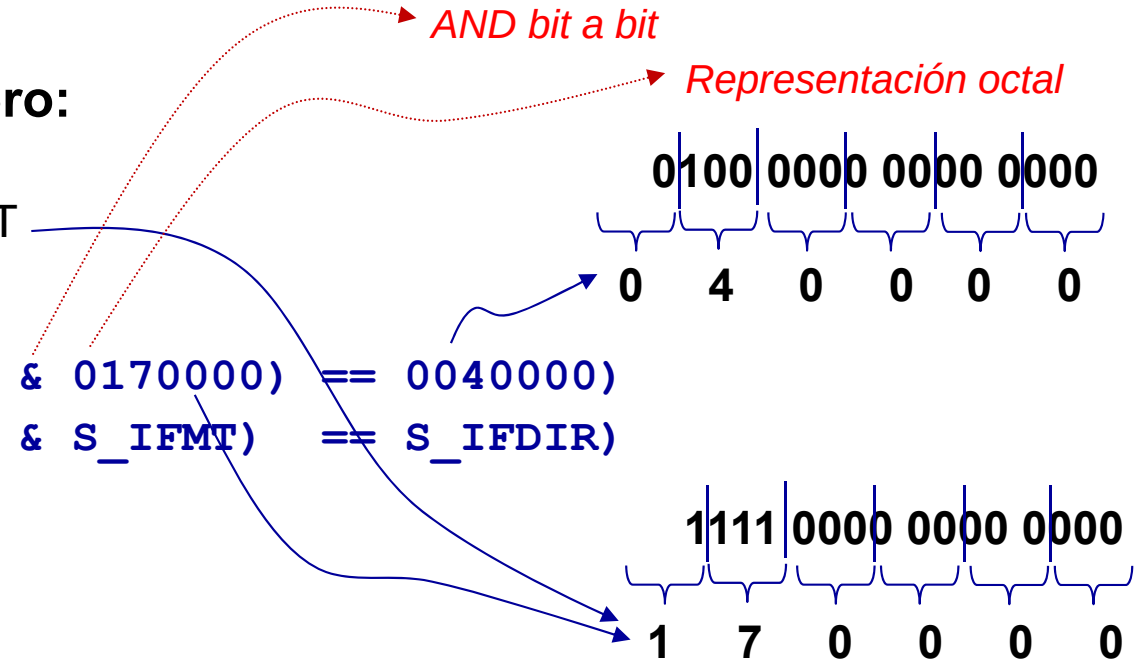
    Ejemplo:

```
if ((campo.st_mode & 0170000) == 0040000)
if ((campo.st_mode & S_IFMT)  == S_IFDIR)
```

  - Mediante las macros
    - `S_ISREG`
    - `S_ISDIR`
    - `S_ISFIFO`

    Ejemplo:

```
if (S_ISDIR(campo.st_mode))
```



Ejemplo

- **Ejemplo 3:**
  - Queremos ver ver si un fichero "f" es un fichero regular

```
struct stat datos;
stat("f", &campo);
```

- Opción 1:

```
if (  ( (0170000)&(campo.st_mode) ) == (0100000)  )
    printf("%s es un fichero regular \n", f);
```

- Opción 2:

```
if (  ( (S_IFMT)&(campo.st_mode) ) == (S_IFREG)  )
    printf("%s es un fichero regular \n", f);
```

- Opción 3:

```
if (  S_ISREG ( campo.st_mode )  )
  printf("%s es un fichero regular \n", f);
```

Ejemplo

- **Ejemplo 4:**

```
$ cat ej_stat.c
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <time.h>
#include <fcntl.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
  struct stat datos;
  struct tm *fecha;
  char tipofich;
  char *fichero2;

  if (argc != 2)
  { printf("Necesario fichero o directorio como parámetro\n");
    exit (-1);
  }

  if (stat(argv[1],&datos) != 0)
  { perror("Error en el stat");
    exit (-1);
  }
```

