

Sistema de Archivos de Unix

LABORATORIO DE SISTEMAS DE OPERACIÓN I
(ci-3825)

Prof. Yudith Cardinale

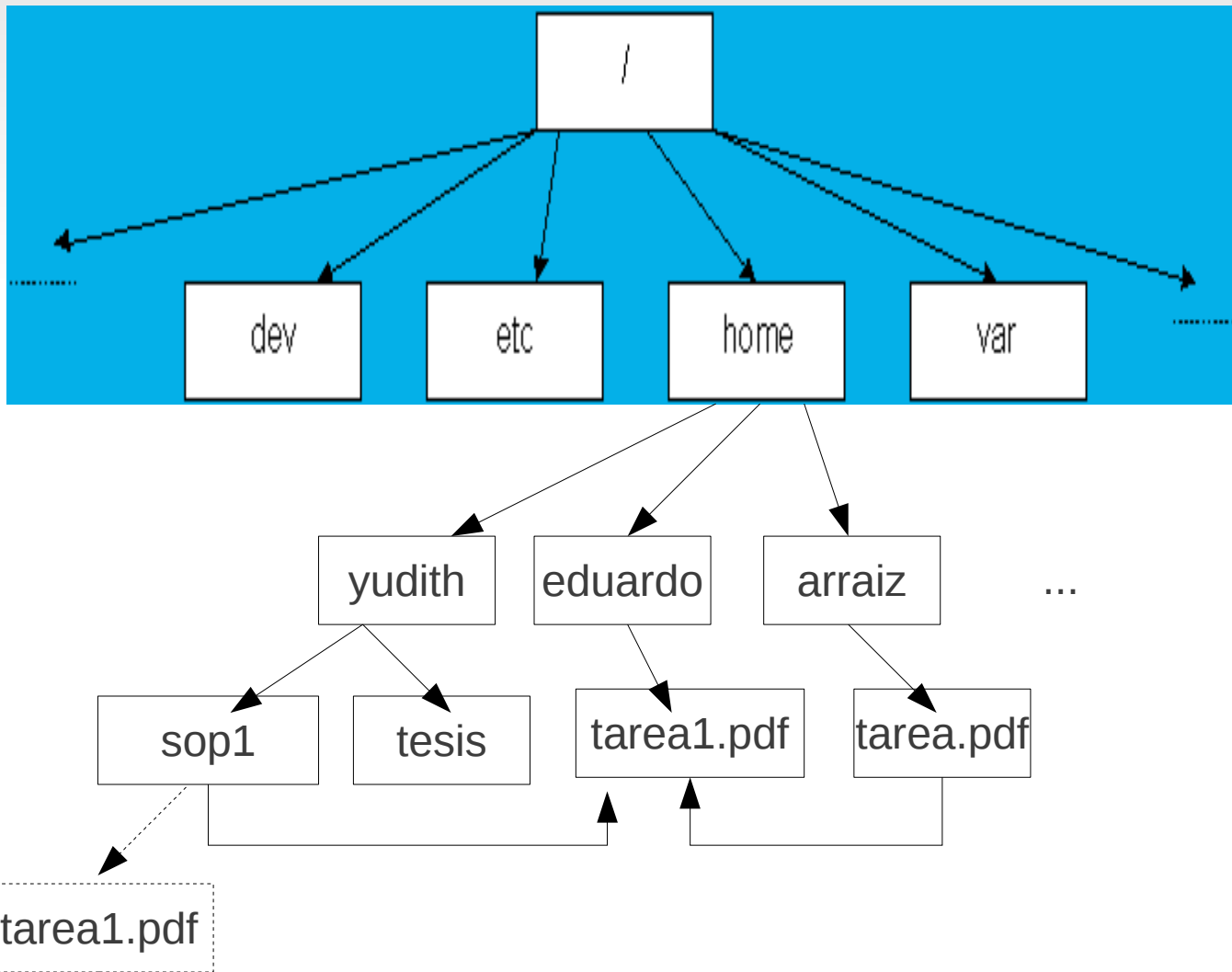
Enero-marzo 2011

Tipos de Archivos

- En la mayoría de los sistemas de operación, los datos y programas ejecutables están organizados en archivos para garantizar un almacenamiento permanente.
- El sistema de archivos de UNIX es una estructura jerárquica tipo árbol de directorios y archivos. Los nodos de este árbol pueden ser de tres tipos:
 - Archivos tipo directorios
 - Archivos tipo regulares
 - Archivos especiales:
 - Caracter
 - Bloque
 - Pipes
 - Sockets
 - Enlace

Jerarquía de directorios

- En UNIX el directorio raíz es "/" y la relación entre los archivos define una estructura de árbol o de grafo.
- Ejemplo:



- /etc contiene información acerca de la red, de usuarios, del filesystem, etc.
- /dev contiene información de los dispositivos
- /var contiene archivos del sistema
- /home para directorios de usuarios...

Caminos absolutos y relativos

- El camino absoluto es aquel que unívocamente especifica un archivo. Contiene todos los nodos en la jerarquía que se encuentran desde la raíz hasta el archivo. Comienzan con `"/`.

Ejemplo:

```
$> cd /home/yudith/sop1/
```

- Si un *pathname* no comienza con `"/`, se supone que el nombre comienza a partir del directorio actual de trabajo, estos *pathnames* se llaman caminos relativos. Ejemplo:

Si el directorio actual es `/home/yudith/`:

```
/home/yudith$>pwd
```

```
/home/yudith/
```

```
/home/yudith$> lpr -Pxerox sop1/tarea1.pdf
```

Directorios . y ..

- El directorio "." se refiere al directorio actual (*current directory*).

Ejemplo: `$> ./tarea Casos.in`

- El directorio ".." se refiere al directorio padre del actual (el que está por encima en la jerarquía)

Ejemplo: `$> cd ..`

- Algunos comandos para manipulación de archivos:
 - `ls`, `cat`, `cd`, `pwd`, `mkdir`, `rmdir`, `rm -r dirName`, ...
 - `quota -v`: muestra cuál es el espacio de disco máximo asignado a la cuenta
 - `df`: reporta el espacio libre del sistema de archivos
 - `du`: reporta la cantidad de KB usados por subdirectorio.
 - `file nombreArchivo`: muestra de que tipo es el archivo.

