Sistemas Operativos

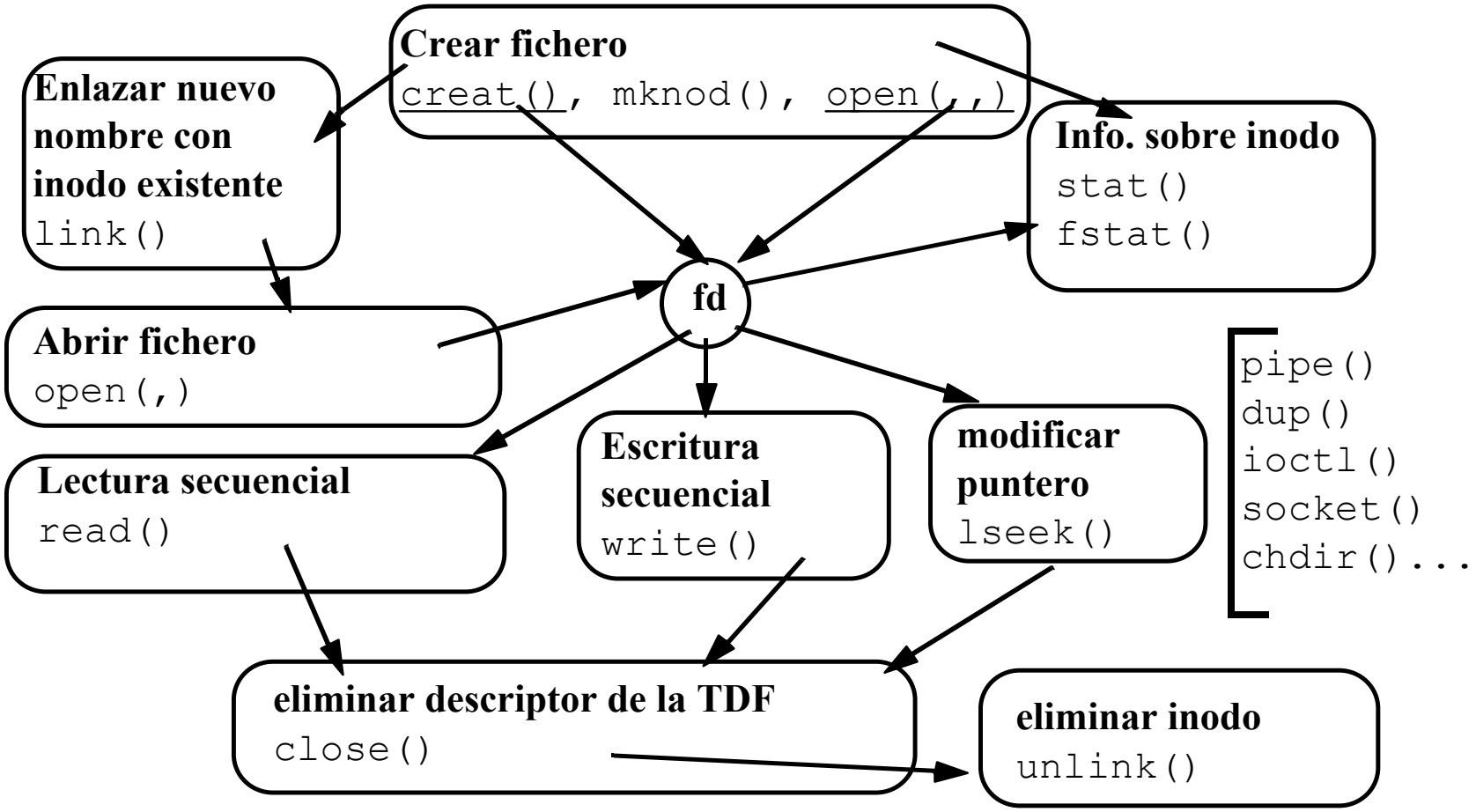
Ficheros: Llamadas al Sistema

Ficheros: Llamadas al Sistema

- Esquema de llamadas sobre ficheros
- Manejo de ficheros existentes: creat(), open(), close(), read(), write(), lseek(). Ejemplos
- Llamada mknod()
- Información sobre ficheros: stat(), fstat()
- Llamadas link() y unlink()
- E/S standard vs. llamadas al sistema
- forkfiles.c