Ejemplo típico que muestra los atributos de un fichero

Ejemplo

- **Ejemplo 4 (cont.):**

```
printf("Nombre del fichero:  %s\n",argv[1]);
printf("Tamaño en bytes:     %d\n",datos.st_size);
switch(datos.st_mode & S_IFMT)
 {case S_IFREG:
      tipofich='R';
      break;
  case S_IFLNK:
      tipofich='L';
      break;
  case S_IFDIR:
      tipofich='D';
      break;
  }
printf("Tipo de fichero:     %c\n",tipofich);
printf("Número de enlaces:   %d\n",datos.st_nlink);
fecha=localtime(&(datos.st_mtime));
printf("Última modificación: ");
printf("%d/%d/%d\n",fecha->tm_mday,fecha->tm_mon+1,fecha->tm_year);
}
```

Enlace simbólico

```
printf("%s \n",ctime(&campo.st_mtime));
printf("%s \n",asctime(localtime(&campo.st_mtime)));
```

Enlaces duros

$ man localtime

O bien:

```
printf("%s \n",ctime(&campo.st_mtime));
printf("%s \n",asctime(localtime(&campo.st_mtime)));
```

Ejemplo

- **Ejemplo 4 (cont.):**

```
$ make ej_stat
$ ej_stat f
Nombre del fichero:  f
Tamaño en bytes:     32
Tipo de fichero:     R
Número de enlaces:   1
Última modificación: 9/10/113
$
```

Función *fstat*

- **Sintaxis:**

```
int fstat (int desc_fich, struct stat *datos);
```

  devuelve:

  - Si todo ha ido bien: 0
  - Si error: -1

- **Descripción:**
  - Análogo a `stat` pero partiendo de un fichero ya abierto identificado por su descriptor `desc_fich`
- **Ejemplo:**
  - `fstat (fd, &datos);`

Función *lstat*

- **Sintaxis:**

```
int lstat (const char *nombre, struct stat *datos);
```

devuelve:

  - Si todo ha ido bien: 0
  - Si error: -1

- **Descripción:**
  - Análogo a `stat` excepto cuando el nombre del fichero `nombre` corresponde a un enlace simbólico
    - `lstat`: Proporciona información sobre el fichero que sirve de enlace
    - `stat`: Proporciona información sobre el fichero al que apunta el enlace
- **Ejemplo:**
  - `lstat("/usr/castano/prueba", &datos);`

Función *lstat*

- **Ejemplo:**

```
$ ls -al > f
$ ln -s f f.lnk
$ ls -i f
9379 f
$ ls -i f.lnk
9380 f.lnk
```

  - **`stat`** de **`f`** ⟶ Información del fichero con inodo 9379
  - **`stat`** de **`f.lnk`** ⟶ Información del fichero con inodo 9379
  - **`lstat`** de **`f`** ⟶ Información del fichero con inodo 9379
  - **`lstat`** de **`f.lnk`** ⟶ Información del fichero con inodo 9380

Función *rename*

- **Sintaxis:**

```
int rename (const char *fuente, const char *destino);
```

  devuelve:

  - Si todo ha ido bien: 0
  - Si error: -1

- **Descripción:**
  - Cambia el nombre (y ubicación) del fichero `fuente` por `destino`
- **Ejemplo:**
  - `rename ("/usr/castano/prueba", "prueba2");`

Tema 4. Gestión del sistema de ficheros

UNIVERSITAT JAUME·I

Índice

- Definición de fichero
- Atributos de un fichero
- Estructura interna de los ficheros
- Métodos de acceso
- Operaciones básicas sobre ficheros
- El sistema de directorios
- Operaciones básicas sobre directorios
- Enlaces
- Protección de un fichero
- Servicios POSIX para gestión de ficheros
- Servicios POSIX para gestión de directorios

Servicios POSIX para gestión de directorios

- **Crear un directorio:** `mkdir`
- **Borrar un directorio:** `rmdir`
- **Abrir un directorio:** `opendir`
- **Cerrar un directorio:** `closedir`
- **Leer de un directorio:** `readdir`
- **Cambiar de directorio:** `chdir`
- **Cambiar nombre de un directorio :** `rename`

