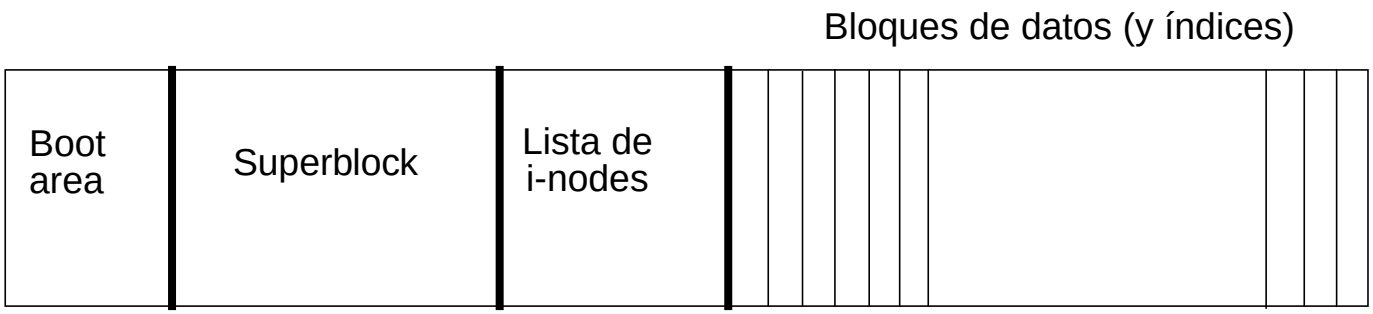
SISTEMA DE FICHEROS

Asignación de disco en UNIX

- Asignación de disco indexada (i-nodes)
- Bloques libres enlazados.



SISTEMA DE FICHEROS

Información que contiene el Superblock

- Fecha de creación.
- Tamaño en bloques del sistema de ficheros (!no área de swapping!)
- Tamaño en bloques de la lista de inodes.
- Número de bloques libres y de inodes.
- Puntero a primer bloque libre.
- Bit map de inodes.

SISTEMA DE FICHEROS

Información que contienen los Inodes

- Dispositivo en el que se encuentra.
- Número de links (nombres) del fichero.
- uid y gid del propietario.
- **Tamaño** del fichero.
- Tiempos de creación, último acceso, última modificación (versión).
- 10 punteros a bloques de datos (acceso directo).
- 1 puntero a un bloque de punteros (indirección).
- 1 puntero a un bloque de punteros a bloques de punteros (doble indirección).
- 1 puntero con triple indirección.

El tamaño del fichero afecta a su tiempo de acceso!!