[Ste05]: cap. 3, 4, 5

Esquema de llamadas sobre ficheros



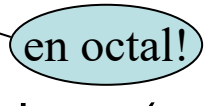
creat()

- **Sintaxis**: # include <fcntl.h>
  int **creat** (char * *path*, mode_t *mode*)
- **Acción**: crea un fichero nuevo o reescribe uno existente
  El fichero queda abierto sólo para escritura

  path nombre del fichero a crear (absoluto o relativo)

  Si *path* no existe: asigna/inicializa i-nodo
  crea entrada en TFA (cursor al principio)
  nueva entrada directorio <*path*,i-nodo>
  Si *path existe:* trunca el fichero

  mode permisos de acceso del fichero (según umask)

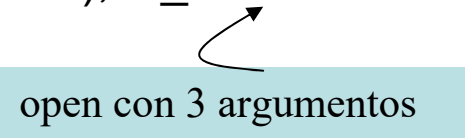


- **Devuelve**: primer fd libre en TDF (apunta a TFA) ó –1 (errno)

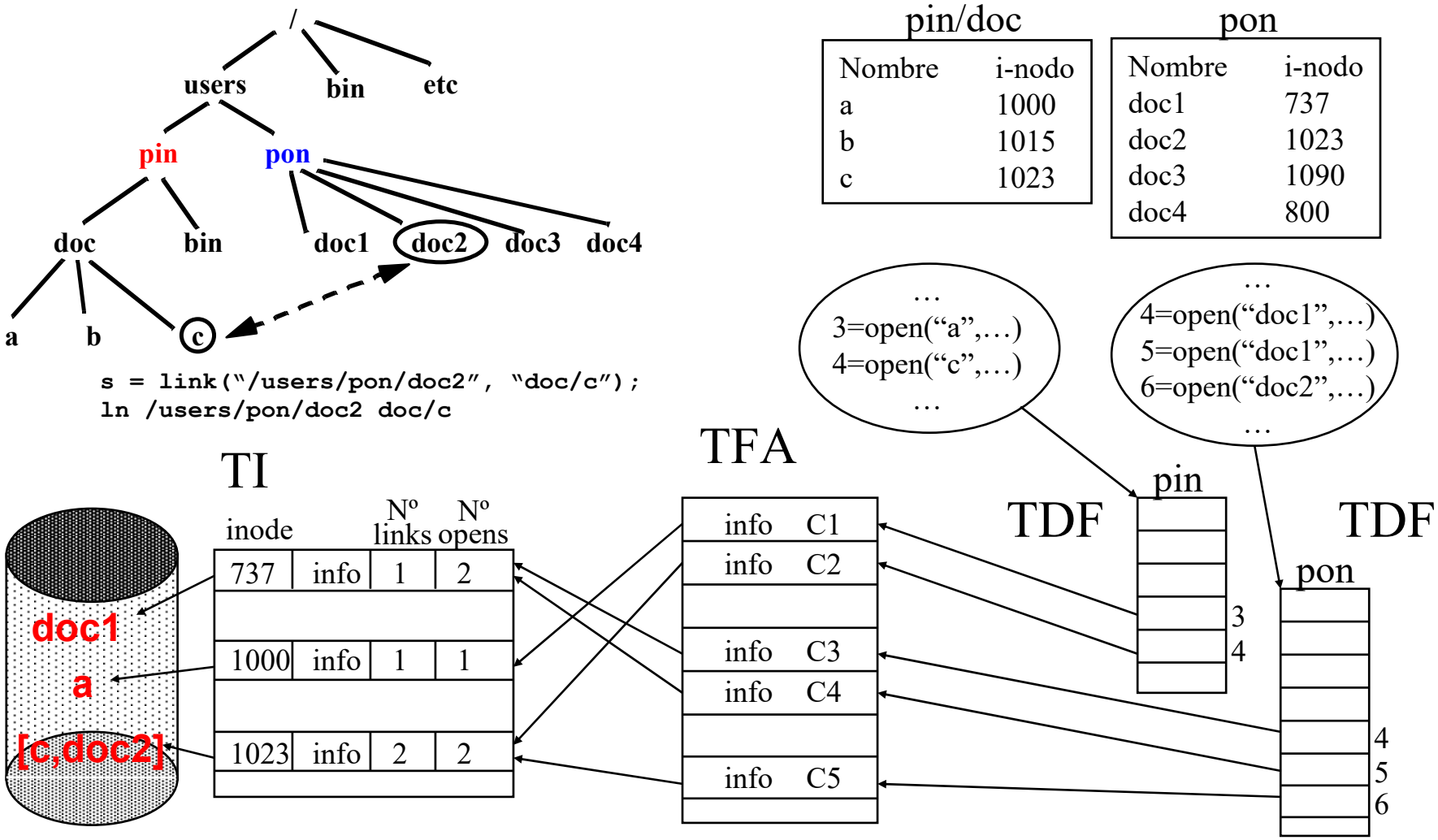
open()