Comandos de manipulación de archivos

- ls, cat, cd, pwd, mkdir, rmdir, rm -r dirName, ...
- quota -v: muestra cuál es el espacio de disco máximo asignado a la cuenta
- df: reporta el espacio libre del sistema de archivos
- du: reporta la cantidad de KB usados por subdirectorio.
- file nombreArchivo: muestra de que tipo es el archivo.
- Chmod: cambia los permisos de un archivo. Sólo lo puede ejecutar el propietario del archivo o el superusuario

Cada archivo tiene asociado 9 bits que definen su tipo y la permisología

-rwxrwxrwx

Tipo de archivo
-: regular
d: directorio
c,b,p,l,s:
especiales

Permisos del
propietario (u)

Permisos de
otros (o)

Permisos del
grupo (g)

Ejemplos:

```
$> chmod go-rwx tarea1.pdf
```

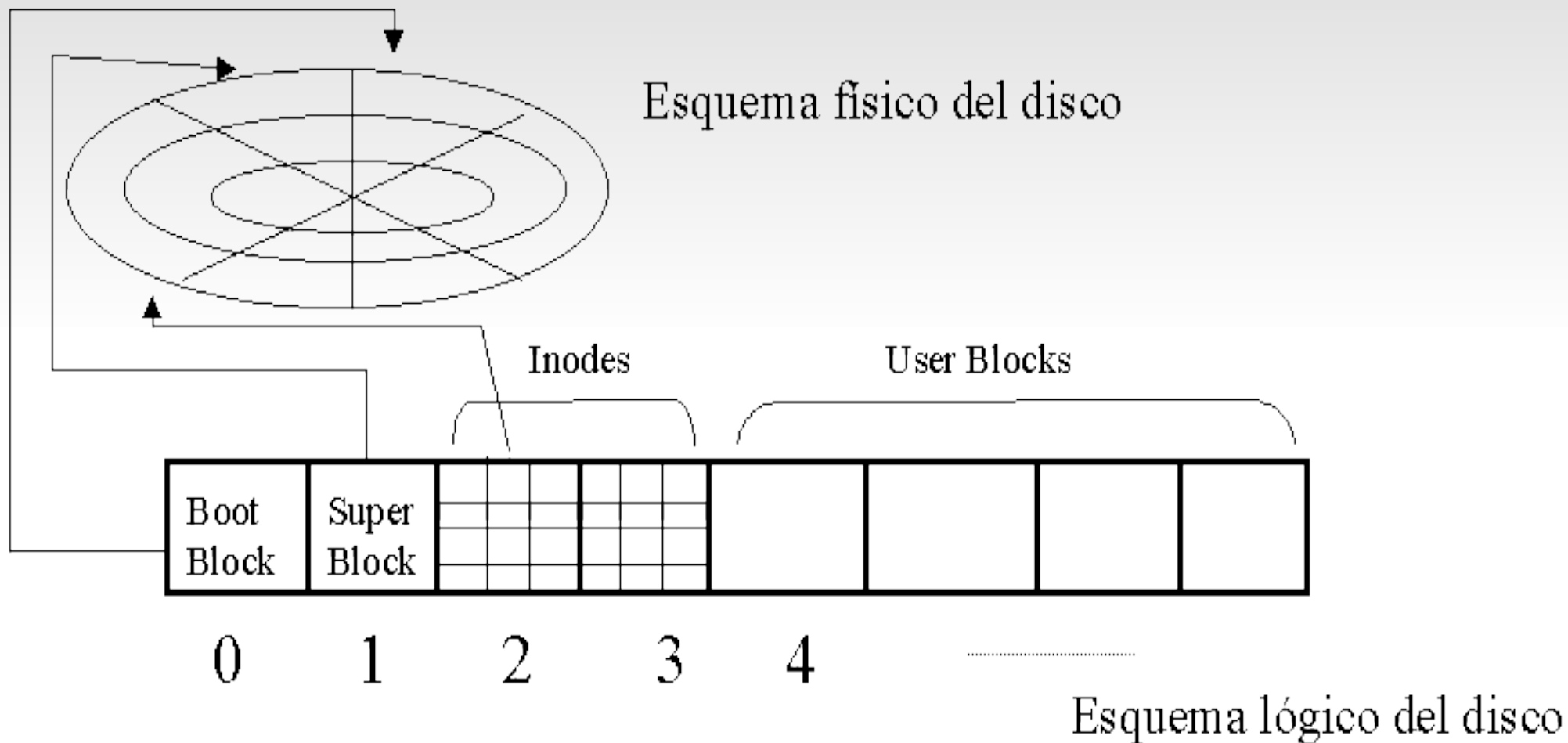
```
$> chmod a-xg+rw myfile
```

```
$> chmod a+x myscript
```

```
$> chmod 777 myfile2
```

Representación interna del FS

- Un sistema de archivos reside en disco y tradicionalmente tiene la siguiente estructura:

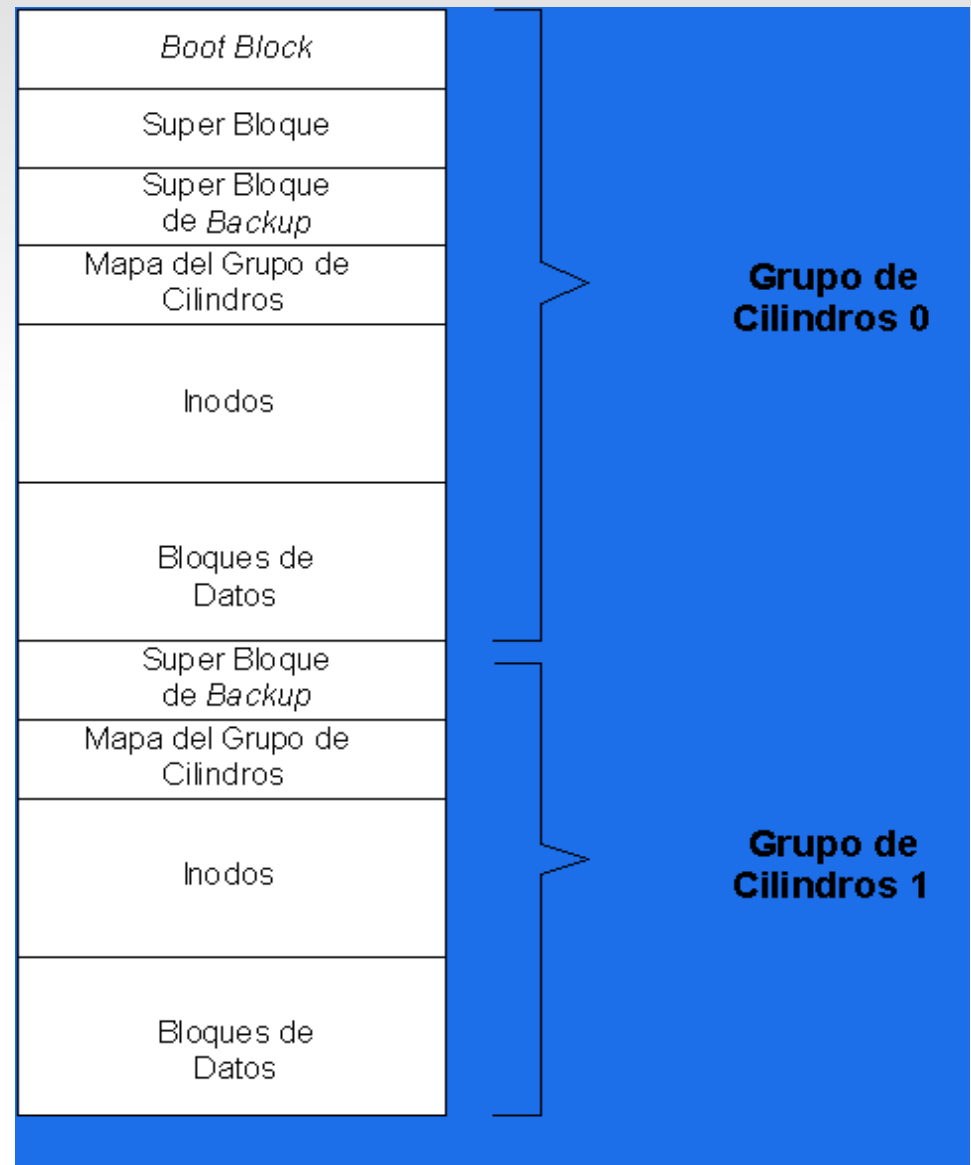
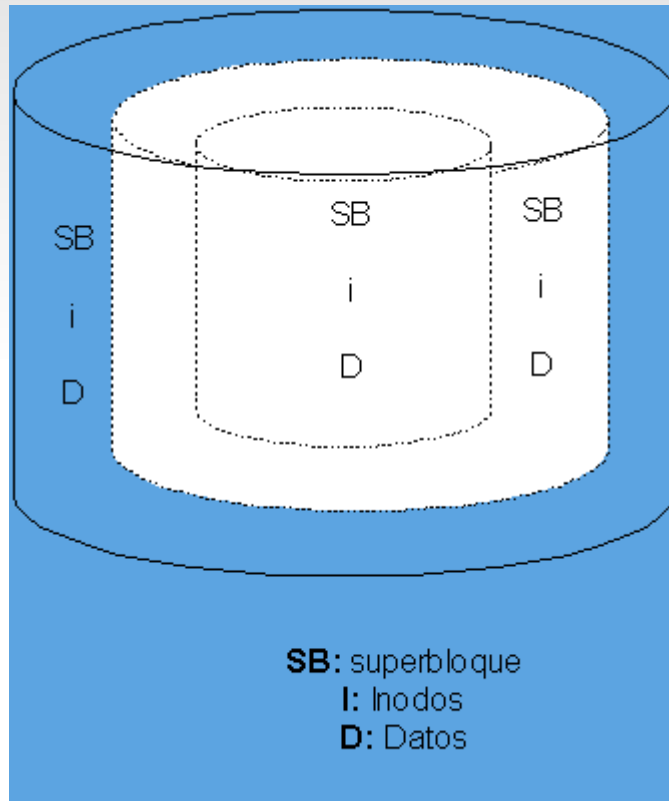


Representación interna del FS

- **Boot block:** Se encuentra en el **bloque 0** del disco y contiene código ejecutable (*bootstrap code*) que se carga en memoria principal durante el arranque del sistema.
- **Superblock:** Se encuentra en el **bloque 1** del disco y contiene información que describe el sistema de archivos:
 - Bitmap de bloques libres
 - Nro. total de bloques en disco
 - Nro. de i-nodos libres
 - Tamaño de un bloque (en bytes)
 - Nro. de bloques libres y usados, etc.
- **I-nodos (*index-node*):** son la estructura más importante del FS. Cada archivo en el sistema está representado por un i-nodo. Contiene información referente al archivo que representa: tamaño, apuntadores a los bloques de datos, tipo de archivo, fechas de creación, modificación y último acceso, *user id*, etc.
- **Bloques de datos:** alojan los datos que contienen los archivos.

Representación interna del FS

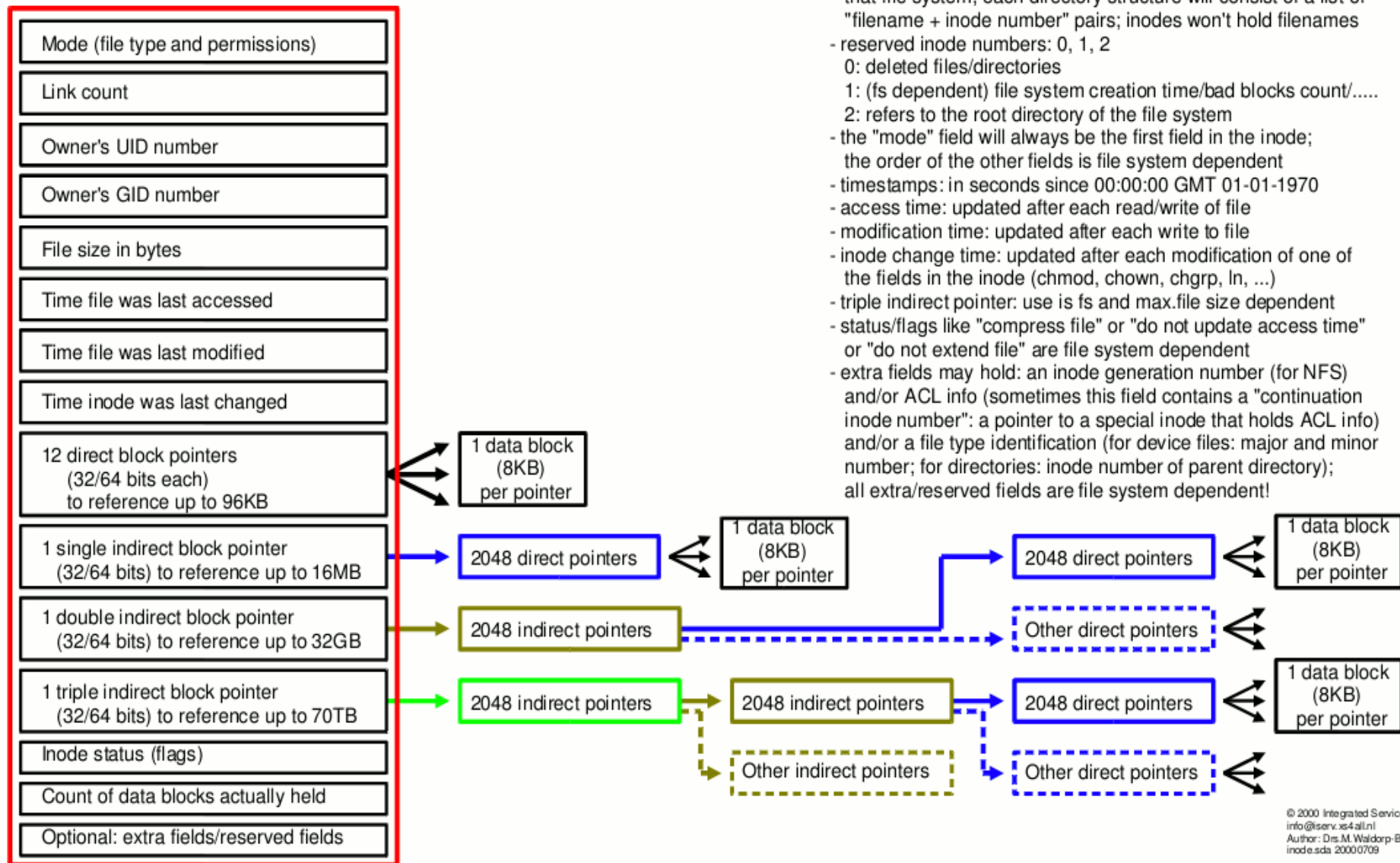
- Muy frecuentemente el sistema de archivos se organiza en particiones por seguridad y para ofrecer mejores tiempos de acceso.