SISTEMA DE FICHEROS DE UNIX

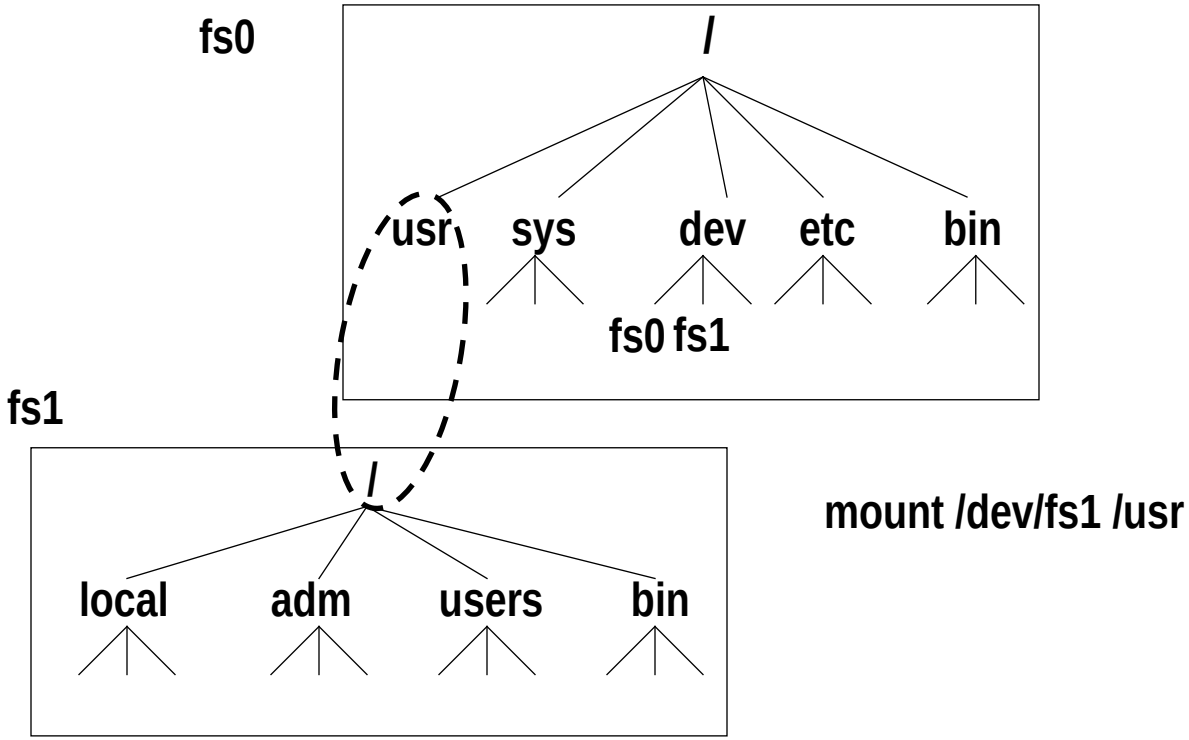
Características del SF de UNIX

- Sistema Jerárquico con una única raíz.
- Espacio de nombres único para ficheros y dispositivos.
- Un fichero (*i-node*) puede tener varios nombres (*paths* o *links*).
- Todos los accesos son secuenciales.
- Todas las operaciones se realizan con buffering (*buffer cache* de disco)

SISTEMA DE FICHEROS DE UNIX

Sistema jerárquico

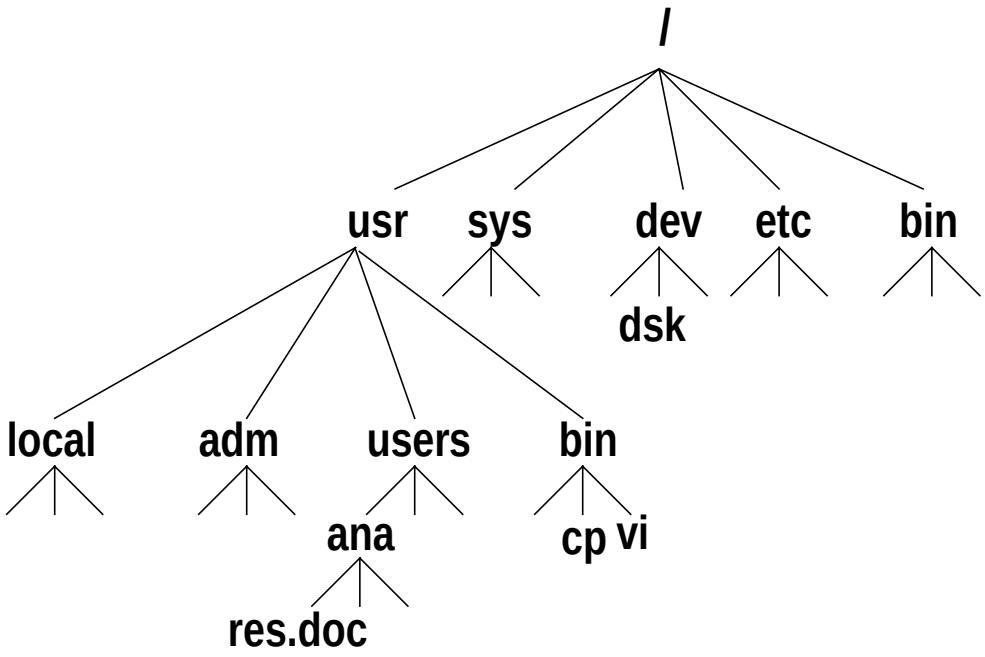
- Existe un único directorio raíz: “/”
- Se montan diversos (sub)sistemas de ficheros para poder acceder a sus ficheros.