- **Sintaxis**: # include <fcntl.h>
  int **open** (char * *path*, int *oflag*)
- **Acción**: abre un fichero existente
  path nombre del fichero a abrir (absoluto o relativo)
  oflag modo de apertura del fichero:
  O_RDONLY, O_WRONLY, O_RDWR (0,1,2)
  sino modo de apertura indefinido
  + OR con O_APPEND (cursor final antes de escribir), O_CREAT...
  se identifica el i-nodo
  se crea entrada en TFA
  (cursor al principio, *oflag*)
- **Devuelve**: primer fd libre en TDF (apunta a TFA) ó –1 (errno)

creat(pathname,mode) =
open(pathname,O_WRONLY | O_CREAT | O_TRUNC,mode)
Si queremos crear fichero pero poderlo leer también
open(pathname,O_RDWR | O_CREAT | O_TRUNC,mode)



Tablas en memoria del Sistema de Ficheros (3)



close()

- **Sintaxis**: # include <unistd.h>
  int **close** (int *fildes*)
- **Acción**: cierra el descriptor de fichero *fildes*

  queda libre la entrada *fildes* de TDF
  decrementa nfildes en TFA
  si (nfildes == 0)
  queda libre entrada en TFA
  decrementa nopens en TI
  si (nopens == 0)
  si (nlinks == 0) borra fichero;
  elimina inodo de TI;
- **Devuelve**: 0 si bien ó –1 (errno)
- Cuando termina un proceso el kernel cierra todos los ficheros de su TDF y actualiza tablas TFA y TI

/* ej42.c */

```
#include <errno.h>
#include <stdio.h>
#include <fcntl.h>
#define DIM 300

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

    if( (fd[0]  = creat( argv[1], 0777 )) == -1 ){
        perror( "creat" ); exit(1);}
    printf( "Abro el %d\n", fd[0] );
    for( i = 1; ( fd[i] = open( argv[1], O_RDONLY )) != -1 ; i++ )
        printf( "Abro el %d\n", fd[i] );
    contador=i;
    if( errno == EMFILE )
        printf( "\tTotal fich. abiertos: %d\n", (contador+3) );
    else{  perror( "open" ); exit(1); }
    for( i=0; i<contador; ++i)
        close( fd[i] );
    unlink( argv[1] );
}
```

Crea tantos descriptores de fichero como permite el sistema (OPEN_MAX=256)

EMFILE: agotada TDF
(ENFILE: agotada TFA)

lseek()

- **Sintaxis**: # include <unistd.h>
  off_t **lseek** (int *fildes*, off_t *offset*, int *whence*)
- **Acción**: mueve el cursor de lectura/escritura de un fichero
  se modifica la posición del cursor *offset* bytes
  hacia delante o hacia detrás (según signo de *offset*)
  desde un punto de referencia (*whence*)

whence	posición desde la que se aplica el offset	
	SEEK_SET / L_SET / 0	principio fichero (offset>=0)
	SEEK_CUR / L_CUR / 1	posición actual
	SEEK_END / L_END / 2	posición final

- **Devuelve**: nueva posición del cursor respecto del principio
  –1 si error (errno)
- **Ejemplo:** Para saber la posición actual del cursor:

```
off_t cursor;
cursor = lseek(fd, 0, SEEK_CUR);
```

read()

- **Sintaxis**: # include <unistd.h>

  ssize_t **read** (int *fildes*, void **buf*, size_t *nbyte*)

- **Acción**: lee *nbyte* bytes del fichero asociado al descriptor *fildes* y los almacena en *buf*

  la lectura comienza en la posición indicada por el cursor (en TFA)

  se modifica la posición del cursor con el número de bytes realmente leídos

- **Devuelve**: número de bytes realmente leídos

  0 si ya no hay más bytes que leer

  –1 si error (errno)

write()

