

# Fonaments dels Sistemes Operatius (FSO)

Departament d'Informàtica de Sistemas i Computadores (DISCA)  
*Universitat Politècnica de València*

Bloc Temàtic 3: Sistema d'Arxius i E/S  
Seminari Unitat Temàtica 7

SUT07: Crides al sistema Unix per  
a arxius

f SO

DISCA



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA

- **Objectius**

- Descriure el concepte de **descriptor d'arxius**
- Comprendre el **concepte de taula de descriptors d'arxius i la seu utilitat**
- Estudiar les **crides al sistema** de Unix per a treballar amb **arxius**
- Manejar els mecanisme de **redirecccionament d'entrada i sortida** de processos Unix
- **Comunicar** processos Unix amb **tubs (“pipe”)**

- **Contingut**
  - Arxius en Unix
  - Crides al sistema per a arxius
  - Redireccions i tubs
- **Bibliografia**
  - “Fundamentos de Sistemas Operativos”, A. Silberschatz. P.B. Galvin, 7<sup>a</sup>Ed. Mc Graw Hill, ISBN: 968-444-310-2
  - “Operating System Concepts”, A. Silberschatz. P.B. Galvin, 8<sup>a</sup>Ed., Wiley, ISBN: 978-0-470-12872-5
  - “UNIX Programación Práctica”, Kay A. Robbins, Steven Robbins. Prentice Hall. ISBN 968-880-959-4
  - “[UNIX Systems Programming](#)”, Kay A. Robbins, Steven Robbins. Prentice Hall. ISBN 0-13-042411-0

- **Contingut**
  - Arxius en Unix
  - Crides al sistema per a arxius
  - Redireccions i tubs (“pipe”)
  - Crides per a redireccions i tubs
  - Exemples en C

- Tipus d'arxius en Unix
  - **Regular:** arxius convencionals de dades (programa, text, ....) emmagatzemats en memòria secundària.
  - **Directori:** que associa noms als arxius.
  - **Tubs:** arxius sense nom, d'accés seqüencial, per a comunicació entre processos.
  - **FIFO:** arxius amb nom, d'accés seqüencial, per a comunicació entre processos.
  - **Especial:** representa un dispositiu físic o fictici del sistema
    - Dispositius de caracters (marcats amb 'c'), com les consoles `/dev/ttynn` o `/dev/pts/n` o l'engolidor fictici `/dev/null`

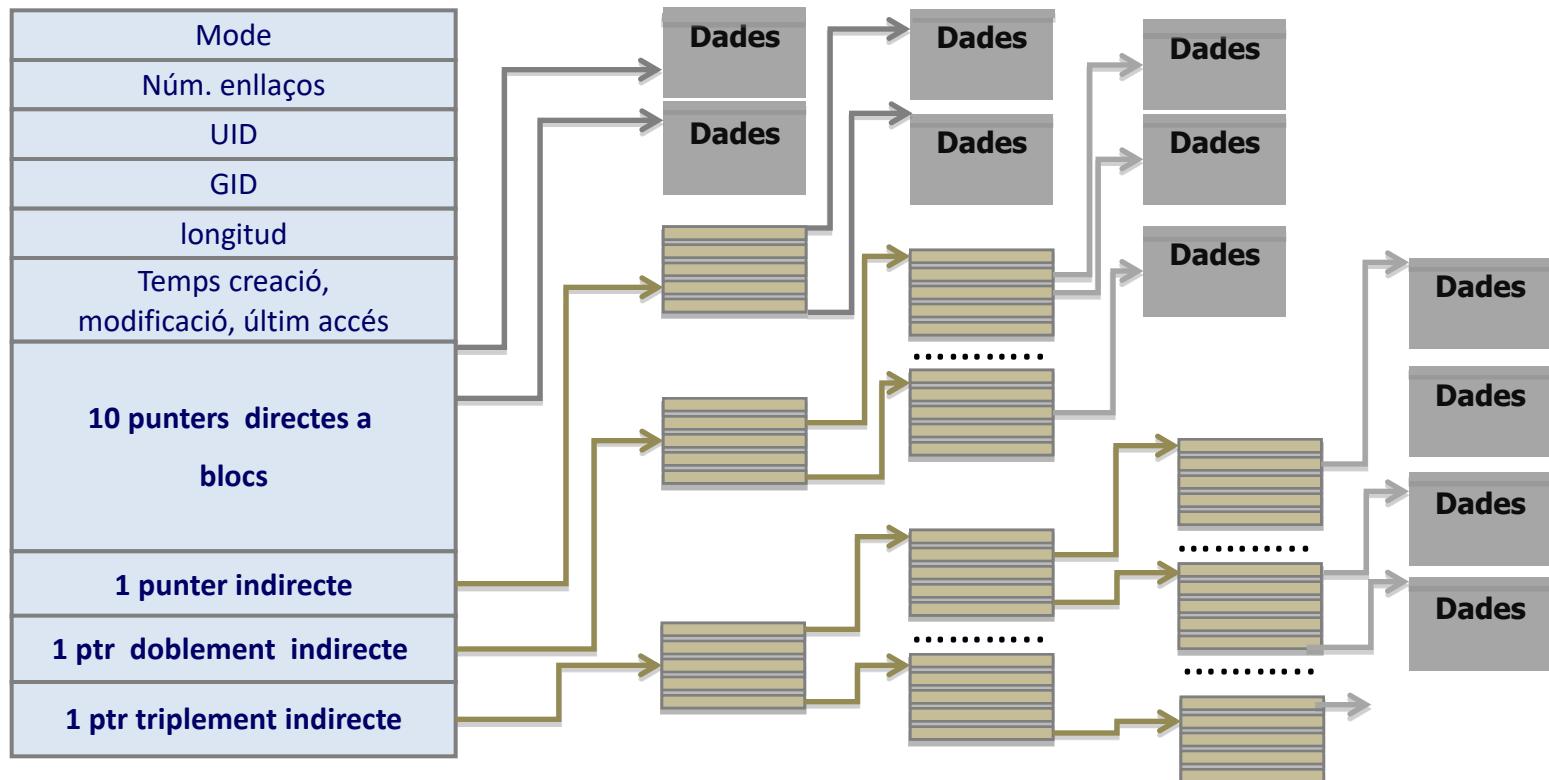
**Nota:** Des del shell de UNIX, el tipus d'arxiu es pot veure amb l'ordre “`ls -la`” que el mostra codificat en la primera lletra del llistat

- arxiu regular: marcat com ‘-’
- arxiu directori: marcat amb ‘d’
- arxiu especials: marcat amb ‘c’ (caràcter) o ‘b’ (blocs)
- arxiu FIFO : marcat com ‘p’

- **Atributs dels arxius Unix**
  - Tipus d'arxiu
  - Propietari (owner UID)
  - Grup propietari (owner GID)
  - Permisos d'accés (permission bits)
  - Nombre d'enllaços
  - Instants d'últim accés i última modificació
  - Grandària (en bytes)

- Node-i (*i-node*)**

- Estructura de dades del s.o per a guardar tots els atributs de l'arxiu excepte el nom
  - A cada arxiu en Unix se li assigna un node-i
  - Utilitza assignació indexada de blocs amb punters a blocs directes, indirectes simples, doblement indirectes i triplement indirectes.



- **Estructura d'arxiu en Unix**

- Vector de bytes

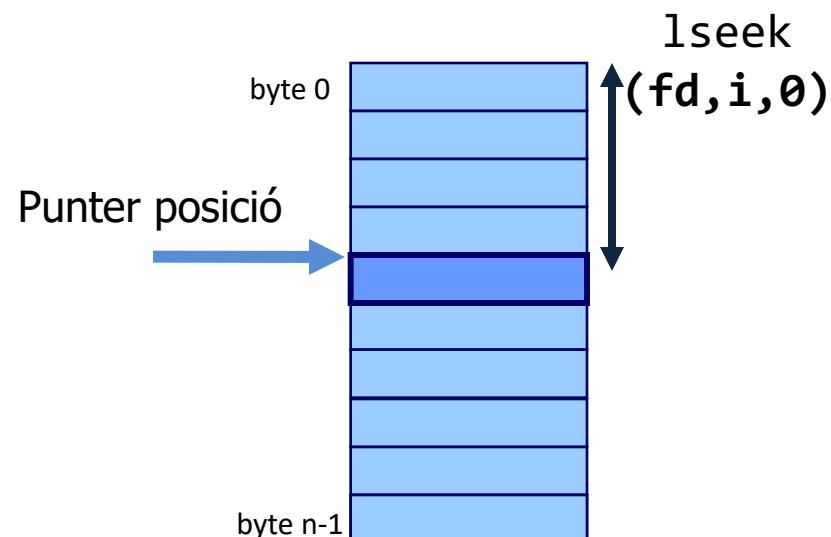
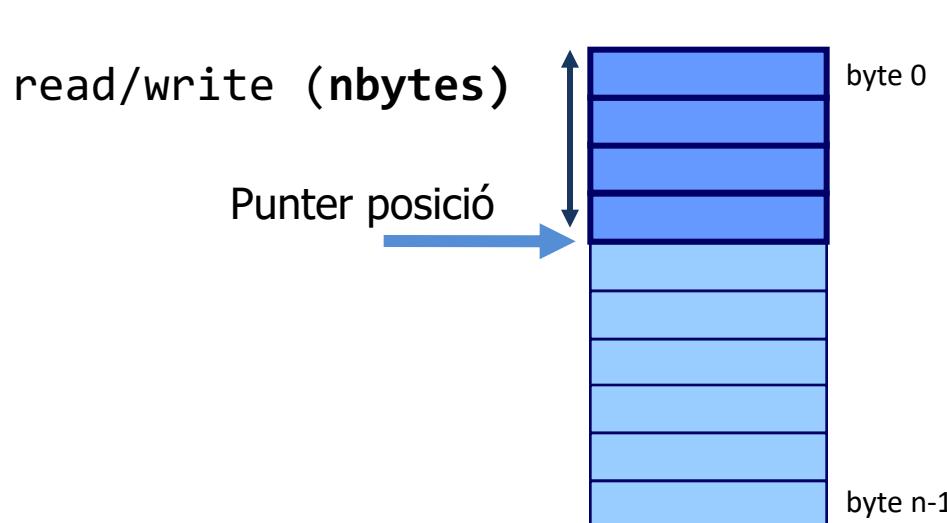
- **Mode d'accés en Unix**

- Accés seqüencial amb les crides **read/write**:

- **read/write (fd, buffer, nbytes)**

- La crida **Iseek** permet realitzar accés directe especificant un offset des del principi, des del final de l'arxiu o des de la posició actual.