SISTEMA DE FICHEROS DE UNIX

Espacio de nombres único

- Unix permite cuatro tipos de ficheros:
  - ordinarios (datos, ejecutables, de comandos,...)
  - directorios (qué inode corresponde a cada nombre)
  - dispositivos (información sobre modo de acceso, tipo,...)
  - especiales (FIFOS, *sockets*, *streams*, *soft-links*,...)
- Todos los ficheros se nombran igual: a través del *path*



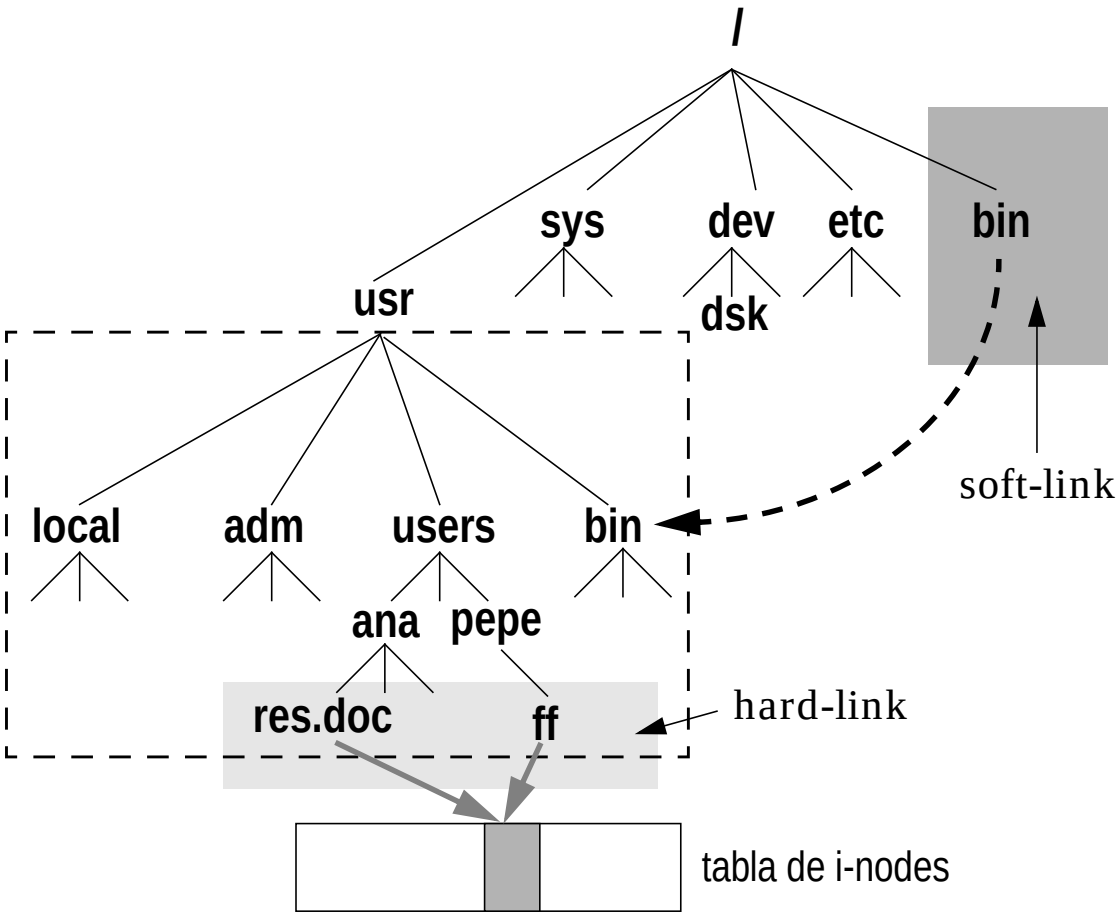
/dev/dsk
/usr/bin/vi
/usr/users/ana/res.doc
/etc

SISTEMA DE FICHEROS DE UNIX

Nombres de un fichero

- El nombre de un fichero puede darse de modo
  - **absoluto** (desde la raíz), o
  - **relativo** (desde el directorio actual, u otro intermedio)

- Un fichero puede tener diferentes nombres (*path* distintos)
  - ***hard link***: contabilizado en el i-node del fichero
  - ***soft link***: fichero especial con el *path* absoluto al fichero