Estructura de los i-nodos

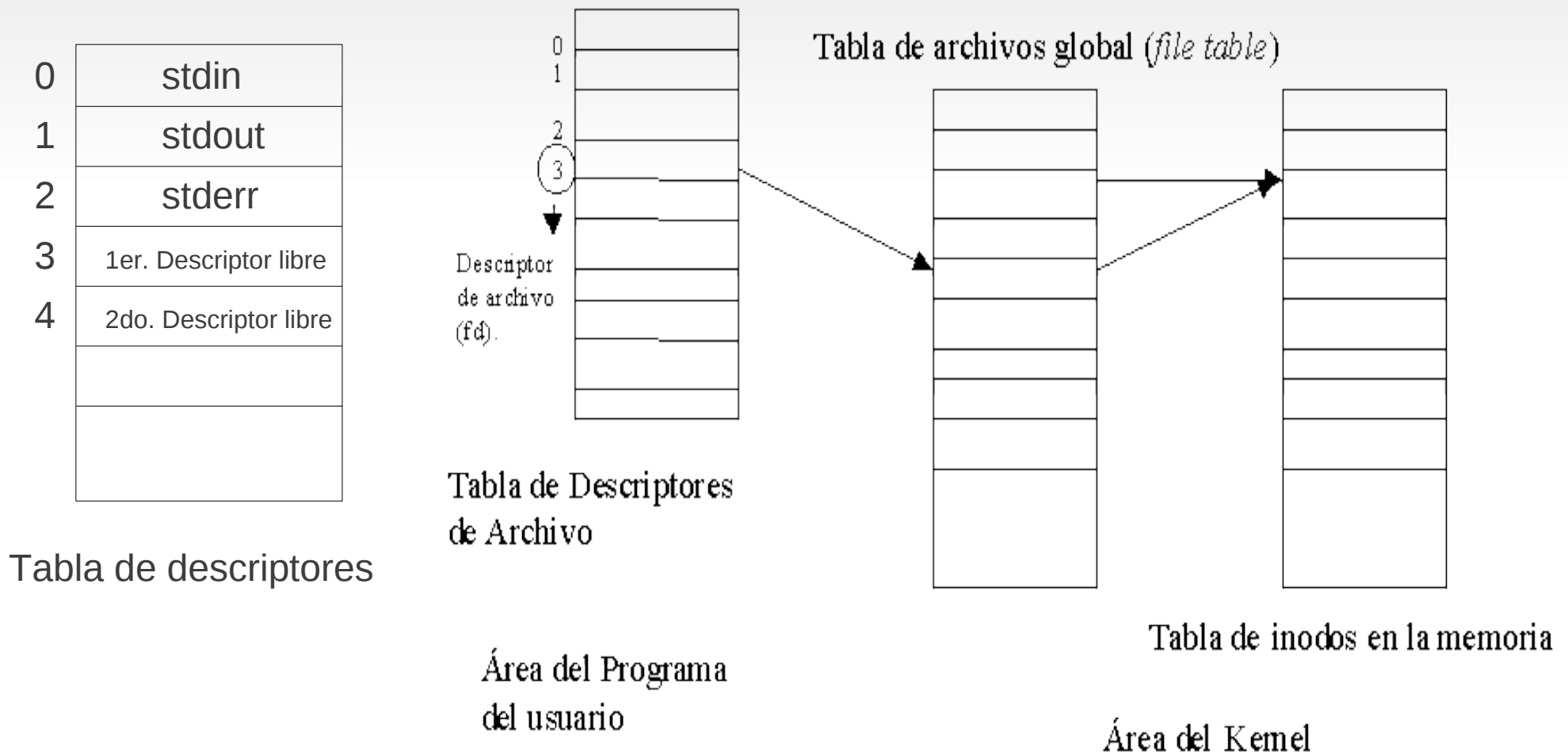
- El tamaño de cada i-nodo es de aprox. 128 bytes y contiene
 - Información general del archivo que representa
 - Vector de 12 direcciones directas a bloques de datos
 - Vector de direcciones de un nivel de indirección
 - Vector de direcciones de dos niveles de indirección
 - Vector de direcciones de tres niveles de indirección
- Los i-nodos son identificados con números consecutivos comenzando en 1.
 - **ls -i** muestra el número de i-nodo asociado a los archivos en el directorio.
- El i-node 1 se reserva para un archivo que contiene solamente bloques malos.
- Cada archivo tiene asociado un solo i-nodo

UNIX INODE STRUCTURE



Tablas internas manejadas por el FS

- Tabla de descriptores: una por proceso
- Tabla de archivos global (*file table*): una por sistema.
- Tabla de i-nodos: una por sistema