- **Iseek (fd, offset, where)**



- **Descriptors d'arxius**

- Per a treballar amb un arxiu cal definir una sessió de treball amb les crides ***open*** (obrir) i ***close*** (tancar).

<b>Obrir:</b>	<b><i>fd</i></b> = <b><i>open</i></b> ( <i>filename, mode</i> )	( <i>obrir</i> )
	<b><i>read</i></b> ( <i>fd, ...</i> ) , <b><i>write</i></b> ( <i>fd, ...</i> ) ,	
	<b><i>lseek</i></b> ( <i>fd, ...</i> ) , ...	( <i>manipular</i> )
<b>Tancar:</b>	<b><i>close</i></b> ( <i>fd</i> )	( <i>tancar</i> )

- Un **descriptor d'arxiu** és un identificador abstracte de l'arxiu particular per a cada procés
  - La crida ***open*** retorna un descriptor d'arxiu (fd)
  - En realitat, es tracta d'un tipus numèric
- Les crides de manipulació d'arxius requereixen un descriptor per a identificar-lo
  - Treballar amb descriptors d'arxius fa més eficient l'accés als arxius i evita cercar-los en disc a cada operació

- **Descriptoros d'arxiu**

- **Semàntica d'obrir un arxiu** (open)

- Cercar l'arxiu en l'estructura de directoris i dur els seus atributs a una entrada d'una **taula d'arxius oberts** en memòria
    - Registrar alguns atributs addicionals com:
      - Punter de posició
      - Nombre d'obertures
      - Ubicació de la informació en disc
    - Transferir part del contingut de l'arxiu del disc a buffers en memòria
  - **Semàntica de tancar arxiu** (close)
    - Alliberar l'entrada corresponent en la taula d'arxius oberts

- Taula d'obertures del sistema versus taula de descriptors d'arxius**



Taula descriptors d'arxius (P1)

0	STD_INPUT
1	STD_OUTPUT
2	STD_ERROR
3	file.txt
4	

```
//codi de P1
.....
fd = open("file.txt", O_RDONLY)
```

Tabla de aperturas

Mode: RONLY	Punter posició
Mode: RDWR	Punter posició

Taulade nodes-i

Node-i file.txt	count= 1
Node-i filew.txt	count= 1

Cache del sistema de archivos

Dades	file.txt
Dades	file.txt
Dades	filew.txt

Taula de descriptors d'arxius:

- És pròpia de cada procés
- Es troba en l'àrea del procés
- El procés accedeix a ella amb crides al sistema



Taula descriptors d'arxius (P2)

0	STD_INPUT
1	STD_OUTPUT
2	STD_ERROR
3	filew.txt
4	

```
//codi de P2
.....
fd = open("filew.txt", O_RDWR)
.....
```

**¡¡La taula d'obertures és única per a tot el sistema!!**

- Conté una entrada per cada open() actiu
- Compartida per tots els processos del sistema
- Accessible només pel SO

- **Descriptors d'arxiu i entrada/eixida estàndard**
  - Els tres primers descriptors d'un procés tenen nom propi:

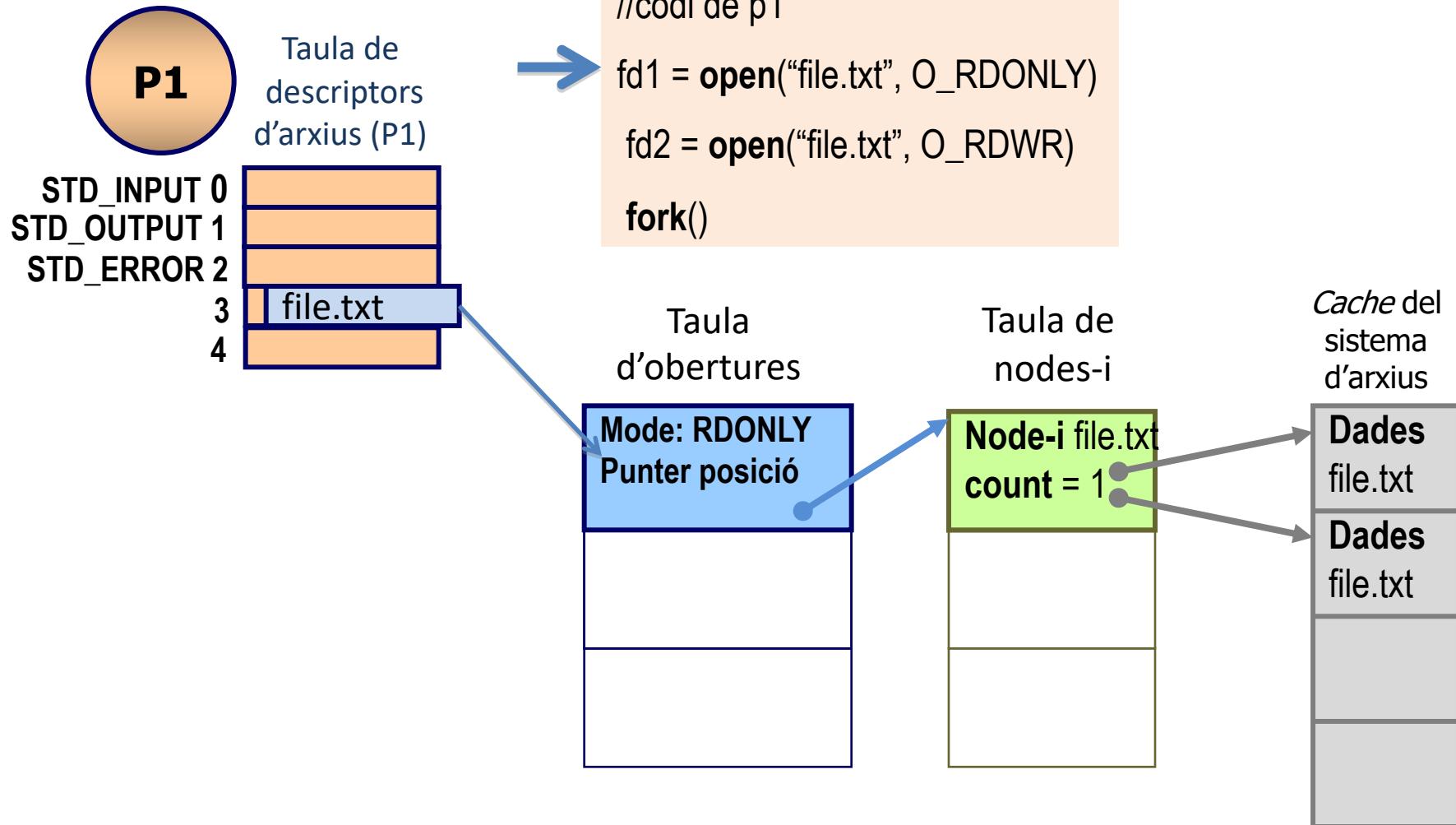
descriptor	nom en <i>unistd.h</i>	Dispositiu físic
0	STDIN_FILENO	<i>stdin</i> , entrada estàndard
1	STDOUT_FILENO	<i>stdout</i> , eixida estàndard
2	STDERR_FILENO	<i>stderr</i> , eixida d'errors

- Per omissió, els processos troben aquests descriptors associats a la consola.
  - És a dir, estan associats a */dev/ttyn* o a */dev/ptn/n*
  - Aquesta associació es pot modificar mitjançant redireccions i tubs
- Exemples d'ús
  - Des de la biblioteca de C: La funció *scanf* llig de l'entrada estàndard i la funció *printf* escriu en l'eixida estàndard.
  - Des del Shell: les ordres lligen i escriuen per l'E/S estàndard. L'ordre *ls* escriu el llistat d'arxius per l'eixida estàndard i el missatge "No such file or directory" per l'eixida d'errors