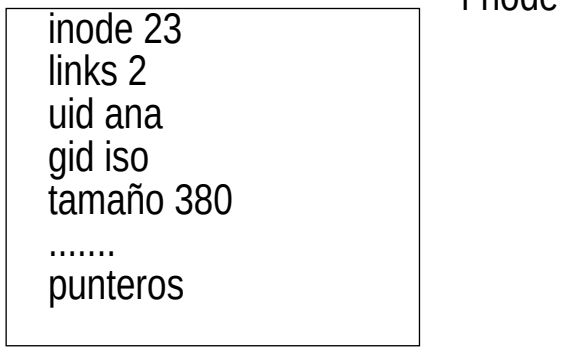


SISTEMA DE FICHEROS DE UNIX

El fichero UNIX: el *i-node*

- Su número es local al SF: desde 1 hasta el máximo formateado.
- Tiene toda la información sobre el fichero y los punteros de acceso a sus datos.
- Cada fichero puede tener más de un nombre (*path*).
- A cada fichero corresponde un único *i-node*.
  - en el *i-node* está el número de *links* (*paths*) que apuntan a ese fichero.

/usr/users/ana/res.doc
/usr/users/pepe/ff



- Borrar un fichero es borrar su path
  - decrementar el número de links del i-node
  - si no quedan más nombres al fichero, borrar de disco