Función *mkdir*

- **Sintaxis:**

```
int mkdir(const char *nombre, mode_t modo);
```

devuelve:

  - Si todo ha ido bien: 0
  - Si error: -1

- **Descripción:**
  - Crea el directorio especificado en `nombre` con el modo de protección especificado en `modo`
- **Ejemplos:**
  - `mkdir (“tmp”,0755);`
  - `mkdir (“/usr/castano/tmp”,0755);`

Ejemplo

- **Ejemplo 5:**

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

int main(int argc, char * argv[])
{
  int i, fd;
  struct stat campo;

  stat(argv[1],&campo);

  if ( !S_ISDIR(campo.st_mode) & !S_ISREG(campo.st_mode) )
    {
       printf("Creo el directorio %s\n",argv[1]);
       mkdir(argv[1],0755);
    }
  exit(0);
}
```

Función *rmdir*

- **Sintaxis:**

```
int rmdir(const char *nombre);
```

  devuelve:

  - Si todo ha ido bien: 0
  - Si error: -1

- **Descripción:**
  - Borra el directorio especificado en `nombre` (si el directorio está vacío)
- **Ejemplos:**
  - `rmdir (“tmp”);`
  - `rmdir (“/usr/castano/tmp”);`

Ejemplo

- **Ejemplo 6:**

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

int main (int argc, char * argv[])

{ struct stat campo;

  stat(argv[1],&campo);
  switch(campo.st_mode & S_IFMT)
  {  case S_IFREG: printf("Borro el fichero %s\n",argv[1]);
                   remove(argv[1]);
                   break;
     case S_IFDIR: printf("Borro el directorio %s\n",argv[1]);
                   rmdir(argv[1]);
                   break;
     default:      printf("%s no es fichero ni directorio\n",argv[1]);
                   exit(-1);
  }
  exit(0);
}
```

Función *opendir*

- **Sintaxis:**

```
DIR * opendir(const char *nombre);
```

devuelve:

  - Un puntero a una estructura de tipo `DIR` (definido en `<dirent.h>`) que identificará al directorio a partir de entonces
  - Si error: NULL y deja en `errno` el código del error

- **Descripción:**
  - Abre el directorio especificado en `nombre`
- **Ejemplos:**
  - `dir = opendir (“tmp”);`
  - `dir = opendir (“/usr/castano/tmp”);`

Función *closedir*

- **Sintaxis:**

```
int closedir(DIR *dirp);
```

  devuelve:

  - Si todo ha ido bien: 0
  - Si error: -1

- **Descripción:**
  - Cierra el fichero directorio identificado a través de `dirp`
- **Ejemplo:**
  - `closedir (dirp);`

Función *readdir*

- **Sintaxis:**

```
struct dirent *readdir(DIR *dirp);
```

  devuelve:

  - Una entrada del directorio a través de una estructura de tipo `dirent` (definida en `<dirent.h>`)
  - 0 si se ha llegado al final de directorio o si error

- **Descripción:**
  - Lee la entrada (registro) en la que se encuentra situado el puntero de lectura/escritura del directorio identificado a través de `dirp`
  - Modifica la posición del puntero
- **Ejemplo:**
  - `entrada_dir = readdir (dirp);`

Función *readdir*

- **Estructura *dirent*:**

```
struct dirent
{
  ino_t  d_ino;    /* Inodo asociado a la entrada del directorio */
   short  d_reclen; /* Longitud de la entrada */
   short  d_namlen; /* Longitud de la cadena que hay en “d_name” */
   char   d_name[_MAXNAMLEN+1]; /* Nombre del fichero */
}
```

Ejemplo

- **Ejemplo 7:**

```
$ cat ej_recorre_dir.c
#include <unistd.h>
#include <stdlib.h>
#include <dirent.h>
#include <sys/types.h>
#include <sys/stat.h>

main(int argc, char *argv[])
{
char fichero[256];
struct stat datos;
DIR *dir;
struct dirent *dp;
int estado;

stat(argv[1],&datos);
if (!S_ISDIR(datos.st_mode))
  { printf("%s no es un directorio\n",argv[1]); exit (-1); }
```