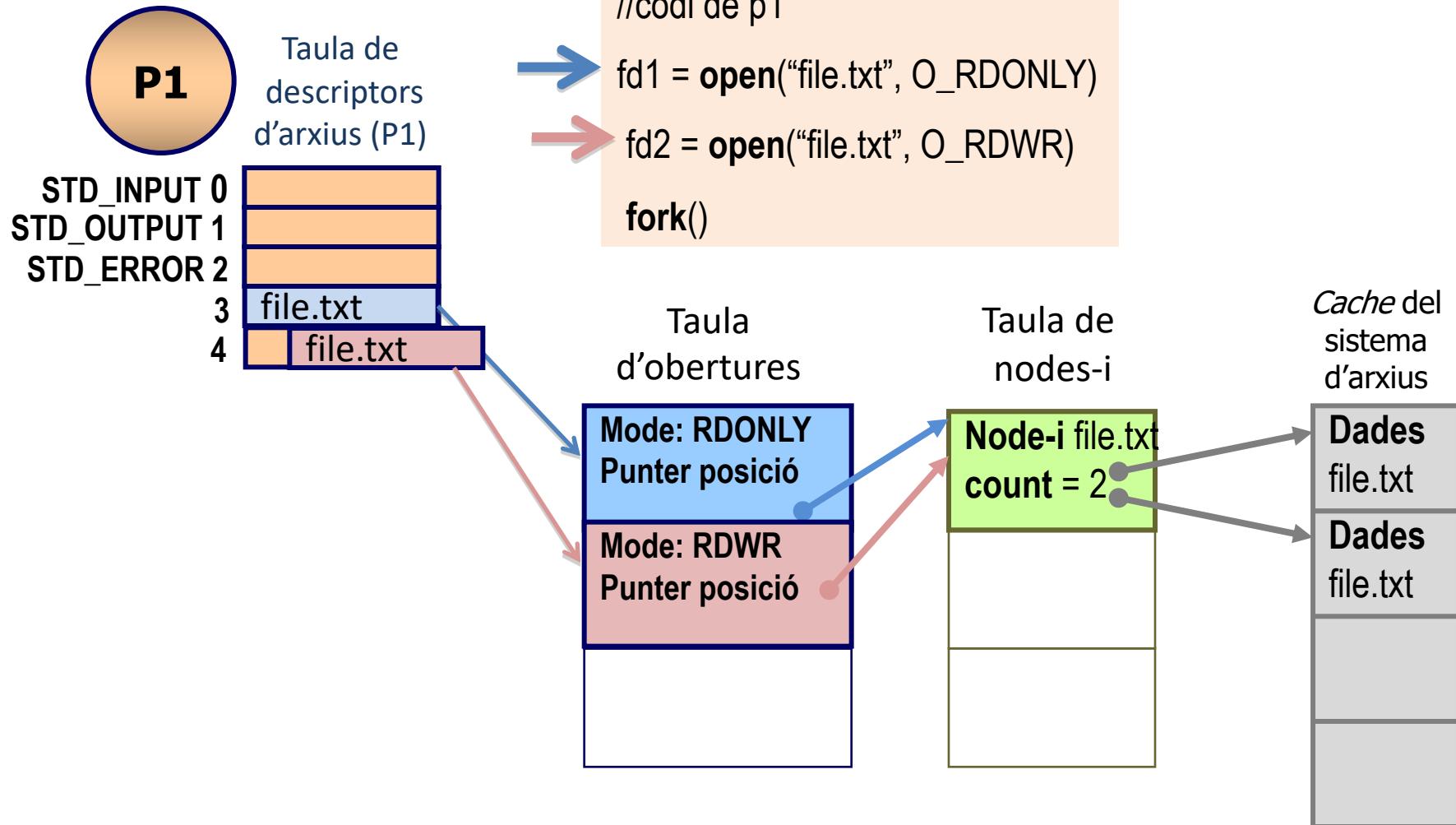
Taula descriptors D'arxius (P1)	
0	STD_IN
1	STD_OUT
2	STD_ERROR
3	file.txt
4	

## • Taula de descriptors d'arxius

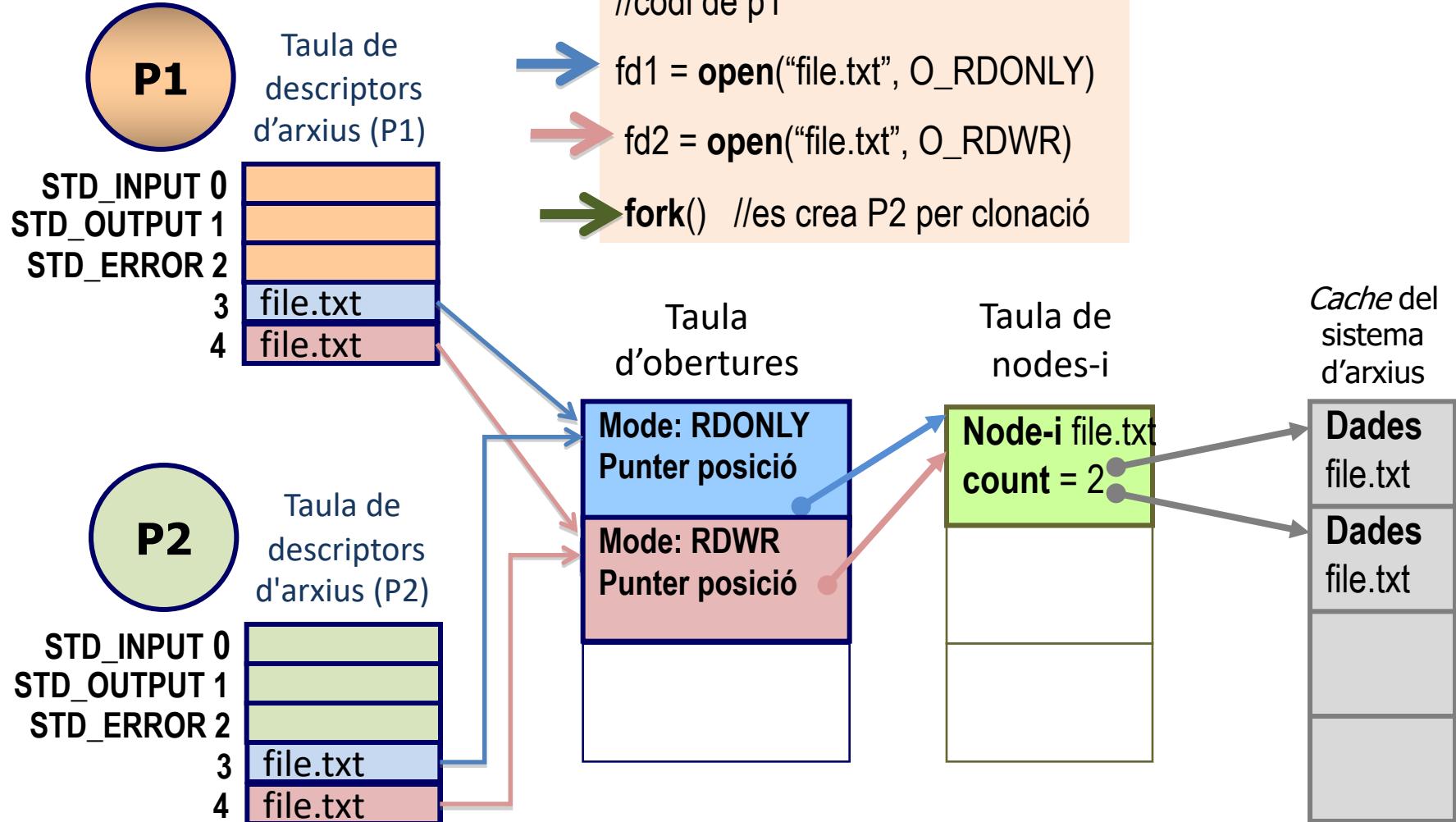


¡¡La taula d'obertures és única per a tot el sistema!!

- Taula de descriptors d'arxius



## • Taula de descriptors d'arxius



!!!Els descriptors d'arxius oberts són atributs heretables

- **Contingut**

- Arxius en Unix
- **Crides al sistema per a arxius**
- Redireccions i tubs (“*pipe*”)
- Crides per a redireccions i tubs
- Exemples en C

- Crides per a treballar amb arxius o dispositius
  - Unix té una única interfície per a treballar amb els dispositius

	Descripció
<b>open</b>	Obertura/creació d'arxius
<b>read</b>	Lectura en arxius
<b>write</b>	Escriptura en arxius
<b>close</b>	Clausura d'un arxiu
<b>lseek</b>	Posicionament en un arxiu
<b>stat</b>	Obtindre informació del node-i d'un arxiu

- Les crides de lectura i escriptura en arxiu no fan conversió de format.
  - Per tant, les funcions de biblioteca que fan entrada/eixida formatada (*scanf* i *printf*) inclouen el codi que se n'encarrega a més de les crides al sistema

## open: obertura/creació arxius

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

int open(const char *path, int flags)
int open(const char *path, int flags, mode_t mode)
```

### Descripció

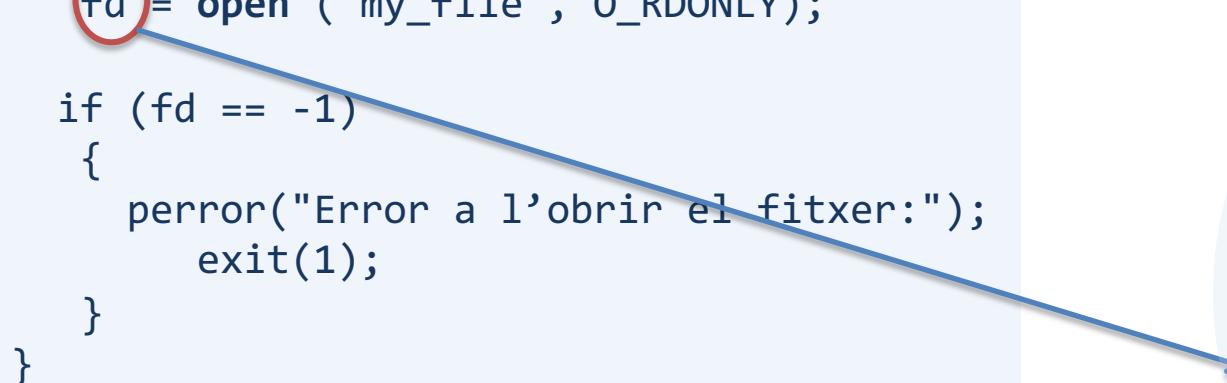
- Associa un descriptor d'arxiu amb arxiu o dispositiu físic
  - Assigna sempre el descriptor mes baix que es trobe lliure en la taula de descriptors d'arxius
  - Els **descriptors d'arxius oberts** són **atributs heretables**
- **flags**
  - O\_RDONLY, O\_WRONLY, O\_RDWR
  - O\_CREAT, O\_EXCL, O\_TRUNC, O\_APPEND
- **mode** Exemples: 0755, 04755
  - S\_IRUSR, S\_IWUSR, S\_IXUSR, S\_IRWXU
  - S\_IRGRP, S\_IWGRP, S\_IXGRP, S\_IRWXG
  - S\_IROTH, S\_IWOTH, S\_IXOTH, S\_IRWXO
  - S\_ISUID, S\_ISGID

- Open: obertura/creació d'un arxiu
  - Exemple d'obertura d'un arxiu

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

main ( int argc, char* argv[] )
{int fd;
    fd = open ("my_file", O_RDONLY);

    if (fd == -1)
    {
        perror("Error a l'obrir el fitxer:");
        exit(1);
    }
}
```



0	/dev/tty0
1	/dev/tty0
2	/dev/tty0
3	my_file
4	

¡L'arxiu “my\_file” ha utilitzat la posició 3 de la taula de descriptors → fd=3!