SISTEMA DE FICHEROS DE UNIX

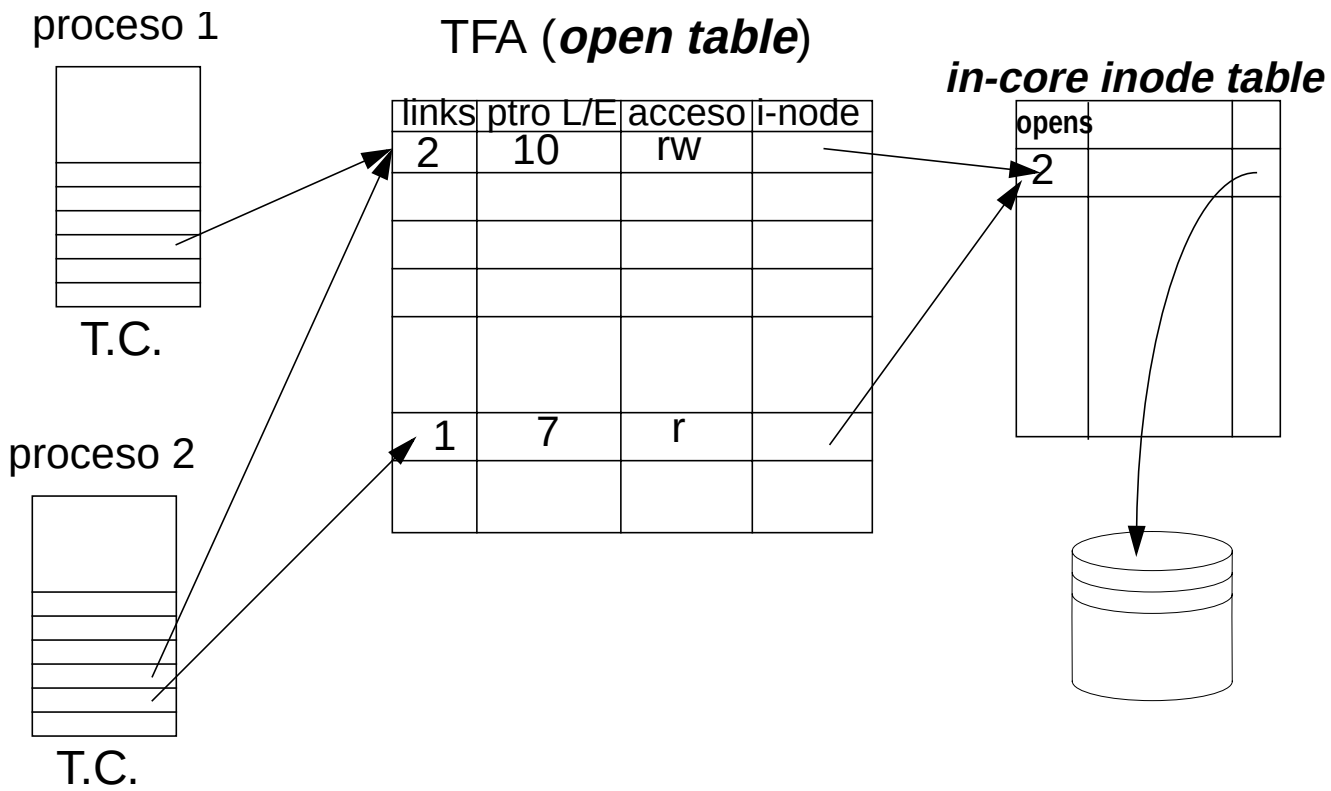
Hard link versus soft link

- **Hard link**: diversos nombres para un mismo fichero
  - el path ha de estar en el mismo SF (el número de inode es local)
  - toda modificación desde un nombre es visible automáticamente al resto
  - el inode es único
- **Soft link**: referencia a otro fichero
  - permite “atravesar” diferentes SF y referenciar a cualquier fichero
  - tiene un inode propio, diferente al del fichero referenciado
  - montar en un directorio diferente
  - borrar el fichero original....
  - es útil para ficheros estables, software de sistema

SISTEMA DE FICHEROS DE UNIX

Operaciones de E/S en UNIX

- Las llamadas al sistema trabajan con **canales virtuales**.
- Cada proceso tiene una **tabla de canales** propia, en su PCB (Process Control Block)
- Un canal se conoce también como **descriptor de fichero.**
- Cuando se abre un fichero (open):
  - se realiza el mapeo canal virtual- fichero
  - se trae a memoria la información del fichero (i-node)
  - se inicializa el puntero de L/E.
  - se actualiza la TC, la TFA, la in-core inode table.



- Al crear un proceso, se incrementa el número de links de la TFA

SISTEMA DE FICHEROS DE UNIX

Buffer cache

- Zona de memoria donde se mantienen lógicamente algunos bloques de disco
- Cuando se accede a disco, si el bloque está en memoria, no se va a disco
- Cada 30 s se hace una actualización al disco (también de la *in-core inode table*)
  - programa update
  - llamada al sistema sync()
- Mejora el rendimiento de la E/S
  - acceso a disco: orden de milisegundos
  - acceso a memoria: orden de microsegundos
- Políticas de mantenimiento de memoria cache (*write-behind*)
- Peligro de corrupción del SF cuando el sistema falla
- Tamaño habitual puede ser un 10% de la memoria física

SISTEMA DE FICHEROS DE UNIX

Llamadas al sistema

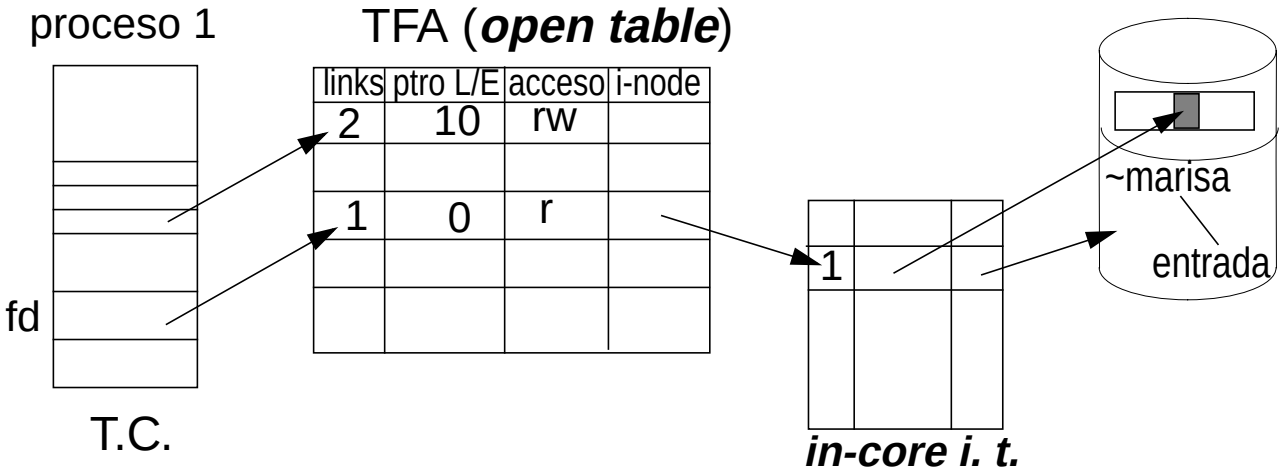
```
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>

int open(const char *path, int oflag[,mode_t mode])

int creat(const char *path,mode_t mode)
int open(path,O_WRONLY|O_CREAT|O_TRUNC, mode)
```