Llamadas al sistema para manipular archivos

- Llamadas estándar de Unix: proveen buffering
- Librerías: `<unistd.h>` , `<sys/types.h>`, `<sys/stat.h>`,`<fcntl.h>`
- Llamadas al sistema: se define id del archivo: `int fd`;
 - `int open(char *path, int oflag [, int permissions])`
 - Oflag: `O_RDONLY`, `O_WRONLY`, `O_RDWR`, `O_APPEND`, `O_CREAT`, `O_EXCL`, `O_TRUNC`
 - Ejemplo: `fd=open(filename,O_CREAT | O_RDWR,0600)`
 - `int read(int fd, char *buf, int count)`, `int write(int fd, char *buf, int count)`
 - `int close(int fd)`
 - `long lseek(int fd, long offset, int mode)`: para posicionar el apuntador
 - Mode: `SEEK_SET`, `L_SET`: relativo al comienzo del archivo.
 - `SEEK_CUR`, `L_CUR`: relativo a la posición actual.
 - `SEEK_END`, `L_END`: relativo al final del archivo
 - `int link(char *oldPath, char *newPath)`: crea un enlace físico (*hard link*)
 - `int unlink(char *fileName)`

Llamadas al sistema para manipular archivos

- Llamadas estándar de librerías de C: son más lentas que las de Unix pero permiten archivos de datos
- Librería: `<stdio.h>`
- Llamadas al sistema: se define id del archivo: `FILE *fd;`
 - `FILE *fopen(char *path, char * mode)`
 - mode: `r, w, a, r+, w+`
 - Ejemplo: `fd=fopen(filename,"r")`
 - `size_t fread(void *ptr, size_t size, size_t nitems, FILE *fd)`
 - `size_t fwrite(void *ptr, size_t size, size_t nitems, FILE *fd)`
 - `int fclose(FILE + fd)`
 - `int fseek(FILE * fd, long offset, int whence)`: para posicionar el apuntador
 - Mode: `SEEK_SET, L_SET`: relativo al comienzo del archivo.
 - `SEEK_CUR, L_CUR`: relativo a la posición actual.
 - `SEEK_END, L_END`: relativo al final del archivo

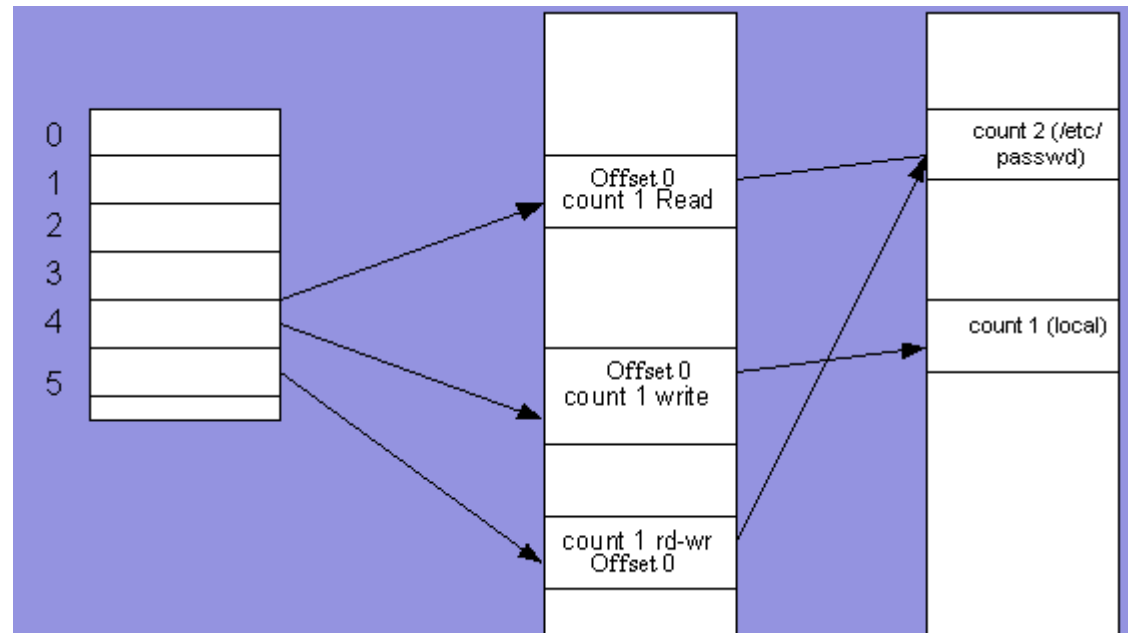
Llamadas al sistema para manipular archivos

- Cada vez que se realizan operaciones sobre archivos, puede haber modificaciones en las 3 tablas del Sistema de Archivos: Tabla de descriptores, Tabla Global de Archivos, Tabla de i-nodos.

Ejemplos:

Suponga que un proceso P1 realiza las siguientes operaciones:

```
int fd1, fd2, fd3;  
fd1= open("/etc/passwd", O_RDONLY);  
fd2 =open("local", O_WRONLY);  
fd3 =open("/etc/passwd", O_RDWR);
```