Ejemplo típico que muestra el contenido de un directorio

Ejemplo

- **Ejemplo 7 (cont.):**

```
if ((dir=opendir(argv[1])) == NULL)
  { perror("opendir"); exit (-1);}

while ((dp=readdir(dir)) != NULL)
{ sprintf(fichero,"%s/%s",argv[1],dp->d_name);
  if ((stat(fichero,&datos)) != 0)
    { perror("stat"); exit (-1);}
  printf("%s (%d bytes)\n",fichero,datos.st_size);
}
closedir(dir);
exit (0);
}
```

Crea ruta de acceso a los ficheros del directorio

Necesario para que encuentre los ficheros y que stat no falle

Ejemplo

- **Ejemplo 7 (cont.):**

```
$ make ej_recorre_dir
$ ej_recorre_dir .
./. (0 bytes)
./.. (0 bytes)
./dup1 (5966 bytes)
./wc_de_fichs.c (760 bytes)
./ej_recorre_dir (5688 bytes)
./ej_mkdir.c (379 bytes)
$
```

Función *rename*

- **Sintaxis:**

```
int rename (const char *fuente, const char *destino);
```

  devuelve:

  - Si todo ha ido bien: 0
  - Si error: -1

- **Descripción:**
  - Cambia el nombre (y ubicación) del directorio `fuente` por `destino`
  - Si `destino` es un directorio no vacío, no se realiza la operación y se devuelve error
- **Ejemplo:**
  - `rename ("/usr/castano/tmp", "/tmp/castano");`

Función *chdir*

- **Sintaxis:**

```
int chdir(const char *nombre);
```

  devuelve:

  - Si todo ha ido bien: 0
  - Si error: -1

- **Descripción:**
  - Cambia el directorio de trabajo actual por el directorio especificado en `nombre`
- **Ejemplo:**
  - `chdir ("/usr/castano/tmp");`

Ejemplo

- **Ejemplo 8:**

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

int main (int argc, char *argv[])
{ int i, fd;
  struct stat campo;
  stat(argv[1],&campo);
  if ( S_ISDIR(campo.st_mode) )
    {
       printf("Cambio al directorio %s\n",argv[1]);
       chdir(argv[1]);
    } else {
       printf("Creo el directorio %s\n",argv[1]);
       mkdir(argv[1],0755);
    }
  exit(0);
}
```

Ejercicios

- **Ejercicio 1:**

  ¿Qué saldrá por pantalla tras compilar el programa `fich.c` y ejecutarlo con los parámetros que a continuación se indica?

```
$cat fich.c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

int main(int argc, char * argv[])
{ int i, fd; char c;

  fd=creat(argv[1],0755);
  for (i=1; i<argc; i++) write(fd,argv[i],1);
  close(fd);

  fd=open(argv[1],0);
  while (read(fd,&c,1) > 0)
  {   printf("%c",c);
      lseek(fd,1,1);
  }
  printf("\n"); close(fd); exit (0);
}
$make fich; fich Pepe Esto es una frase
```

Ejercicios

- **Ejercicio 1 (solución):**

```
$make fich; fich Pepe Esto es una frase
Pef
$ cat Pepe
PEeuf$
```

¿Sorprendente? Presta atención al tamaño del dato que se escribe en cada operación de escritura

Ejercicios

- **Ejercicio 2:**

  Contesta a las siguientes preguntas a partir del código `ej_punteros.c` que aparece a continuación:

  a) ¿Cuál es el resultado de ejecutar dicho código si se le pasa como argumento el fichero `prueba`, cuyo contenido es el siguiente:

  ```
  Esto es una prueba. Esto quién lo imprime. Esto lo imprimo yo.
  ```

  b) Muestra el valor de la variable `fd`, así como el contenido de la tabla de ficheros abiertos de cada proceso, de la tabla de ficheros abiertos en el sistema y de la tabla de inodos una vez creados los procesos y antes que estos ejecuten la función `close`.

  c) ¿En qué varía la ejecución si se elimina del código la función `wait`?

Ejercicios

- **Ejercicio 2 (cont.):**

```
$ cat ej_punteros.c
#include<unistd.h>
#include<fcntl.h>
#include<stdio.h>
#include <stdlib.h>

int main(int argc, char ** argv)
{
   int n, fd;
   char buffer[15];

   fd = open (argv[1],O_RDONLY);
   n = read (fd, buffer, 10);
   buffer[n] = 0;
   printf("Datos leidos desde el fichero por proceso %d: %s\n\n",
           getpid(), buffer);
```

Ejercicios

- **Ejercicio 2 (cont.):**

```
if (fork() != 0)
{
  fd =  open (argv[1],O_RDONLY);
  wait (NULL);
  while ((n=read(fd,buffer,10))>0)
  {  buffer[n] = 0;
     printf("Datos leidos desde el fichero por proceso %d: %s\n",
             getpid(), buffer);
  }
} else {
  while ((n=read(fd,buffer,10))>0)
  {  buffer[n] = 0;
     printf("Datos leidos desde el fichero por proceso %d: %s\n",
             getpid(), buffer);
  }
}
close (fd);
exit (0);
}
```

Ejercicios

- **Ejercicio 2 (solución):**

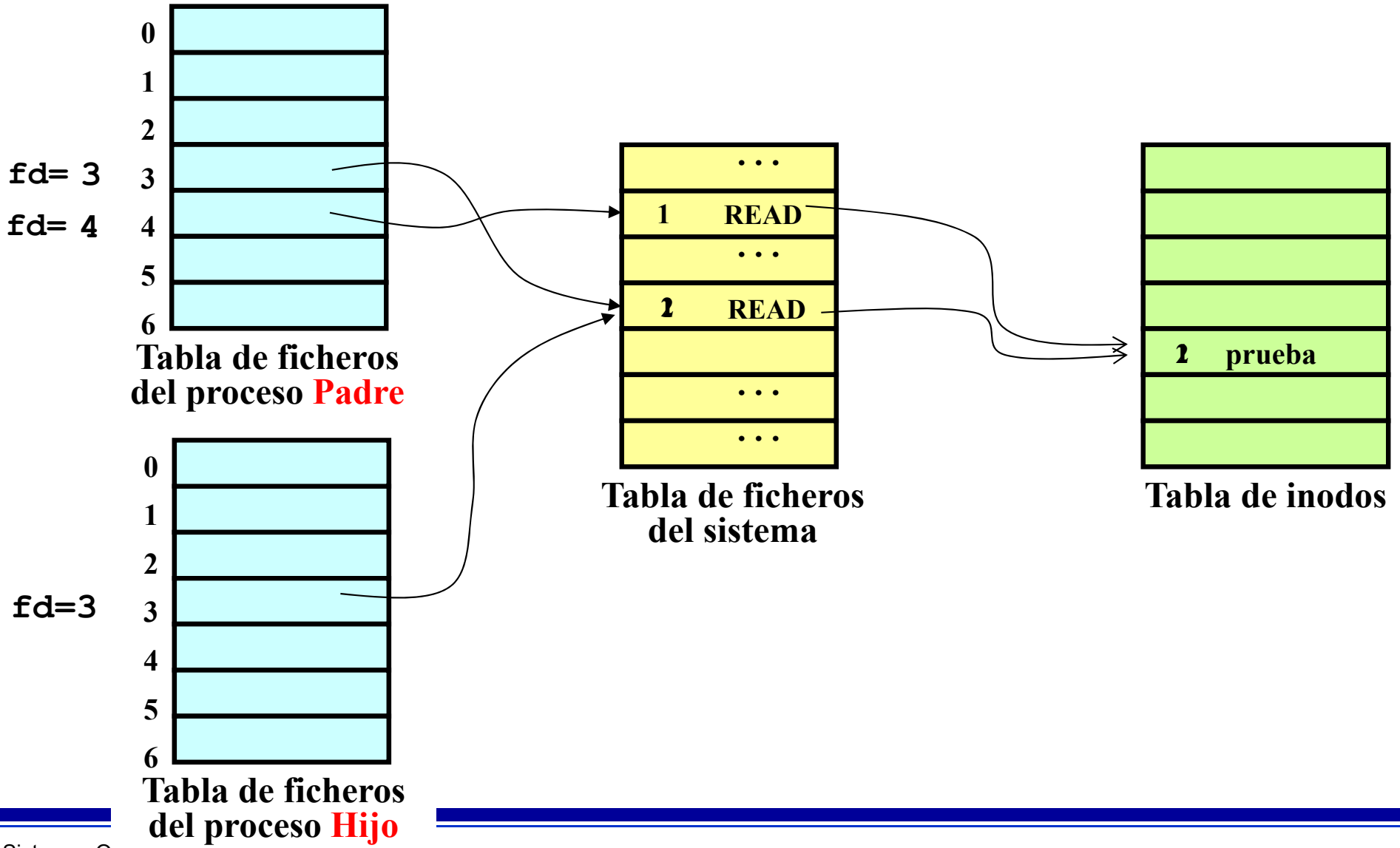
  a) El hijo hereda la tabla de ficheros abiertos por el proceso padre y, por tanto, hereda también el puntero de lectura/escritura del fichero `prueba`. Tras crear al hijo, el padre abre de nuevo ese fichero y, consecuentemente, utilizará un puntero nuevo para acceder a él. Así pues, la salida del programa será:

```
Datos leidos desde el fichero por proceso 13528: Esto es un

Datos leidos desde el fichero por proceso 13529: a prueba.
Datos leidos desde el fichero por proceso 13529: Esto quién
Datos leidos desde el fichero por proceso 13529:  lo imprim
Datos leidos desde el fichero por proceso 13529: e. Esto lo
Datos leidos desde el fichero por proceso 13529:  imprimo y
Datos leidos desde el fichero por proceso 13529: o.

Datos leidos desde el fichero por proceso 13528: Esto es un
Datos leidos desde el fichero por proceso 13528: a prueba.
Datos leidos desde el fichero por proceso 13528: Esto quién
Datos leidos desde el fichero por proceso 13528:  lo imprim
Datos leidos desde el fichero por proceso 13528: e. Esto lo
Datos leidos desde el fichero por proceso 13528:  imprimo y
Datos leidos desde el fichero por proceso 13528: o.
```

Ejercicios

- **Ejercicio 2 (solución):**

b)



Ejercicios

- **Ejercicio 2 (solución):**

  c) Los procesos leerían la misma información porque, cuando el padre vuelve a abrir el fichero, no comparte el puntero de lectura/escritura con el hijo. Pero, se podría mezclar la salida de ambos procesos. Además, si el padre finaliza antes que el hijo, mostraría en la pantalla el prompt antes de que el hijo acabase de mostrar sus datos.

```
Datos leidos desde el fichero por proceso 13615: Esto es un

Datos leidos desde el fichero por proceso 13615: Esto es un
Datos leidos desde el fichero por proceso 13615: a prueba.
Datos leidos desde el fichero por proceso 13615: Esto quién
Datos leidos desde el fichero por proceso 13615:  lo imprim
Datos leidos desde el fichero por proceso 13615: e. Esto lo
Datos leidos desde el fichero por proceso 13615:  imprimo y
Datos leidos desde el fichero por proceso 13615: o.