- Ejemplo:

```
fd = open("entrada", O_RDONLY)
```



- fd es el menor descriptor de fichero libre para ese proceso.
- El número de canales abiertos por proceso es limitado (config.).
- En DEC OSF/1, el mínimo es 64.

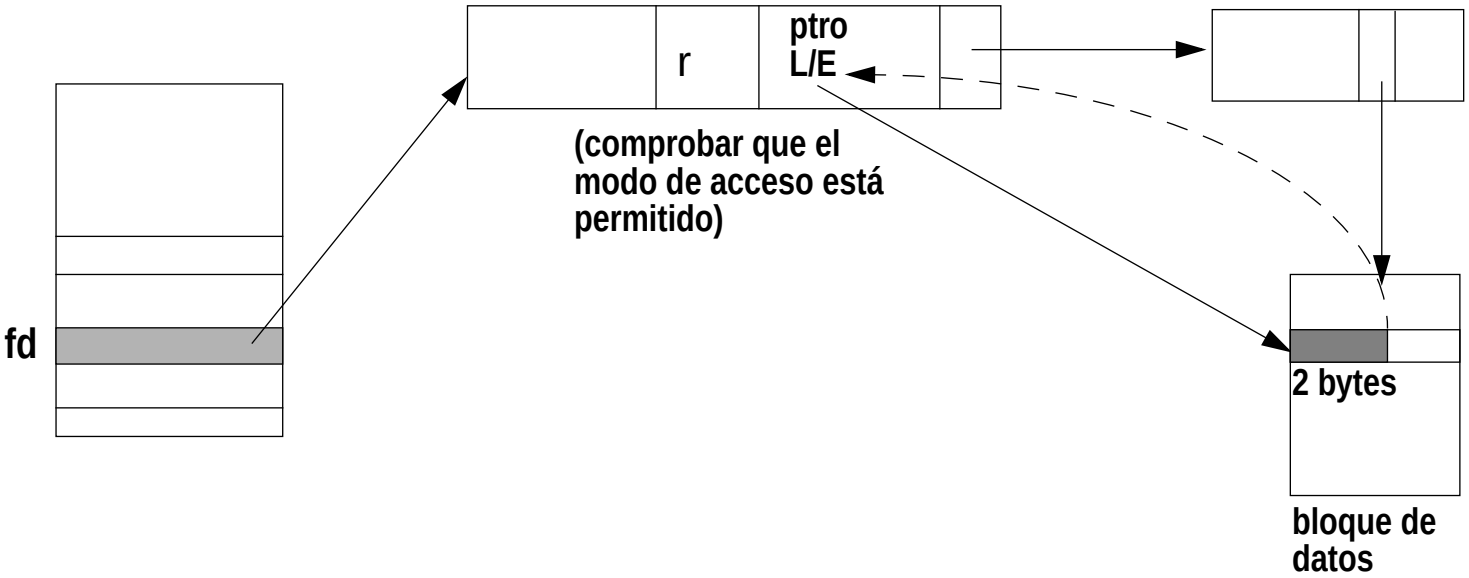
SISTEMA DE FICHEROS DE UNIX

```
#include <unistd.h>

ssize_t read(int fildes,void *buffer,size_t nbytes)
ssize_t write(int fildes,void *buffer,size_t nbytes)
```