## read/write: lectura/escriptura arxius

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t nbytes)
ssize_t write(int fd, const void *buf, size_t nbytes)
```

### Descripció

- **read:** sol·licita llegir *nbyte* bytes de l'arxiu amb descriptor *fd*
  - Els bytes llegits s'emmagatzemen en *buf*
  - *read* pot llegir menys bytes dels sol·licitats si aplega a final de arxiu
- **write:** sol·licita escriure *nbyte* bytes presos del buffer *buf* en l'arxiu amb descriptor *fd*
- Per defecte *read* i *write* són bloquejants i poden ser interrompudes per un senyal
- **Valor de retorn**
  - **un sencer >0**, que correspon al nombre de bytes escrits/llegits
  - **-1 : error** o interrupció per senyal (errors: *fd* no és un descriptor vàlid, *buf* no és una adreça vàlida, disc ple, operació no permesa, etc.)
  - **0**: intent de llegir després de final de arxiu

**close:** clausura un arxiu

```
#include <unistd.h>

int close(int fd)
```

## Descripció

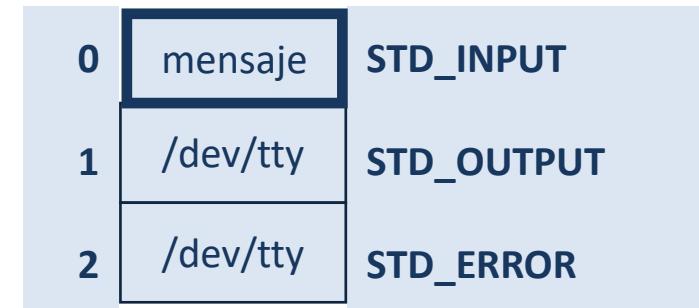
- Tanca el descriptor d'arxiu fd, alliberant eixa posició en la taula de descriptors d'arxiu
- Valor de retorn
  - Nova posició del punter d'accés
  - -1 en cas d'error (error EBADF: el descriptor fd no és vàlid)

- **Contingut**

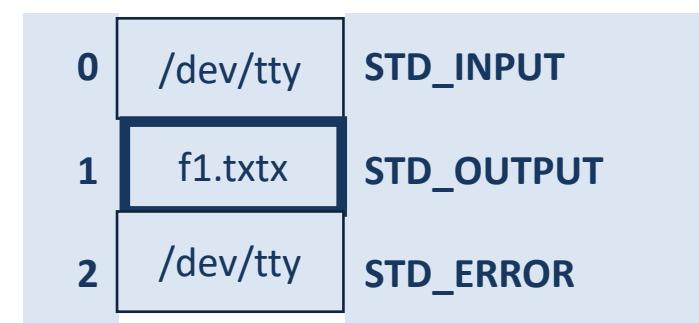
- Arxius en Unix
- Crides al sistema per a arxius
- **Redireccions i tubs (“pipe”)**
- Crides per a redireccions i tubs
- Exemples en C

- Redirecció de l'entrada/sortida estàndard des de l'intèrpret d'ordres

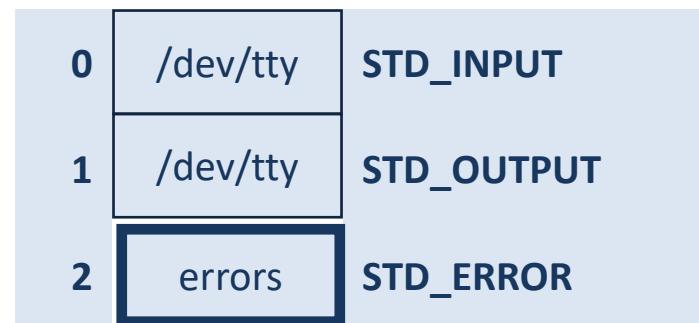
- Redirecció d'entrada estàndard  
`$ mail gandreu < mensaje`



- Redirecció de la sortida estàndard  
`$ echo hola > f1.txt`



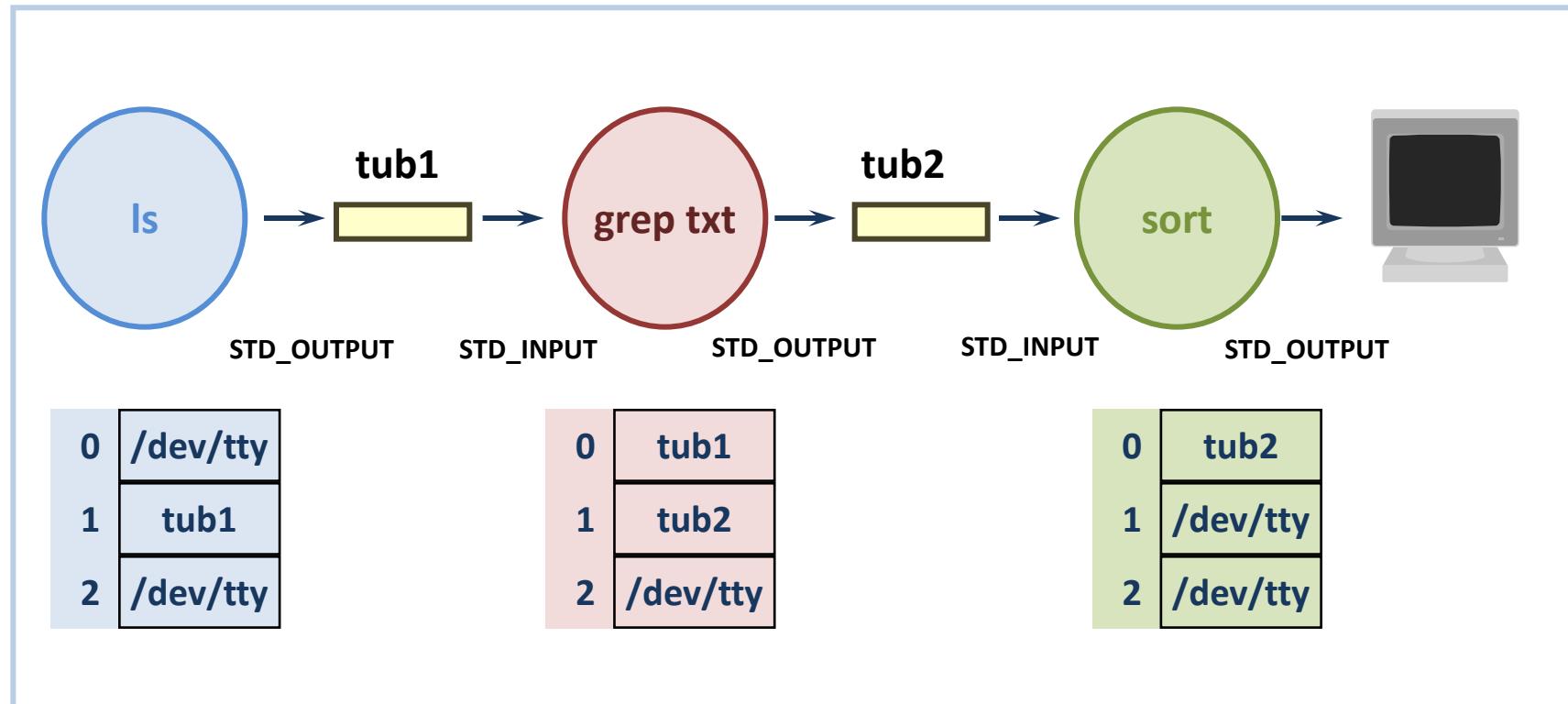
- Redirecció de la sortida d'error  
`$ gcc prg.c -o prg 2> error`



- Comunicació de processos Unix

- La comunicació entre processos en Unix es fa mitjançant tubs
- **Tubs (“PIPE”)**
  - són un tipus especial d'arxiu de capacitat limitada amb accés seqüencial
  - poden compartir-se gràcies al mecanisme d'herència

```
$ ls | grep txt | sort
```



- **Contingut**

- Arxius en Unix
- Crides al sistema per a arxius
- Redireccions i tubs (“pipe”)
- **Crides per a redireccions i tubs**
- Exemples en C

- **Crides per a redireccions i tubs**
  - Ens permetrà establir comunicació entre processos pare i fill a través del mecanisme d'herència

	<b>Descripció</b>
<b>dup2</b>	Duplicar un descriptor de arxiu
<b>pipe</b>	Creació d'un tub
<b>mkfifo</b>	Creació d'un tub amb nom (fifo)

## dup, dup2: duplicar un descriptor d'arxiu