- **Sintaxis**: # include <unistd.h>

  ssize_t **write** (int *fildes*, const void **buf*, size_t *nbyte*)

- **Acción**: escribe *nbyte* bytes del buffer *buf* en el fichero asociado al descriptor *fildes*

  la escritura comienza en la posición indicada por el cursor (en TFA)

  se modifica la posición del cursor con el número de bytes realmente escritos

  se actualiza el tamaño del fichero en i-nodo

- **Devuelve**: número de bytes realmente escritos

  –1 si error (errno)

/* ej10.c */

Copia el contenido de fich1 en fich2

Probar:
- ej10 fich_existe fich_nuevo
- ej10 ej10.c nuevo.c
- ej10 ej10 ej10_nuevo
- ej10 . direct
- ej10 fich_existe /dev/tty

```
#include <fcntl.h>
#include <stdio.h>

main( argc, argv )
int argc; char *argv[];
{    int fdfnt, fddst;
     void copia();

     if (argc != 3){
          printf( "Uso: %s fich1 fich2 ", argv[0] ); exit( 1 ); }

     if ( (fdfnt = open( argv[1], O_RDONLY )) == -1 )
          { fprintf( stderr, "\tError open\n" ); exit( 1 ); }

     if ( (fddst = creat( argv[2], 0666 )) == -1 )
          { fprintf( stderr, "\tError creat\n" ); exit( 1 ); }

     copia( fdfnt, fddst );
     exit( 0 );
}
```

```
void copia ( fnt, dst )
int fnt, dst;
{    int cuenta;
     char buf[BUFSIZ];

     while ( (cuenta = read( fnt, buf, sizeof( buf ) )) > 0 )
          write( dst, buf, cuenta );
}
```

en <stdio.h>

/* reverse.c */

```
#include <stdio.h>
#include <fcntl.h>
#include "error.h"

main(argc,argv)
int argc;    char *argv[];
{    char c;
     int i, fdfnt;
     long where;

     if(argc != 2){ printf( "Uso: %s fichero_a_invertir" argv[0]); exit(1); }

     if((fdfnt = open( argv[1], O_RDONLY )) == -1) syserr("open");
     if((where = lseek( fdfnt, -1L ,SEEK_END )) == -1 )      syserr("lseek");

     while(where >= 0){
          read(fdfnt, &c, 1);
          write(1, &c, 1);
          where = lseek ( fdfnt, -2L ,SEEK_CUR );
     };
}
```

Invierte el contenido de un fichero

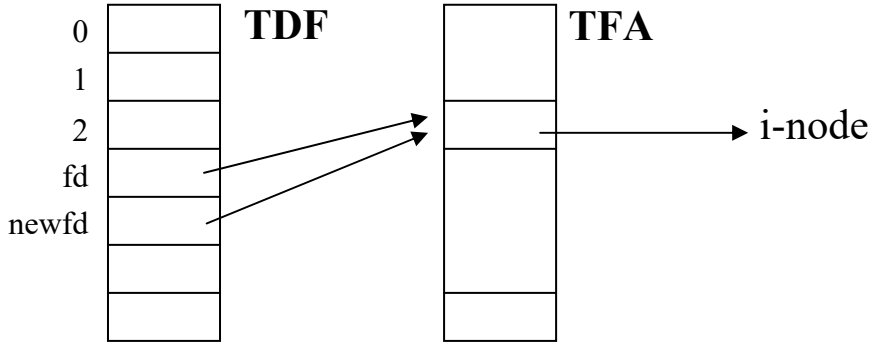
formato long

dup( )

- **Sintaxis**: #include <unistd.h>
  int **dup**(int *fd*);

- **Acción**: duplica *fd* con un nuevo descriptor (*newfd*)
  fd es un descriptor ya asignado con open, creat, pipe, dup... => apunta a TFA
  tras dup, *fd* y *newfd* apuntan a la misma entrada en TFA
- **Devuelve**: el siguiente descriptor libre en TDF ó –1 si error

newfd = dup(fd);