Datos leidos desde el fichero por proceso 13616: a prueba.
$ Datos leidos desde el fichero por proceso 13616: Esto quién
Datos leidos desde el fichero por proceso 13616:  lo imprim
Datos leidos desde el fichero por proceso 13616: e. Esto lo
Datos leidos desde el fichero por proceso 13616:  imprimo y
Datos leidos desde el fichero por proceso 13616: o.
```

Ejercicios

- **Ejercicio 3:**

  Realizar un programa que muestre el número de líneas (comando **`wc -l`**) de cada uno de los ficheros regulares de un determinado directorio que se le pasa como argumento al programa.

Ejercicios

- **Ejercicio 3 (solución):**

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <dirent.h>
#include <sys/types.h>
#include <sys/stat.h>

main(int argc, char *argv[])
{
char fichero[256];
struct stat datos;
DIR *dir;
struct dirent *dp;
int estado;

stat (argv[1],&datos);
if (!S_ISDIR(datos.st_mode))
  { printf("%s no es un directorio\n",argv[1]); exit (-1); }
```

Ejercicios

- **Ejercicio 3 (solución):**

```
if ((dir=opendir(argv[1])) == NULL)
  { perror("opendir"); exit (-1);}

while ((dp=readdir(dir)) != NULL)
{ sprintf(fichero,"%s/%s",argv[1],dp->d_name);
  if ((stat(fichero,&datos)) != 0)
    { perror("stat"); exit (-1);}
  if ( S_ISREG(datos.st_mode))
  { if (fork()!=0) wait(&estado);
    else {
           execlp("wc","wc","-l",fichero,NULL);
           perror("wc"); exit(-1);
         }
  }
}
closedir(dir);
exit (0);
}
```

Ejercicios

- **Ejercicio 4:**

  Write a program in C lenguaje so that it executes the following shell commands for each of the subdirectories that a directory contains at its first level:

  ```
  ls -l subdir | grep ^d
  ```

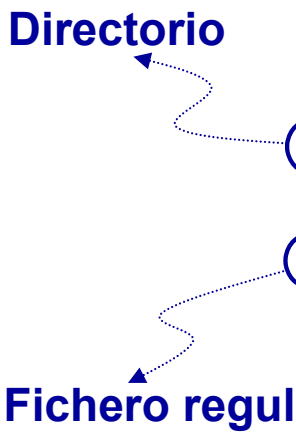
  The directory is given as an argument to the program and, threrefore, it should be navigated by using the corresponding system calls. Notice that recursion is not required to search for the subdirectories.

  This could be a possible output dumped on the screen by the program:

```
$ ls_grep dir1
SUDIRECTORIES OF DIRECTORY dir1/d1:
drwx-w---- 5 castano uji    0 Nov 23 15:04 d1.1
drwx-w---- 7 castano uji    0 Nov 23 15:00 d1.2
SUDIRECTORIES OF DIRECTORY dir1/d2:
SUDIRECTORIES OF DIRECTORY dir1/d3:
drwx-w---- 2 castano uji    0 Nov 21 21:54 d3.1
drwx-w---- 2 castano uji    0 Nov 20 13:59 d3.2
drwx-w---- 2 castano uji    0 Nov 20 13:59 d3.3
$
```

Atributos de un fichero

```
$ ls –l
total 636
-rw------- 1 castano uji  10354 Oct 25 18:24 assignment5.tex
-rw------- 1 castano uji   1330 Nov 11 10:04 books.db
drwx-w---- 2 castano uji      0 Nov  8 18:53 d1
drwx-w---- 2 castano uji      0 Nov  8 18:53 d2
-rwx------ 1 castano uji   5871 Sep  6 12:16 escribe_pid
-rw------- 1 castano uji   1027 Sep  6 12:16 escribe_pid.c
$
```




```
ls –l | grep ^d

ls –l | grep ^-
```

Ejercicios

- **Ejercicio 4 (solución):**

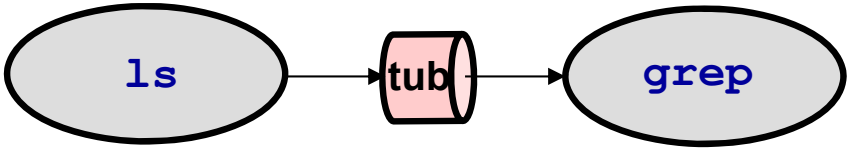
```
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <dirent.h>
#include <unistd.h>
#include <stdlib.h>

main(int argc, char *argv[])
{
struct stat buf;
DIR *dir;
struct dirent *dp;
char * fichero;
int tub[2];

if (argc != 2) {printf("Argumentos incorrectos.\n"); exit(-1);}

if ( (stat(argv[1],&buf)==0)  && (S_ISDIR(buf.st_mode)) )
{
```