```
#include <unistd.h>
int dup(int fd)
int dup2(int oldfd, int newfd)
```

### Descripció

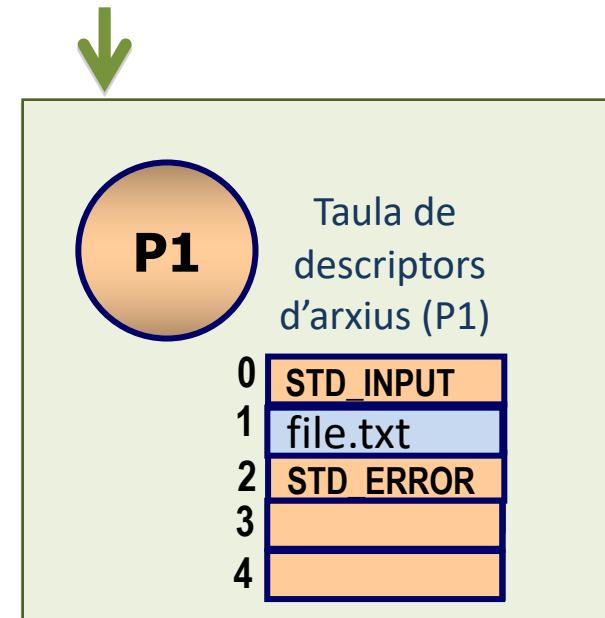
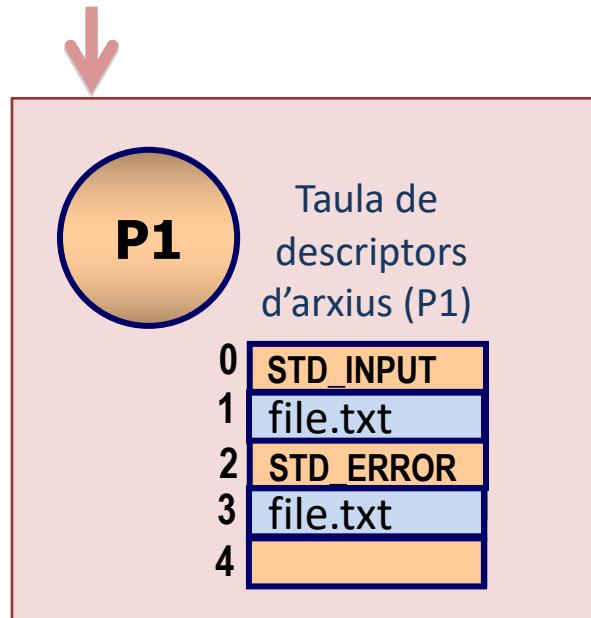
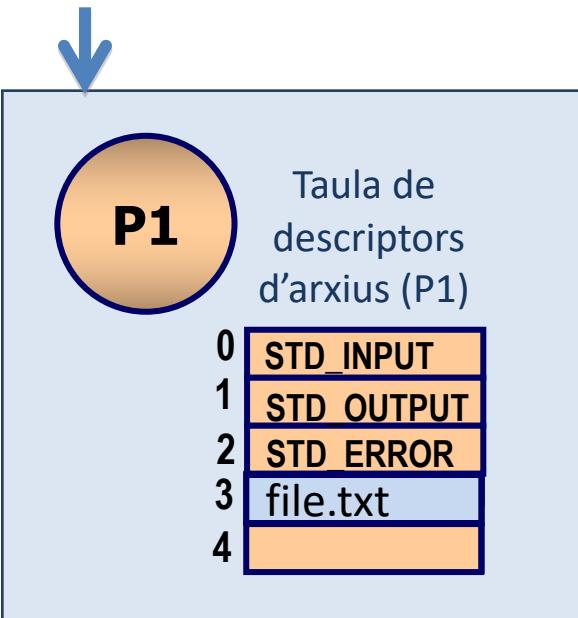
- **dup**: retorna un nombre de descriptor d'arxiu el contingut del qual és una còpia del que es correspon amb el paràmetre **fd**
  - El descriptor que retorna és el mes baix que està disponible en la taula de descriptors
- **dup2**: **tanca** el descriptor que correspon a **newfd** i després **còpia** l'entrada del corresponent al primer paràmetre **oldfd** **sobre** la del segon paràmetre **newfd**.
- Valor de retorn
  - Nou descriptor de arxiu
  - -1 en cas d'**error** ( **fd** no és descriptor vàlid. Se supera el màxim de arxius oberts (**OPEN\_MAX**)

## Exemple: dup2

```
//codi de p1
#define NEWFILE (O_WRONLY | O_CREAT | O_TRUNC)
#define MODE644 (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)
→ fd = open("file.txt", NEWFILE, MODE644)
→ dup2(fd, STDOUT_FILENO);
→ close(fd);
```



0	STD INPUT
1	STD OUTPUT
2	STD ERROR
3	
4	

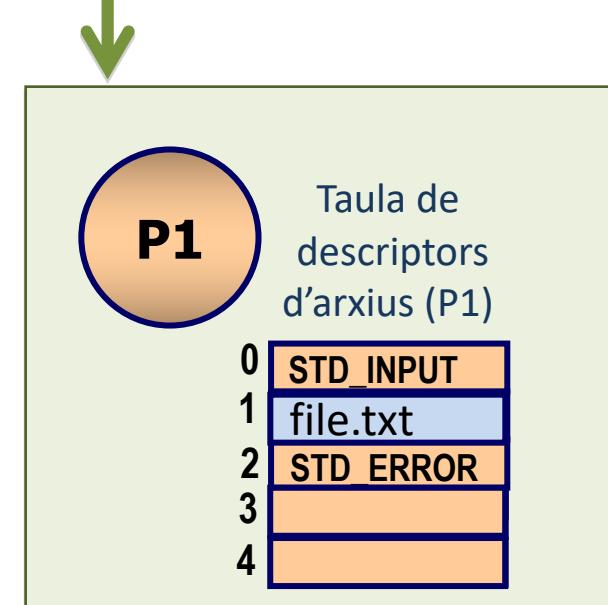
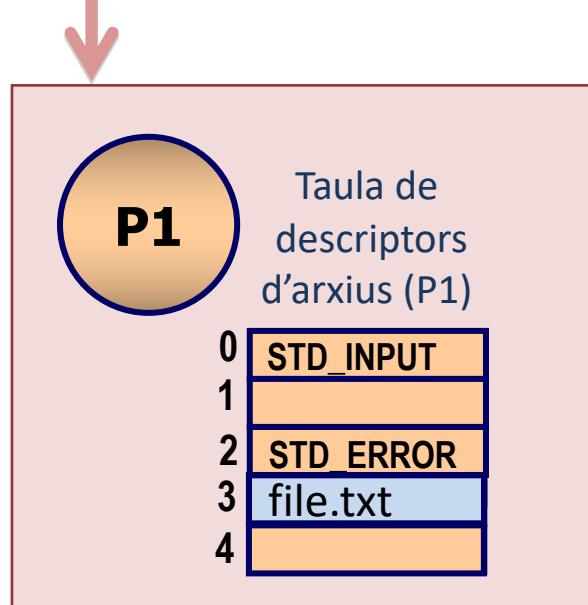
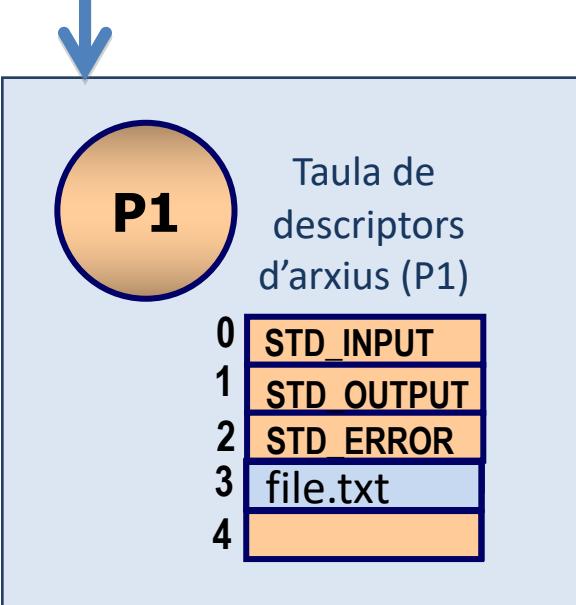


## Exemple: dup

```
//codi de p1  
  
#define NEWFILE (O_WRONLY | O_CREAT | O_TRUNC)  
  
#define MODE644 (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)  
  