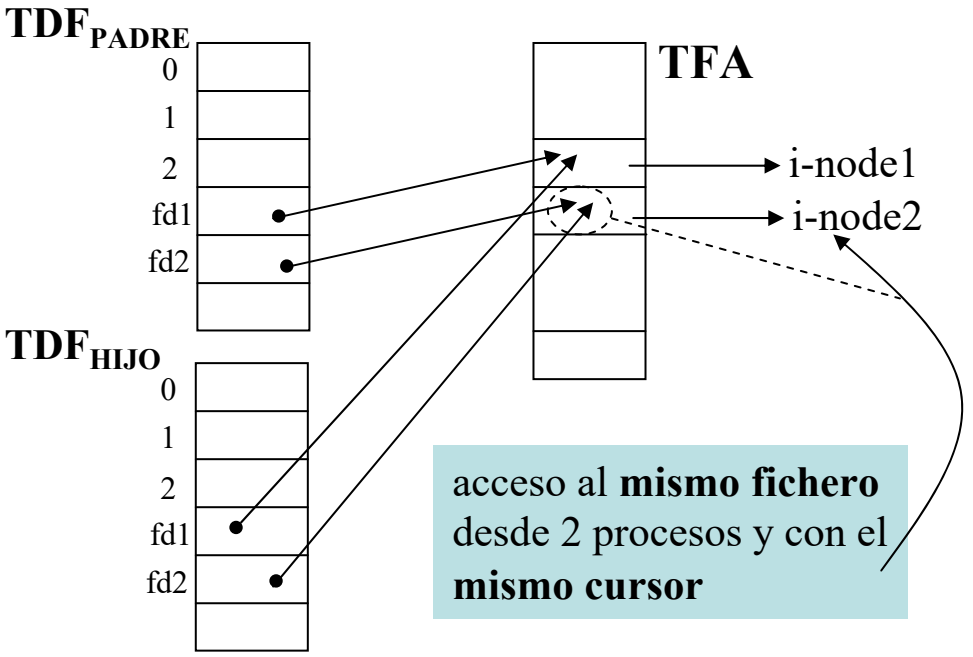
Tablas *vs* fork() y exec()

- Tras hacer un **fork( )** el proceso HIJO recibe una copia de la Tabla de Descriptores de Fichero del proceso PADRE:

  => - la TDF se *duplica* entera

  - TODAS las entradas de la TDF_{HIJO} apuntan a la misma entrada en TFA que la correspondiente entrada de la TDF_{PADRE}

- Tras hacer un **exec( )** el proceso *ejecutado* mantiene la TDF del proceso *ejecutor*



/* forkfiles.c */

```
#include<fcntl.h>
#include<stdio.h>
#include "error.h"

int sfd, tfd;
char c;

main(argc, argv)
int argc; char **argv;
{
    if( argc != 3 ) exit( 1 );
    if( (sfd = open( argv[1], O_RDONLY ) ) == -1 ) syserr( "open" );
    if( (tfd = creat( argv[2], 0777 ) ) == -1 )  syserr( "creat" );

    if( fork() == -1) syserr( "fork" );
    copy();
    exit( 0 );
}

copy()
{
    for(;;){
        if( read( sfd, &c, 1) != 1)   return;
        write( tfd, &c, 1 );
    }
}
```

Padre e Hijo acceden a los mismos ficheros a través del cursor

stat(), fstat()

- **Sintaxis**: # include <sys/stat.h>
  int **stat** (char **path*, struct stat **buf*)
  int **fstat** (int *fildes*, struct stat **buf*)
- **Acción**: recogen información del fichero asociado al nombre *path* (**stat**) o al descriptor *fildes* (**fstat**)
  se lee el i-nodo para rellenar la estructura *buf*

fichero abierto

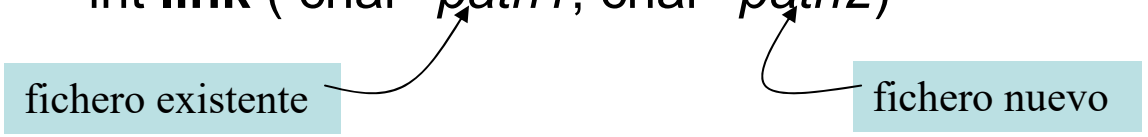
```
struct stat {
    dev_t  st_dev;  /* ID of dev containing a directory entry for this file */
    ino_t  st_ino;            /* Inode number */
    mode_t st_mode;           /* File type & Permission bits */
    nlink_t  st_nlink;        /* Number of links */
    uid_t  st_uid;            /* User ID of file owner */
    gid_t  st_gid;            /* Group ID of file group */
    dev_t  st_rdev; /* mayor/minor IDs;defined only for char or blk spec files */
    off_t  st_size;           /* File size (bytes) */
    time_t st_atime;          /* Time of last access */
    time_t st_mtime;          /* Last modification time */
    time_t st_ctime;          /* Last file status change time */
    blksize_t st_blksize;     /* best I/O block size (8192) */
    ...                       /* hay mas campos */
};
```