- Ejemplo:

```
res = read(fd,&c,sizeof(char))
```



SISTEMA DE FICHEROS DE UNIX

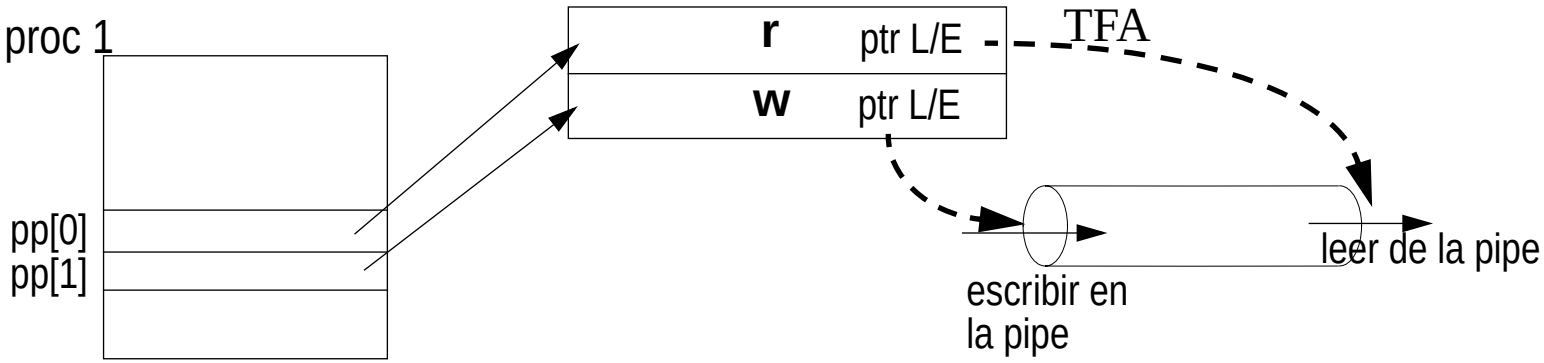
Otras llamadas al sistema

```
#include <unistd.h>

int pipe(int filedes[2])
```

- Ejemplo:

```
res= pipe(&pp);
```



- No es visible en el espacio de nombres
- Se hereda de padres a hijos (copia de la tabla de canales)

```
int mknod(const char *path, int mode, dev_t device)
```