Ejercicios

- **Ejercicio 4 (solución):**

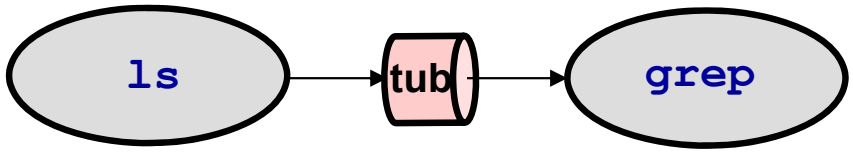
```
dir=opendir(argv[1]);
while ((dp=readdir(dir)) != NULL)
{
  if (strcmp(dp->d_name,".") && strcmp(dp->d_name,".."))
 {
    fichero=(char *)malloc(strlen(argv[1])+strlen(dp->d_name)+1);
    sprintf(fichero,"%s/%s",argv[1],dp->d_name);
    stat(fichero,&buf);
    if (S_ISDIR(buf.st_mode))
       if (fork()==0)
       {  pipe(tub);
          if (fork()==0)
          {   close(1);
              dup(tub[1]);
              close(tub[1]);
              close(tub[0]);

              execlp("ls","ls","-l",fichero,NULL);
              perror("ls");
              exit(-1);
          } else
```

No vale sizeof



Ejercicios

- **Ejercicio 4 (solución):**




```
                {
                    close(0);
                    dup(tub[0]);
                    close(tub[0]);
                    close(tub[1]);

                    printf("SUBDIRS OF DIR %s: \n",fichero);
                    execlp("grep","grep","^d",NULL);
                    perror("grep");
                    exit(-1);
                }
            } else wait(NULL);
        free(fichero);
     }
    }
    closedir(dir);
    exit(0);
} else { printf("El directorio %s no existe\n",argv[1]); exit(-1); }
}
```

Ejercicios

- **Ejercicio 4 (solución):**

  Nótese que es preciso incluir la primera función `fork`. Con esta función se crea un proceso hijo que a su vez crea otro hijo. Estos dos últimos procesos procesos (padre e hijo) ejecutan, respectivamente, los comandos `grep` y `ls` para un subdirectorio. De esta manera el proceso abuelo puede continuar recorriendo el directorio pasado como argumento. Si no se ejecutase ese `fork`, perderíamos el código del programa y este no se ejecutaría de forma correcta.

  Además, el proceso abuelo ha de esperar a que se muestre la salida que proporciona el `grep` para el subdirectorio que se está tratando antes de continuar con la siguiente entrada del directorio. Si eliminásemos la función `wait` se podría mezclar la salida que proporciona el programa para los diferentes subdirectorios.

Ejercicios

- **Ejercicio 4 (solución):**

  Y, finalmente, cabe llamar la atención sobre la operación efectuada por la función `sprintf`. Para realizar una operación `stat` sobre un fichero o subdirectorio de un directorio, debemos indicar dónde se encuentra dicho fichero o subdirectorio: dentro del directorio. En otro caso, la función `stat` no lo encontraría.

Ejercicios

- **Ejercicio 5:**

  ¿Qué ocurriría si se realizasen las siguientes operaciones?

```
$ln f f.hard
$ln –s f f.soft
$rm f
$cat f.hard
```

  ¿Y si a continuación se ejecuta el siguiente comando?

```
$cat f.soft
```

Ejercicios

- **Ejercicio 6:**

  ¿Qué ocurriría si se realizasen las siguientes operaciones?

```
$ln f f.hard
$ln –s f f.soft
$mv f.soft ..
$cat f.hard
```

  ¿Y si a continuación se ejecuta el siguiente comando?

```
$cat ../f.soft
```