- **Devuelve**: 0 si bien ó –1 (errno)

link()

- **Sintaxis**: # include <unistd.h>

  int **link** (char **path1*, char **path2*)



- **Acción**: crea para *path2* una nueva entrada en el directorio con el mismo i-nodo de *path1*:

  <*path1*, i-nodo i > <*path2*, i-nodo i >

  (número de links del i-nodo i)++;

- **Devuelve**: 0 si bien ó –1 (errno)

unlink()

- **Sintaxis**: # include <unistd.h>
  int **unlink** (char **path*)

- **Acción**: elimina la entrada en el directorio del fichero *path*:

  <*path*, i-nodo>

  nlinks = nlinks-1 (en i-nodo);
  si (nlinks == 0) {
    si (nopens (en TI) == 0)
      borra fichero;
    sino el borrado se hará en el último **close**;
  }

- **Devuelve**: 0 si bien ó –1 (errno)

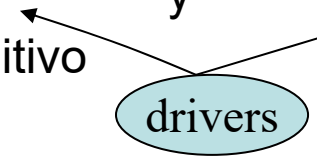
mknod()

- **Sintaxis**: # include <sys/stat.h>
  int **mknod** (char * *path*, mode_t *mode*, dev_t *dev*)
- **Acción**: crea tipos de ficheros especiales
  (directorios, dispositivos, FIFOS...)
  uso restringido al super-usuario excepto en FIFOS

path nombre del fichero a crear

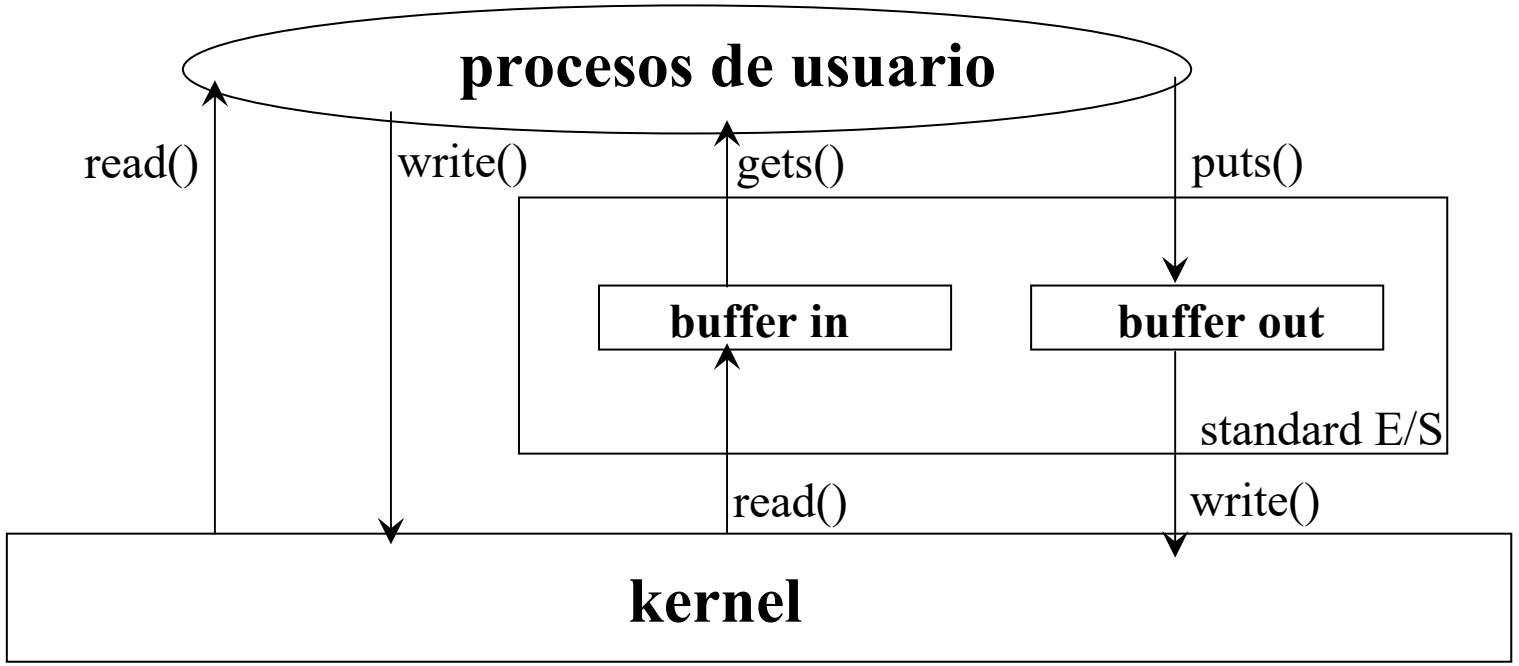
mode tipo y permisos de acceso del fichero (+ umask)
constantes simbólicas definidas en *stat.h*:
S_IFDIR, S_IFBLK, S_IFCHR, S_IFIFO ...
S_IRWXU, S_IRWXG, S_IRWXO ...