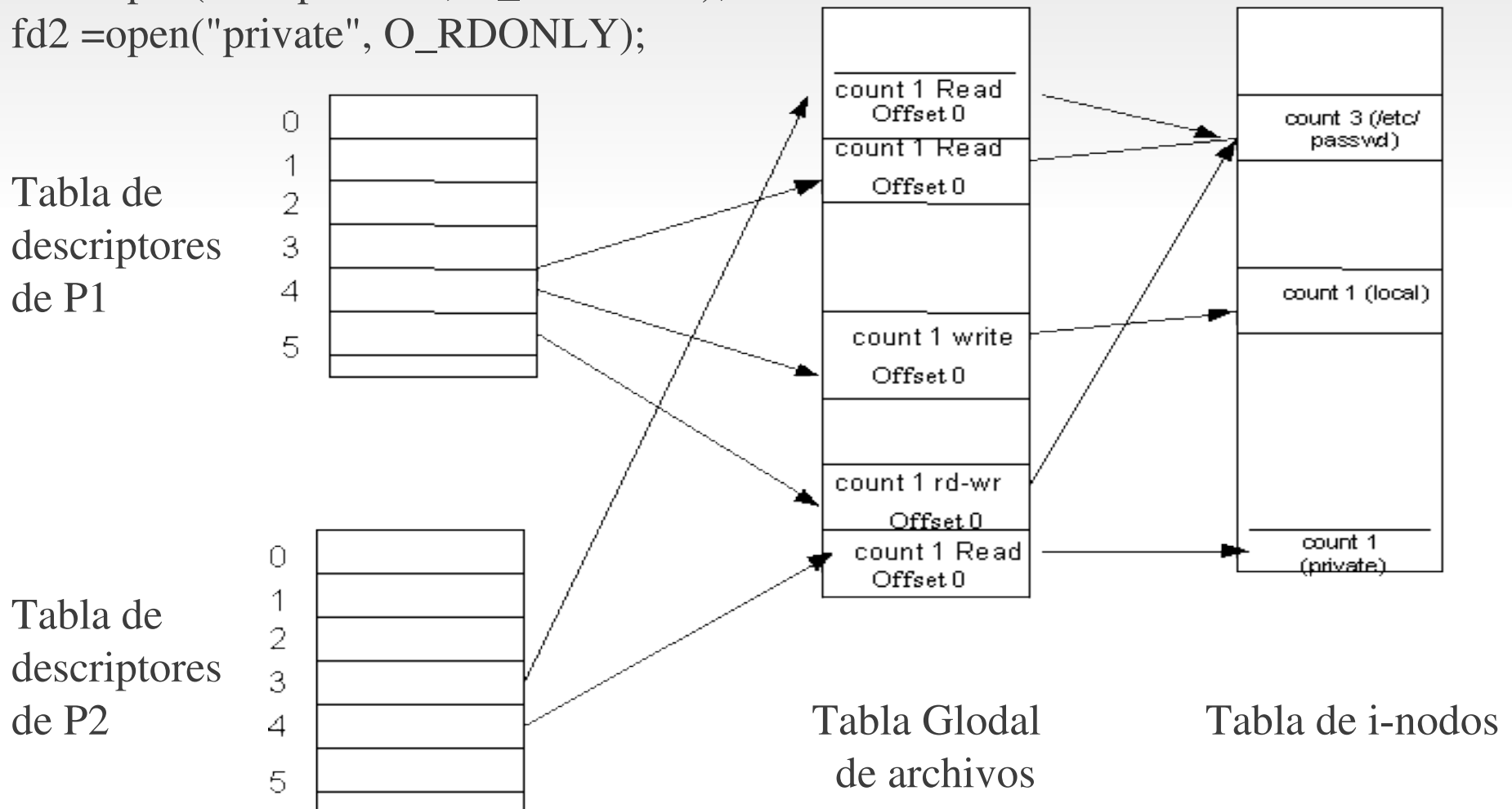


Llamadas al sistema para manipular archivos

Ejemplos (cont):

Suponga ahora que otro proceso P2 (sin ninguna relación de parentesco) realiza las siguientes operaciones:

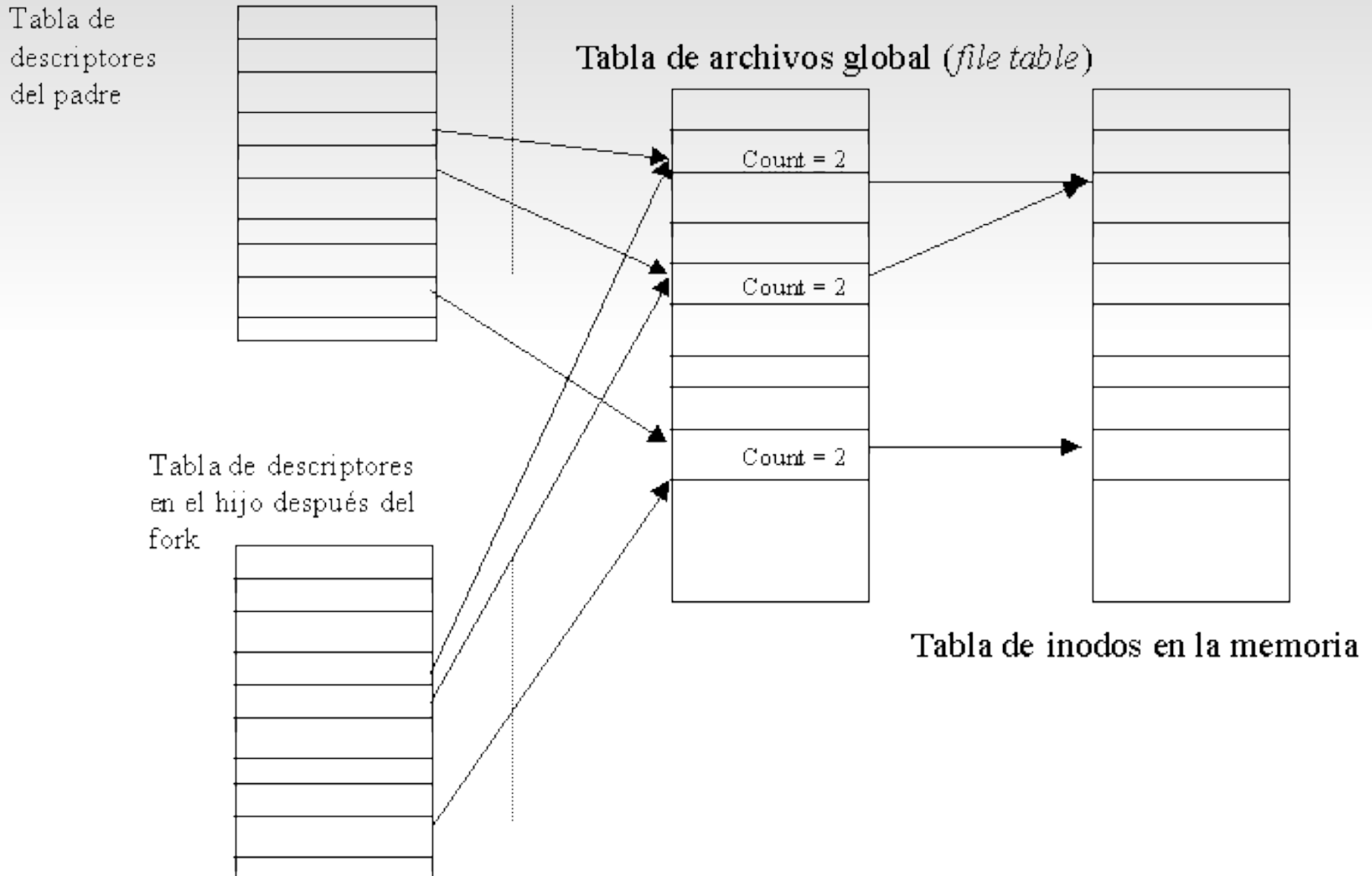
```
int fd1, fd2;  
fd1= open("/etc/passwd", O_RDONLY);  
fd2 =open("private", O_RDONLY);
```