→ fd = open("file.txt", NEWFILE, MODE644);  
→ close (STDOUT_FILENO);  
→ dup(fd);
```

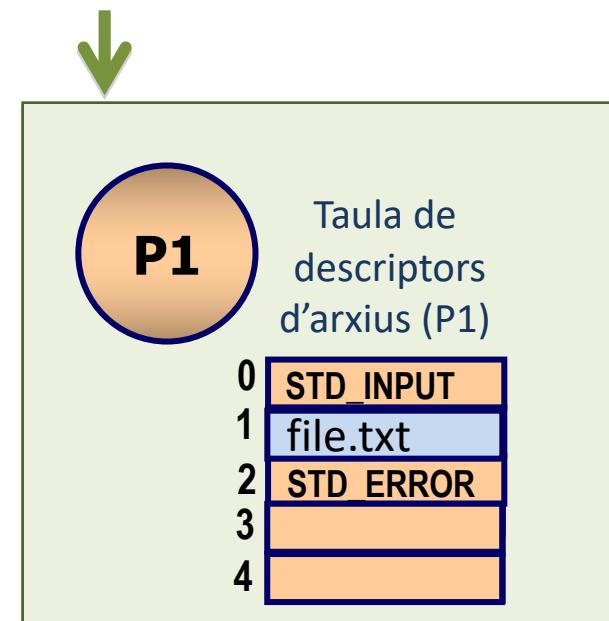
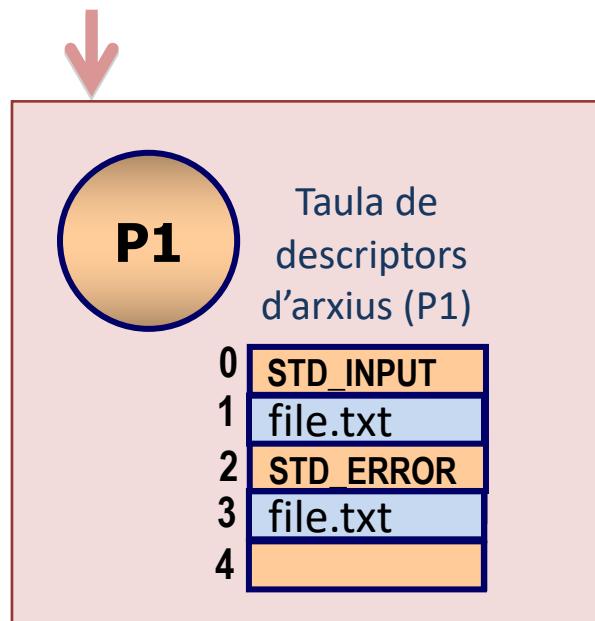
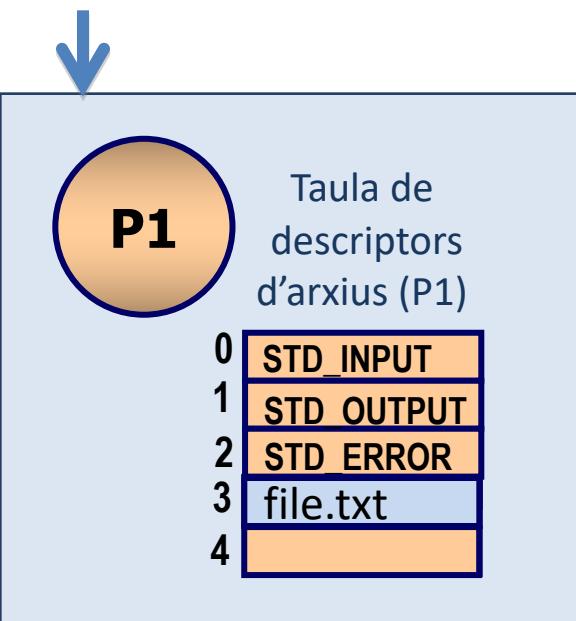


0	STD_INPUT
1	STD_OUTPUT
2	STD_ERROR
3	
4	



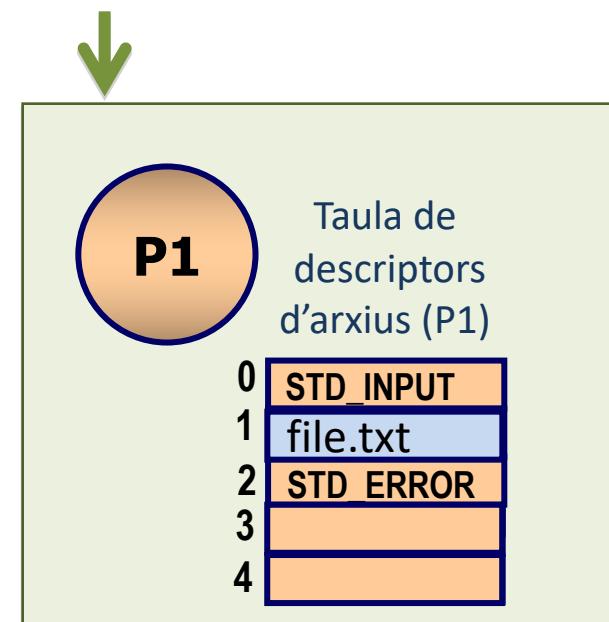
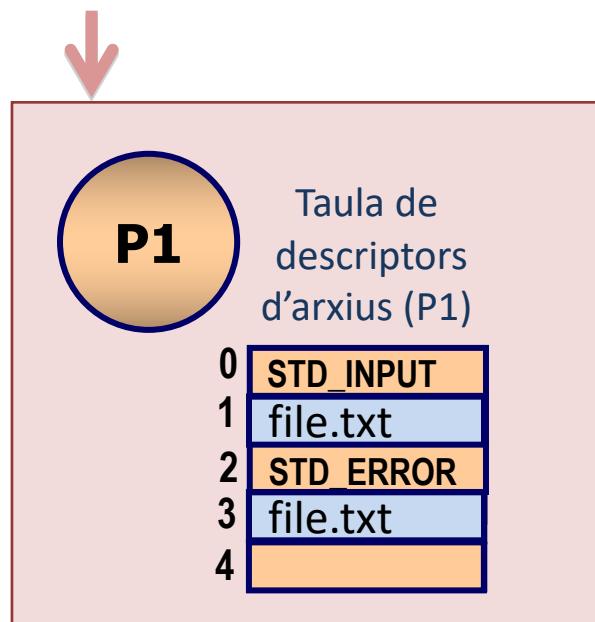
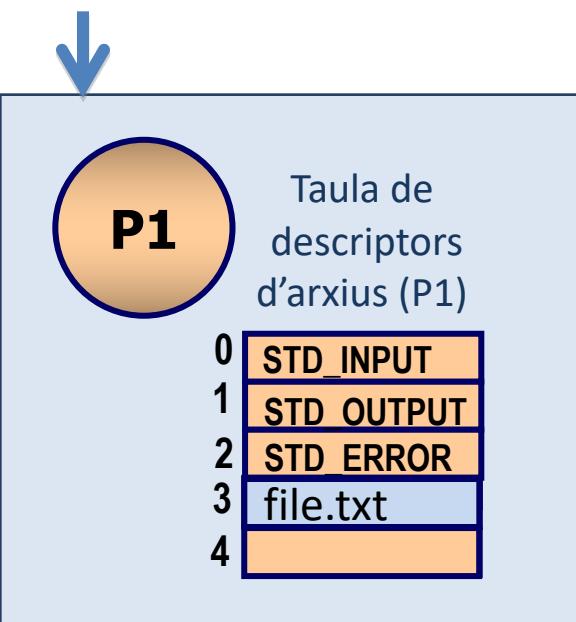
## Exemple: dup, dup2

```
//codi de p1  
  
#define NEWFILE (O_WRONLY | O_CREAT | O_TRUNC)  
  
#define MODE644 (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)  
  
→ fd = open("file.txt", NEWFILE, MODE644);  
→ dup2(fd, STDOUT_FILENO);  
→ close(fd);
```



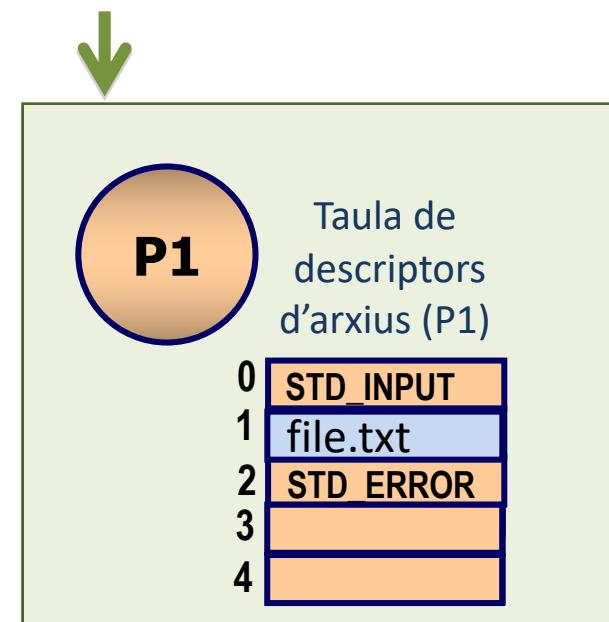
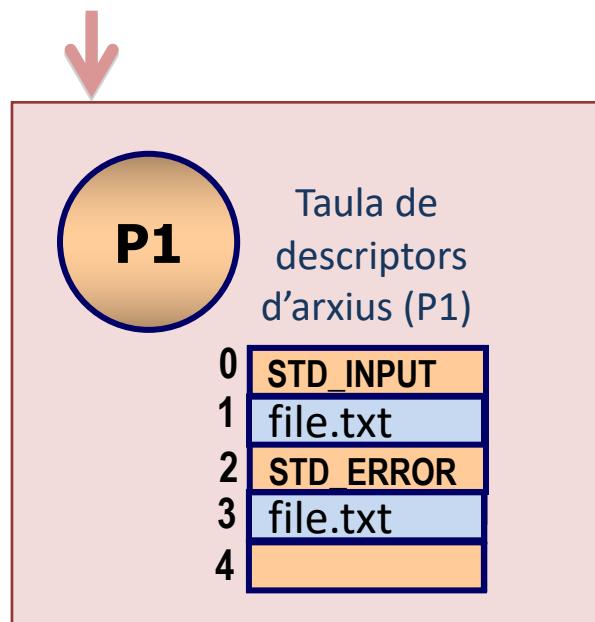
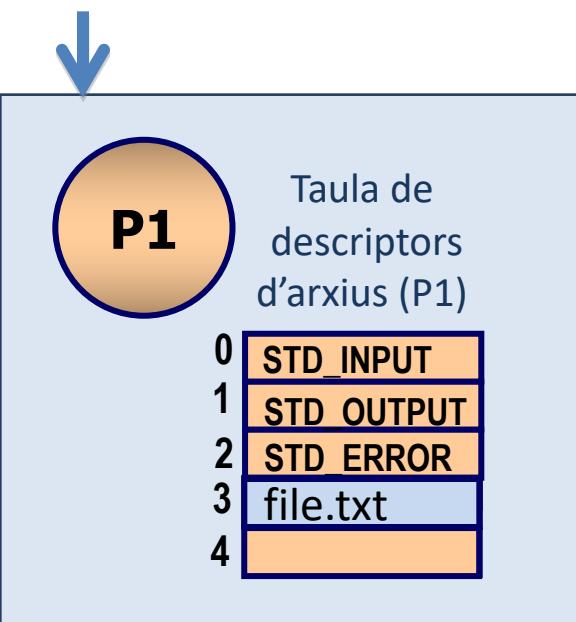
## Exemple: dup, dup2

```
//codi de p1
#define NEWFILE (O_WRONLY | O_CREAT | O_TRUNC)
#define MODE644 (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)
→ fd = open("file.txt", NEWFILE, MODE644)
→ dup2(fd, STDOUT_FILENO);
→ close(fd);
```



## Exemple: dup, dup2