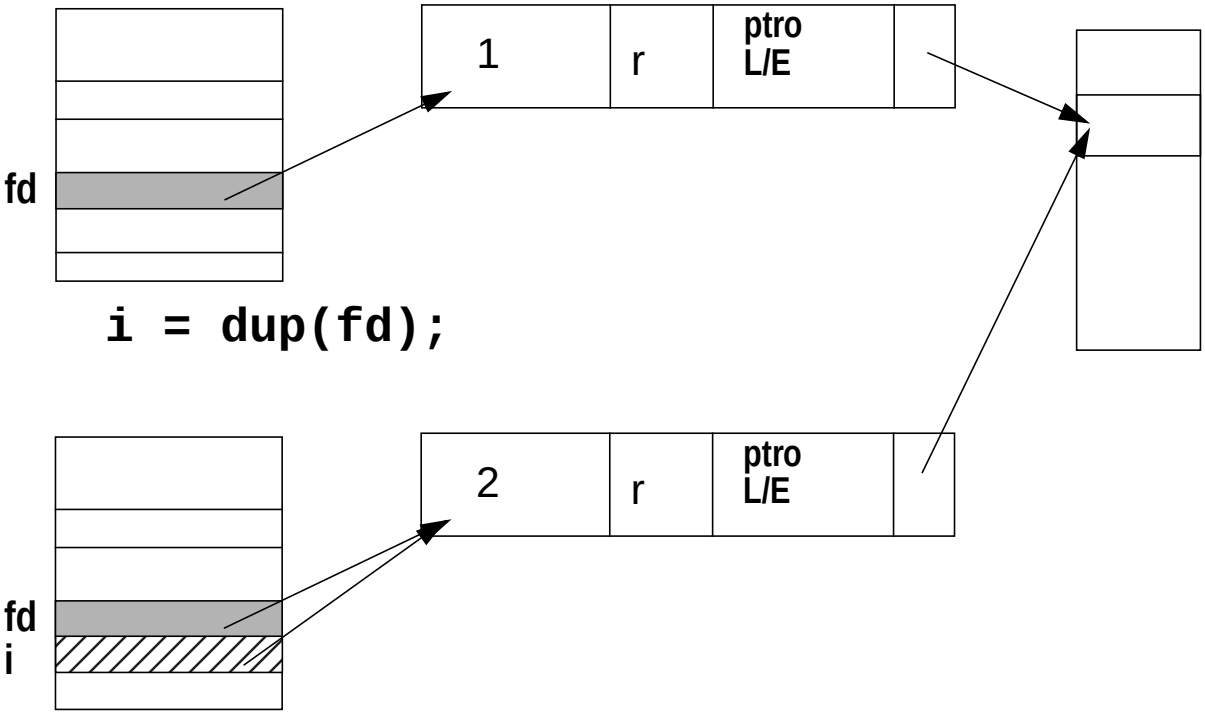
SISTEMA DE FICHEROS DE UNIX

Otras llamadas al sistema

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

int dup(int filedes )
```

- Ejemplo:



- Se asigna el primer descriptor de fichero libre.
- Útil en la redirección de los canales estándar.

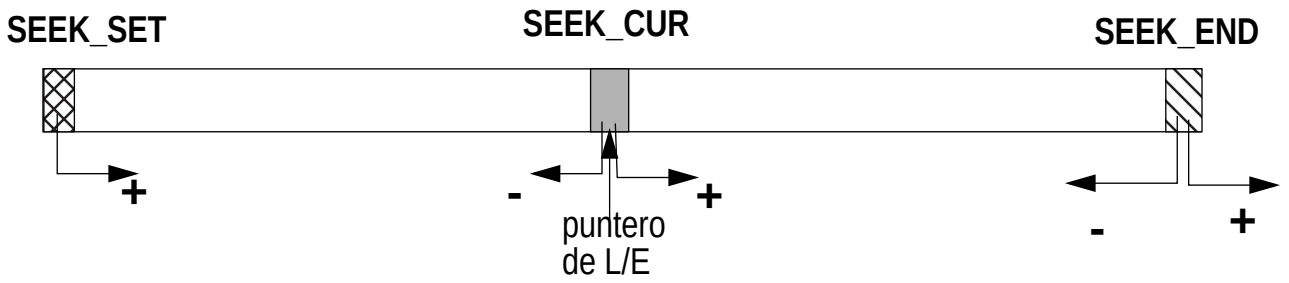
Otras llamadas al sistema

```
#include <sys/types.h>
#include <unistd.h>

off_t lseek(int fildes, off_t offset, int whence)

WHENCE = SEEK_SET
WHENCE = SEEK_CUR
WHENCE = SEEK_END
```

- Permite emular acceso directo a los ficheros.
- No extiende el tamaño del fichero (posteriores escrituras)
- Las constantes SEEK_SET... están definidas en `unistd.h`



SISTEMA DE FICHEROS DE UNIX

Otras llamadas al sistema

```
int ioctl(int fildes, int request, /* arg */ ...)
int fcntl(int fildes, int cmd, /* arg */ ...)

int link(const char *existing, const char *new)
int unlink(const char *path)

int chmod(const char *path, mode_t mode)

int mkdir(const char *path, mode_t mode)
int rmdir(const char *path)
```

PROTECCION Y SEGURIDAD

Fiabilidad del sistema de ficheros

- El Sistema de Ficheros está en uso constante.
- Los datos almacenados en los ficheros pueden ser más importantes que el propio sistema.
- Es importante proporcionar:

  - FIABILIDAD (la información está disponible y es precisa)
  - ROBUSTEZ (los errores se detectan y corrigen en lo posible)

Redundancia

Backups (totales e incrementales)

PROTECCION Y SEGURIDAD

Backups (copias de seguridad)

- **TOTAL**

  Copia periódica de los ficheros en cinta u otro medio de almacenamiento.

- **INCREMENTAL**

  Copia de los ficheros que han sido modificados desde el último *backup* total.

La frecuencia de *backups* depende del tipo de trabajo de los usuarios.

PROTECCION Y SEGURIDAD

Protección de ficheros

Definición:

Mecanismo para activar y permitir sólo determinados accesos a ficheros.

- OBJETIVO:
  - Evitar destrucciones o modificaciones accidentales.
  - Evitar acciones maliciosas.
  - Restringir el acceso a los ficheros.
- ELEMENTOS:
  - Identificación de usuarios.
  - Determinación de autorizaciones.
  - Control de acceso.