dev solo para ficheros dispositivos (caracteres, bloques)
número mayor y número menor (ls -l /dev)
clase de dispositivo número dentro de una clase



- **Devuelve**: 0 si bien ó –1 si error (errno).

E/S estándar vs. llamadas al sistema



- Las funciones de biblioteca agrupan la E/S (menos SC's)
  - Fully buffered: E/S real se realiza cuando el buffer se llena o se ejecuta fflush
  - Line buffered: E/S real cuando llega caracter fin de linea ('\n' o ASCII 10)
    Pensado para dispositivos de líneas (terminales, ...)
  - Unbuffered: E/S real sin uso de buffers

E/S estándar vs. llamadas al sistema

- Por defecto
  - stdin y stdout son:
    - Line buffered si están dirigidas a una terminal o
    - Fully buffered en caso contrario
  - stderr es unbuffered
- Se puede modificar el comportamiento de las funciones de biblioteca con setbuf() o setvbuf()
  - Cambiar tamaño de los buffers (por defecto BUFSIZ=1024)
  - fully buffered (_IOFBF)
  - line buffered (_IOLBF)
  - unbuffered (_IONBF)

E/S estándar vs. llamadas al sistema

- **Llamadas al sistema**

```
int fd;
/*file descriptor*/

fd = open(nombre,...);
read(fd, ...);
write(fd, ...);
lseek(fd, ...);
...
close(fd);
```

- **Funciones de biblioteca C**

```
FILE *fp;
/*file pointer*/

fp = fopen(nombre,...);
fgets(fp, ...);
fprintf(fp, ...);
fseek(fp, ...);
...
fclose(fp);
```

E/S estándar vs. llamadas al sistema

- **Llamadas al sistema**

```
Entrada/salida estandar
0=STDIN_FILENO
1=STDOUT_FILENO
2=STDERR_FILENO
son file descriptor

read(0, ...);
write(1, ...);
write(2, ...);
...
close(1);
```

- **Funciones de biblioteca C**

```
Entrada/salida estandar
stdin
stdout
stderr
son file pointer

fgets(stdin, ...);
fprintf(stdout, ...);
fprintf(stderr, ...);
...
fclose(stdout);
```

E/S estándar vs. llamadas al sistema

- Qué ocurre con los bufferes cuando:

– **fork**: se copia el $Buffer_{PADRE}$ al $Buffer_{HIJO}$

(si había cosas pendientes de escribir, se escriben 2 veces)

– **exec**: desaparece el Buffer

(si había cosas pendientes de escribir, se pierden)

– **exit**: se vacía el Buffer (equivalente a fflush)

– **terminación involuntaria**:

desaparece el Buffer

(si había cosas pendientes de escribir, se pierden)

Ejemplo: ficheros, E/S estándar, redirecciones

```
#include<stdio.h>
main(){
        int id,estado,fd;

        printf("linea de texto n 1\n");
        if((id=fork())==0) {
                close(1);
                creat("salida.dat",0777);
                write(1,"linea de texto n 2\n",19);
                exit(1);
        }
        else {
                while(wait(&estado)!=id);
                write(1,"linea de texto n 3\n",19);
                exit(0);
        }
}
```