```
//codi de p1
#define NEWFILE (O_WRONLY | O_CREAT | O_TRUNC)
#define MODE644 (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)
→ fd = open("file.txt", NEWFILE, MODE644)
→ dup2(fd, STDOUT_FILENO);
→ close(fd);
```



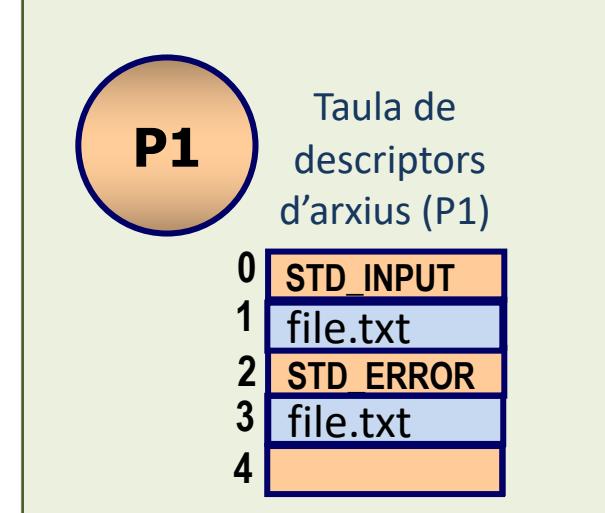
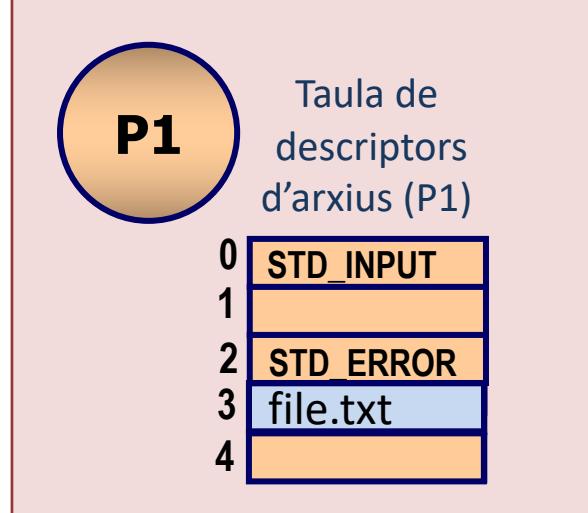
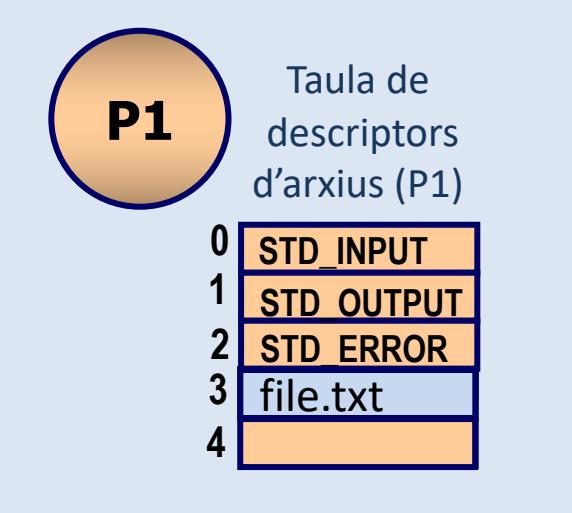
## Exemple: dup, dup2

```
//codi de p1
#define NEWFILE (O_WRONLY | O_CREAT | O_TRUNC)
#define MODE644 (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)

→ fd = open("file.txt", NEWFILE, MODE644)
→ dup (STDOUT_FILENO);
→ close (fd);
```



dup2 (fd, STDOUT\_FILENO);



## pipe: creació d'un tub

```
#include <unistd.h>
int pipe(int fildes[2])
```

### Descripció

- Crea un *pseudoarxiu*, denominat **tub**, l'estructura del qual és una cua FIFO de bytes inicialment buida
  - Són un mecanisme de comunicació entre processos UNIX
  - La capacitat màxima del tub està limitada per la implementació
- **Després de la crida,**
  - **fildes[0]** és un **descriptor** d'arxiu per a **lectura** del tub
  - **fildes[1]** és un **descriptor** d'arxiu per a **escriptura del tub**
- Els tubs junt amb els descriptors d'accés són heretats pels processos fills i preservats en executar exec.
- Valor de retorn
  - 0 si funciona amb èxit
  - -1 en cas d'error (Se supera el màxim d'arxius oberts (OPEN\_MAX), **fildes** no és vàlid)

## pipe: funcionament de read i write sobre tubs:

### – *read()*

- Si existeixen bytes disponibles, se lligen como a molt els nbytes sol·licitats.
- Si el **tub** està **buit**, **read suspen al procés** que l'invoca fins que existisca disponibilitat de bytes en el tub
- Quan no existeix cap descriptor d'escriptura associat al tub (del procés lector o de algun altre procés) read no suspen i retorna 0, indicant així la condició de final de dades (final de arxiu EOF).

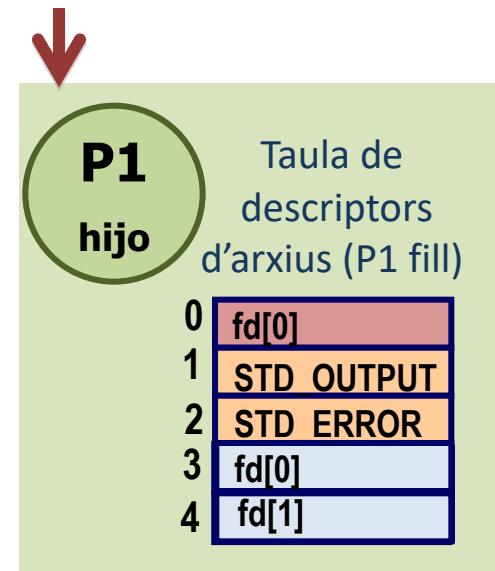
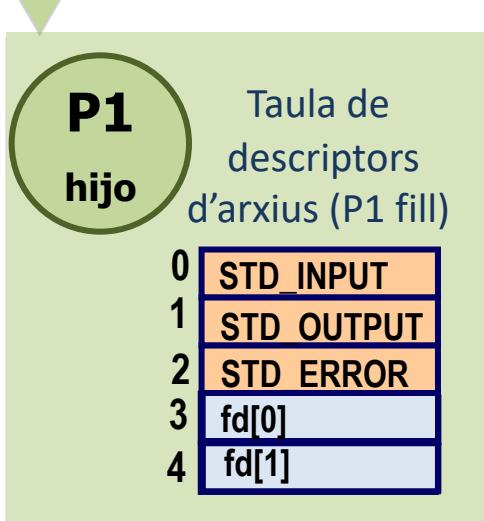
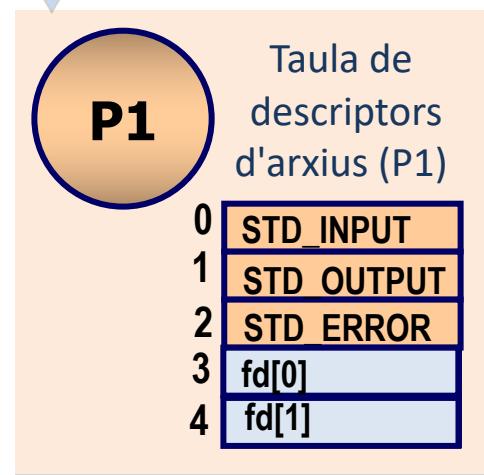
### – *write()*

- Si existeix capacitat suficient en el tub per a allotjar els nbytes sol·licitats, aquests són emmagatzemats en el tub en ordre FIFO.
- Si no existeix capacitat suficient per a allotjar els nbytes sol·licitats (tub ple), el procés escriptor es suspen fins que hi haja disponibilidad d'espai.
- Si s'intenta **escriure en un tub que no té cap descriptor de lectura associado** (del procés escriptor o de algun altre procés), **el procés que intenta escriure reb el senyal SIGPIPE**.
  - Aquest mecanisme facilita la eliminació automàtica d'una cadena de processos comunicats per tubs quan s'aborta inesperadament un component d'aquesta.

# Crides per a redireccions i tubs

fso

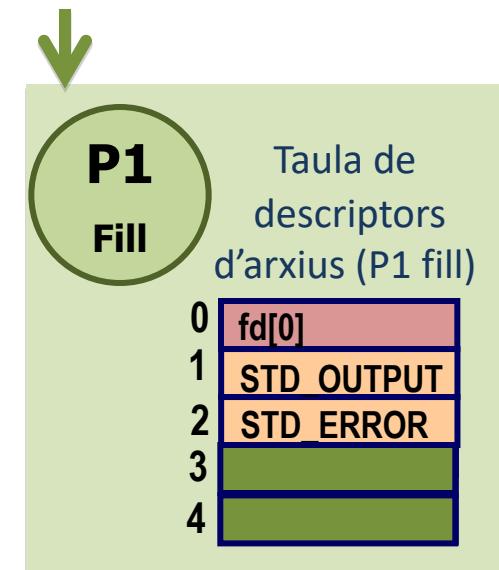
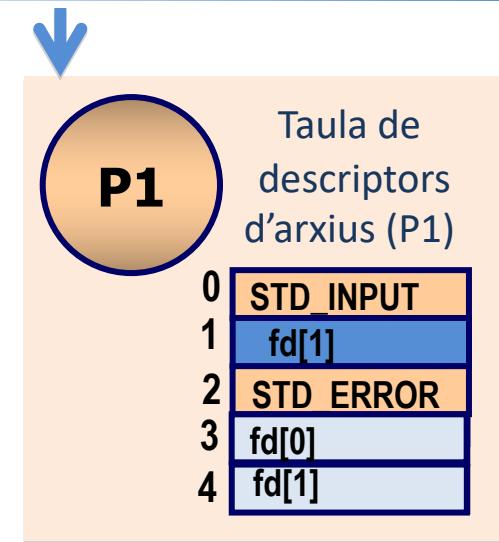
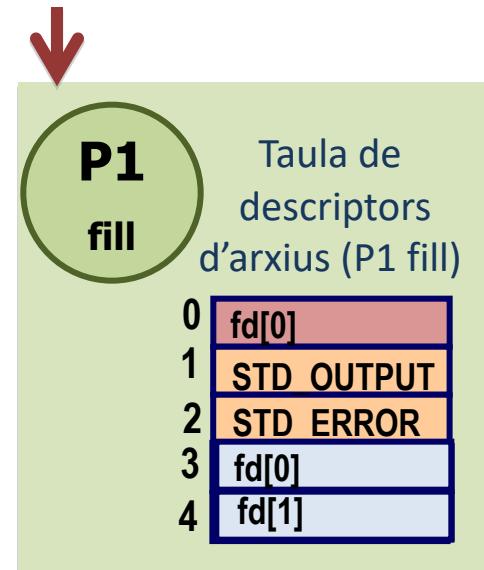
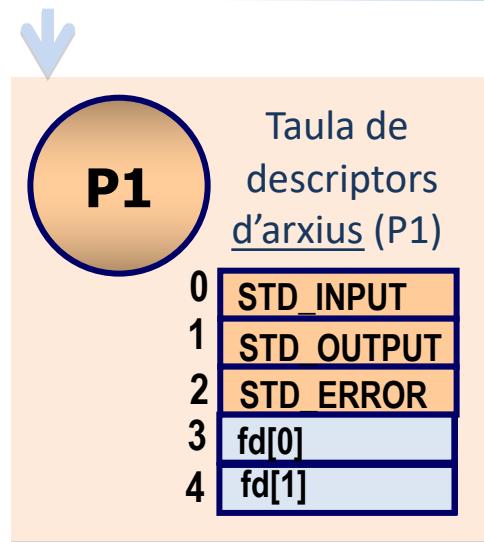
```
//codi procés p1
pipe(fd);
if (fork() == 0)
{
    // codi del procés fill
    dup2 (fd[0], STDIN_FILENO);
    close (fd[0]);
    close (fd[1]);
    ...
}
else{
    // codi del procés pare
    dup2 (fd[1], STDOUT_FILENO);
    close (fd[0]);
    close (fd[1]);
    ...
}
```



# Crides per a redireccions i tubs

fSO

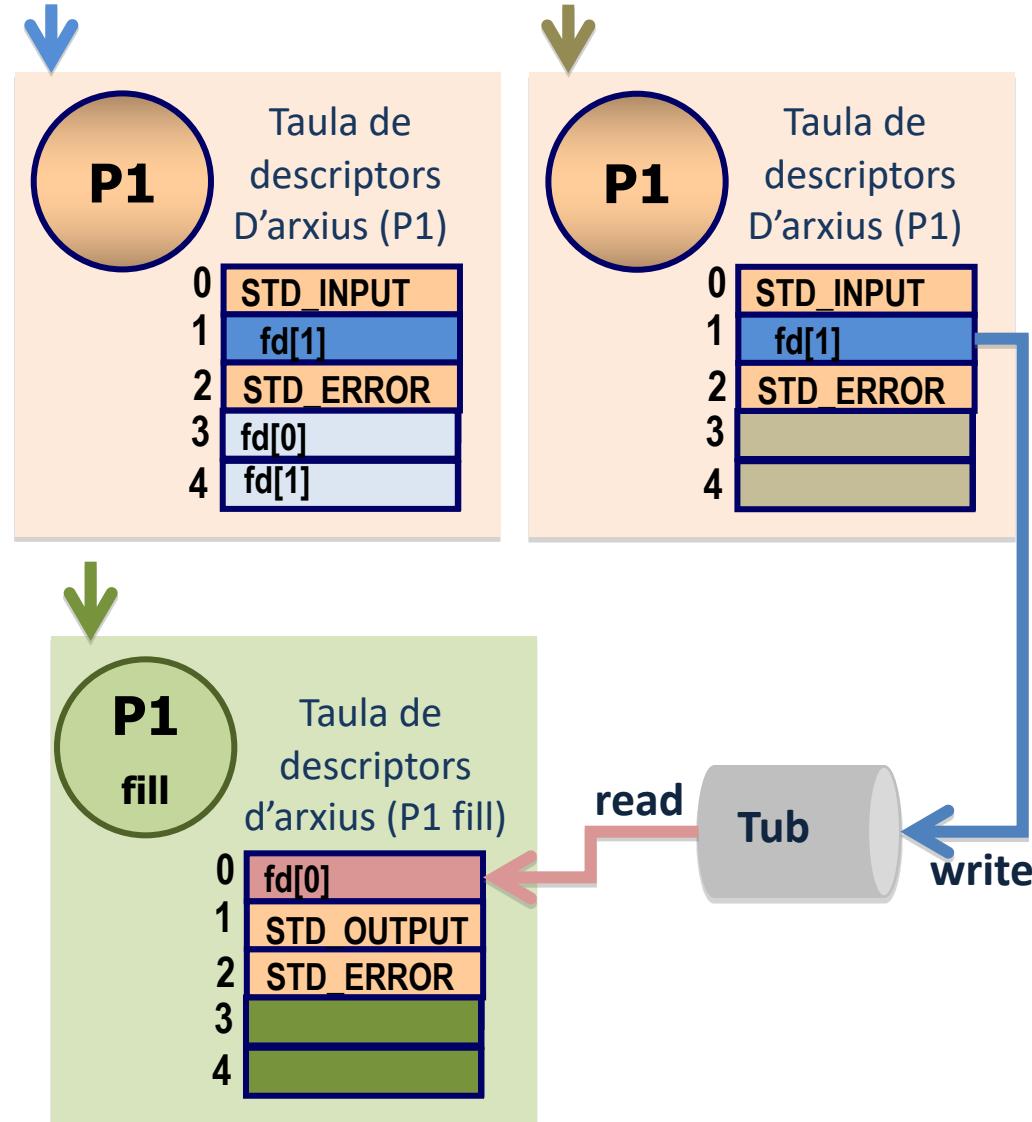
```
//codi procés p1
pipe(fd);
if (fork() == 0)
{
    // codi del procés fill
    → dup2 (fd[0], STDIN_FILENO);
    close (fd[0]);
    → close (fd[1]);
    ...
}
else{
    // codi del procés pare
    dup2 (fd[1], STDOUT_FILENO);
    close (fd[0]);
    close (fd[1]);
    ...
}
```



# Crides per a redireccions i tubs

fso