Llamadas al sistema para manipular archivos

Ejemplos (cont):

Suponga un proceso P1 abre tres archivos abiertos y luego crea un proceso hijo:



Llamadas al sistema para manipular i-nodos

- Las llamadas al sistema para recuperar información almacenada en un i-nodo, son:

```
int stat(const char *path, struct stat *buf);
```

```
int fstat(int fildes, struct stat *buf);
```

```
int lstat(const char *path, struct stat *buf);
```

- La librería: `#include <sys/stat.h>`
- El parámetro **path** de **stat** y **lstat** representa el nombre del archivo (nombre simbólico)
- El parámetro **fildes** de **fstat** es el descriptor de archivos devuelto por `open` (nombre físico)
- fstat** se usa cuando el archivo está abierto
- lstat** devuelve información referida a los enlaces lógicos
- En el segundo parámetro de las tres llamadas, **struct stat *buf**, se devuelve el contenido del i-nodo. La información de **struct stat** es:

Llamadas al sistema para manipular i-nodos

mode_t st_mode /* modo del archivo pueden ser de tipo regular, directorio, especial (tipo bloque o caracter) o FIFO (Pipes) */

ino_t st_ino /* Número del inodo (sólo en la copia en memoria)*/

dev_t st_dev /* ID del dispositivo que contiene una entrada en el directorio para este archivo */

dev_t st_rdev /* ID del dispositivo, esta entrada se define sólo para archivos especiales tipo bloque o caracter */

nlink_t st_nlink /* Numero de enlaces */

uid_t st_uid /* ID de usuario del propietario del archivo */

gid_t st_gid /* ID de grupo del grupo al que pertenece el archivo */

off_t st_size /* Tamaño del archivo en bytes */

time_t st_atime /* Fecha del ultimo acceso al archivo */

time_t st_mtime /* Fecha de la ultima modificación */

time_t st_ctime /* Fecha de la ultima modificación al estado del archivo. Tiempo medido en segundos desde 00:00:00 UTC, Enero, 1, 1970. */

long st_blksize /* Tamaño del bloque de E/S*/

long st_blocks /* Numero de bloques de tamaño st_blksize asignados al archivo */

Llamadas al sistema para manipular i-nodos

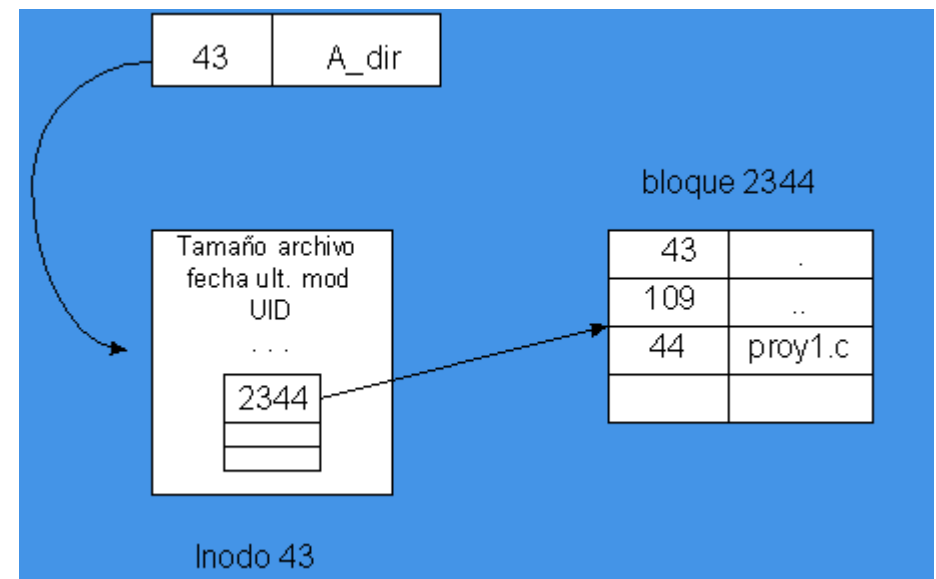
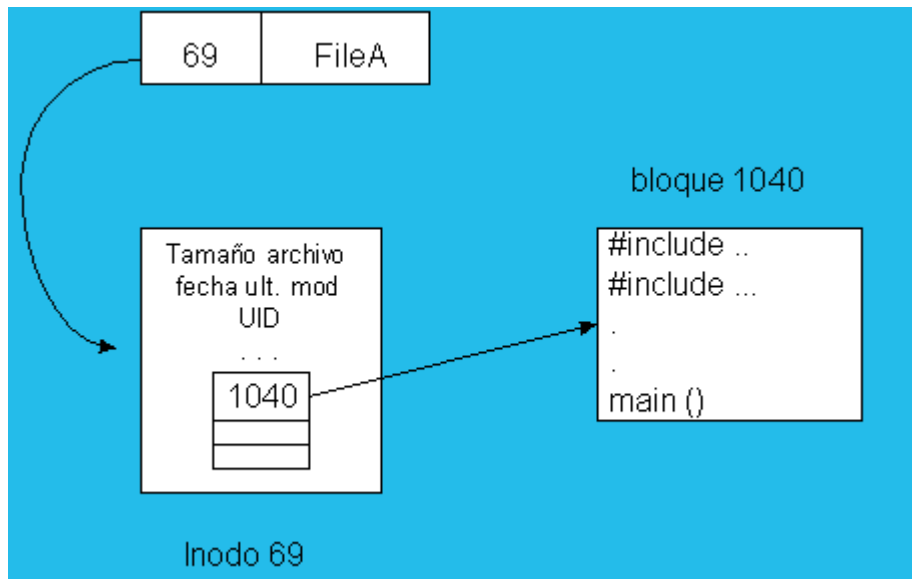
Ejemplo:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
void main(int argc, char *argv[]) {
    struct stat statbuf;
    if (argc != 2) {
        fprintf(stderr, "Usage: %s nombre_de_archivo\n", argv[0]);
        exit(1);
    }
    if (stat([argv[1]], &statbuf) == -1) {
        fprintf(stderr, " No se pudo aplicar stat sobre el archivo %s: %s \n", argv[1], strerror(errno));
        exit(1);
    }
    if (statbuf.st_mode & S_IFDIR)
        printf("%s es un directorio\n", argv[1]);
    else
        printf("%s no es un directorio\n", argv[1]);
    exit(0);
}
```

Llamadas al sistema para manipular directorios

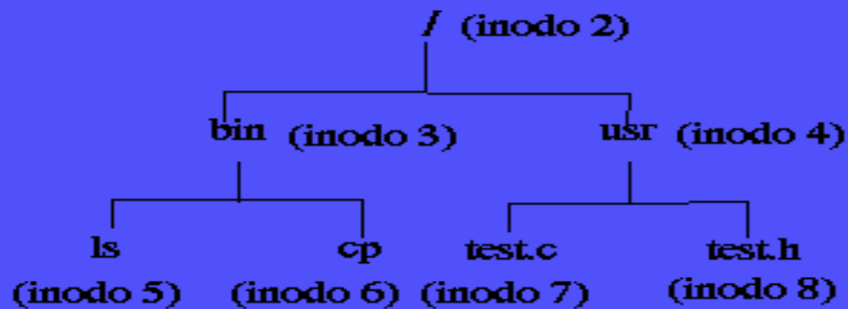
- Los directorios son archivos especiales que almacenan información del sistema de archivos:

Número de i-nodo	Nombre del Archivo
59	.
757	..
69	FileA
66	FileB
43	A_dir
...	...



Llamadas al sistema para manipular directorios

- Ejemplo de una estructura de directorio:



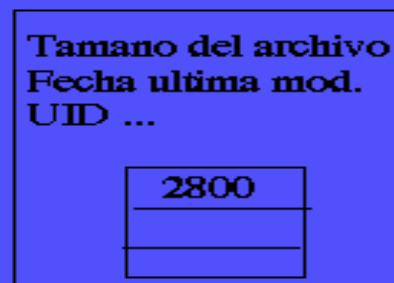
.	2
..	2
bin	3
usr	4

bloque 2000



.	3
..	2
ls	5
cp	6

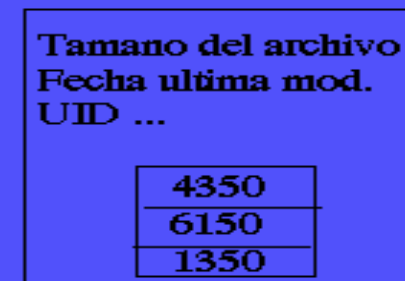
bloque 2300



inodo 4 (/usr)

.	4
..	2
test.c	7
test.h	8

bloque 2800



inodo 5 (archivo de datos ls)

Llamadas al sistema para manipular directorios

- La estructura completa de una entrada en los archivos directorios contiene:
 - Nro. de i-nodp
 - El tamaño del nombre (esto permite un manejo eficiente de nombres de longitud variable),
 - La longitud del registro
 - El nombre del archivo.

- Llamada al sistema para leer las entradas en un directorio:

```
#include <sys/dir.h>
```

```
int getdents(int fd, struct direct *buf, int structsize)
```

- La estructura dirent contiene:

```
off_t d_off /*offset de la próxima entrada de directorio */
```

```
ino_t d_fileno /* Nro. de i-nodo */
```

```
long d_reclen /* Longitud de la entrada */
```

```
long d_namlen /* Longitud del nombre del archivo */
```

```
name_t d_name /* Nombre del archivo */
```

Llamadas al sistema para manipular directorios

- Ejemplos del uso de `getdents`

```
1.- #include <sys/dirent.h>
```

```
#include <dirent.h>
```

```
void procesaDirectorio(char* dirName)    {  
    int fd, charsRead;  
    struct dirent dirEntry;  
    char fileName[MAX_FILENAME];  
    fd=open(dirName,ORDONLY);  
    if (fd==-1) fatal_error();  
    while (TRUE) {  
        charsread=getdents(fd,&dirEntry,sizeof(struct dirent));  
        if (charsRead ==-1 fatal_error());  
        if (charsRead ==0 break;  
        if (strcmp(dirEntry.d_name, ".") !=0 &&  
            (strcmp(dirEntry.d_name, "..") !=0) {  
            ...  
        }  
        lseek(fd,dirEntry.d_off,L_SET);  
    }  
    close(fd);  
}
```

Llamadas al sistema para manipular directorios

- Ejemplos del uso de `getdents` (cont.)

2. `while (1) {`

```
    charsRead = getdents ( fd ,&buf, (size_t) DIRENTSIZE);
```

```
    if (charsRead == -1) {
```

```
        perror("printdir:");
```

```
        return -1;
```

```
    }
```

```
    if (charsRead == 0) break; /* termino */
```

```
    printf ("%s\t%d\n",buf.d_name, buf.d_ino);
```

```
    lseek (fd, buf.d_off, SEEK_SET);
```

```
}
```

Llamadas al sistema para manipular directorios

- Otras llamadas para manipular directorios:

```
#include <dirent.h>
```

```
DIR *opendir(const char *dirname)
```

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

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

- Cuando un programa referencia un archivo a través de un *pathname* el sistema de archivos atraviesa la jerarquía (el árbol del sistema de archivos) hasta encontrar el nombre del archivo y el inodo en el directorio apropiado. Cuando ya se tiene el número del inodo, el sistema de operación puede determinar cualquier otra información que esté buscando respecto al archivo mismo.
- La función **opendir** retorna un descriptor asociado al directorio que sirve para ser utilizado por otras funciones de la misma librería.
- Cada llamada a **readdir** retorna un apuntador a una estructura que contiene información acerca de la próxima entrada al directorio. La función **readdir** retorna NULL cuando alcanza el fin del directorio.
- La función **closedir** cierra el directorio.

Llamadas al sistema para manipular directorios

- Ejemplo:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <errno.h>
void main(int argc, char *argv[]) {
    DIR *dirp;
    struct dirent *direntp;
    if (argc != 2) {
        fprintf(stderr, "Usage: %s nombre_directorio\n", argv[0]);
        exit(1);
    }
    if ((dirp = opendir(argv[1])) == NULL) {
        fprintf(stderr, "No se pudo abrir el directorio %s: %s", argv[1], strerror(errno));
        exit(1);
    }
    while ((direntp = readdir (dirp)) != NULL)
        printf("%s\n", direntp->d_name);
    closedir(dirp);
    exit(0);
}
```

Enlaces (links)

- Los enlaces son un mecanismo para compartir archivos entre directorios
- **Enlaces físicos (hard links), se crean con:**
 - **Línea de comando:**
 - `In /fuentes/fileA /backup/fileA.bck`
 - `Rrm /backup/fileA.bck`
 - **Llamadas al sistema:**
 - `int link(char *oldPath, char * newPath)`
 - `int unlink(char *fileName)`