```
//codi procés p1
pipe(fd);
if (fork() == 0)
{
    // codi del procés fill
    dup2 (fd[0], STDIN_FILENO);
    close (fd[0]);
    close (fd[1]);
    ...
}
else{
    // codi del procés pare
    dup2 (fd[1], STDOUT_FILENO);
    close (fd[0]);
    close (fd[1]);
    ...
}
```



- **Contingut**

- Arxius en Unix
- Crides al sistema per a arxius
- Redireccions i tubs (“pipe”)
- Crides per a redireccions i tubs
- **Exemples en C**

# Exemples en C

- Exemple: open, write,read**

```

int main(int argc, char *argv[]) {
    int from_fd, to_fd;
    int count;
    char buf[BLKSIZE];

    if (argc != 3) {
        fprintf(stderr, "Usage: %s from_file to_file\n", argv[0]);
        exit(1);
    }
    if ((from_fd = open(argv[1], O_RDONLY)) == -1) {
        fprintf(stderr, "Could not open %s: %s\n", argv[1], strerror(errno));
        exit(1);
    }
    if ((to_fd = open(argv[2], NEWFILE, MODE600)) == -1) {
        fprintf(stderr, "Could not create %s: %s\n", argv[2], strerror(errno));
        exit(1);
    }
    while ((count = read(from_fd, buf, sizeof(buf))) > 0) {
        if (write(to_fd, buf, count) != count) {
            fprintf(stderr, "Could not write %s: %s\n", argv[2], strerror(errno));
            exit(1);
        }
    }
    if (count == -1) {
        fprintf(stderr, "Could not read %s: %s\n", argv[1], strerror(errno));
        exit(1);
    }
    close(from_fd);
    close(to_fd);
    exit(0);
}

```

```

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

#define BLKSIZE 1
#define NEWFILE (O_WRONLY | O_CREAT | O_EXCL)
#define MODE600 (S_IRUSR | S_IWUSR)

```

Per a compilar i executar:

```

$ gcc my_copy.c -o my_copy
$ echo 'Hola read write' > hola.txt
$ ./my_copy hola.txt hola_copia.txt
$ cat hola_copia.txt

```

# Exemples en C

- **Exemple : dup2**

```

int redirect_output(const char *file) {
    int fd;
    if ((fd = open(file, NEWFILE, MODE644)) == -1) return -1;
    if (dup2(fd, STDOUT_FILENO) == -1) return -1;
    close(fd);
    return 0;
}
int main(int argc, char *argv[]) {
    int from_fd, to_fd;
    if (argc < 3) {
        fprintf(stderr, "Usage: %s to_file command args\n", argv[0]);
        exit(1);
    }
    if (redirect_output(argv[1]) == -1){
        fprintf(stderr, "Could not redirect output to: %s\n", argv[1]);
        exit(1);
    }
    if (execvp(argv[2], &argv[2]) < 0){
        fprintf(stderr, "Could not execute: %s\n", argv[2]);
        exit(1);
    }
    return 0;
}

```

```

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

```

```

#define NEWFILE (O_WRONLY | O_CREAT | O_EXCL)
#define MODE644 (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)

```

Per a compilar i executar:

```

$ gcc dup2.c -o dup2
$ ./dup2 prueba ls
$ cat prueba

```

# Exemples en C

- **pipe:** exemple1

```

int main(int argc, char *argv[]) {
    int i, fd[2];
    if (argc < 2) {
        fprintf(stderr, "Usage: %s command filter\n", argv[0]);
        exit(1);
    }
    pipe(fd);
    for(i=0; i<2; i++) {
        if (fork() == 0) { // fills
            dup2 (fd[1], STDOUT_FILENO);
            close (fd[0]);
            close (fd[1]);
            execl("/bin/ls", "ls", NULL);
            perror("The exec of ls failed");
        }
    }
    // padre
    dup2 (fd[0], STDIN_FILENO);
    close (fd[0]);
    close (fd[1]);
    execvp(argv[1],&argv[1]);
    fprintf(stderr,"The exec of %s failed", argv[1]);
    exit(1);
}

```

```

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

```

Per a compilar i executar:

```

$ gcc pipe.c -o pipe
$ ./pipe wc

```