

Grupo ARCOS
Universidad Carlos III de Madrid

Lección 3

Señales, excepciones y pipes

Sistemas Operativos
Ingeniería Informática



A recordar...

Antes de clase

Clase

Después de clase

Preparar los pre-requisitos.

Estudiar el material asociado a la **bibliografía**:
las transparencias solo no son suficiente.
Preguntar dudas (especialmente tras estudio).

Ejercitarse las competencias:

- ▶ Realizar todos los **ejercicios**.
- ▶ Realizar **laboratorios** y **prácticas** de forma progresiva.

Lecturas recomendadas

Base



1. **Carretero 2020:**
 1. Cap. 5
2. **Carretero 2007:**
 1. Cap. 3.6 y 3.7
 2. Cap. 3.9 y 3.13

Recomendada



1. **Tanenbaum 2006:**
 1. (es) Cap. 2.2
 2. (en) Cap.2.1.7
2. **Stallings 2005:**
 1. 4.1, 4.4, 4.5 y 4.6
3. **Silberschatz 2006:**
 1. 4

Contenidos

1. Señales y excepciones.
2. Temporizadores.
3. Entorno de un proceso.
4. Comunicación de procesos con tuberías (pipes).
Paso de mensajes local.

Contenidos

1. Señales y excepciones.
2. Temporizadores.
3. Entorno de un proceso.
4. Comunicación de procesos con tuberías (*pipes*).
Paso de mensajes local.

- Interprete de mandatos: redirección y tuberías
- Los descriptores de ficheros
 - Modelo: abstracción ofrecida
 - Redirección
 - Duplicación
 - *fork()*
- Tuberías o *pipes*

Contenidos

1. Señales y excepciones.
2. Temporizadores.
3. Entorno de un proceso.
4. Comunicación de procesos con tuberías (pipes).
Paso de mensajes local.

- **Interprete de mandatos: redirección y tuberías**
- Los descriptores de ficheros
 - Modelo: abstracción ofrecida
 - Redirección
 - Duplicación
 - *fork()*
- Tuberías o *pipes*

Ejemplo redirección entrada

```
uno,           dos,           tres  
cuatro,        cinco,        seis  
siete,         ocho,         nueve  
diez,          once,         doce
```

f1.txt

```
uno,   dos,   tres  
cuatro, cinco, seis  
siete,  ocho,  nueve  
diez,   once,  doce
```

grep ocho < f1

Ejemplo redirección salida

uno,	dos,	tres
cuatro,	cinco,	seis
siete,	ocho,	nueve
diez,	once,	doce

f1.txt

```
uno, dos, tres  
cuatro, cinco, seis  
siete, ocho, nueve  
diez, once, doce  
siete, ocho, nueve
```

grep ocho < f1 > s1

Ejemplo redirección salida

uno,
cuatro,
siete,
diez,

dos,
cinco,
ocho,
once,

tres
seis
nueve
doce

f1.txt

Dependiente del
intérprete de
mandatos usado

grep ocho f1 1> s1

siete, ocho, nueve

Ejemplo redirección error

uno,
cuatro,
siete,
diez,

dos,
cinco,
ocho,
once,

tres
seis
nueve
doce

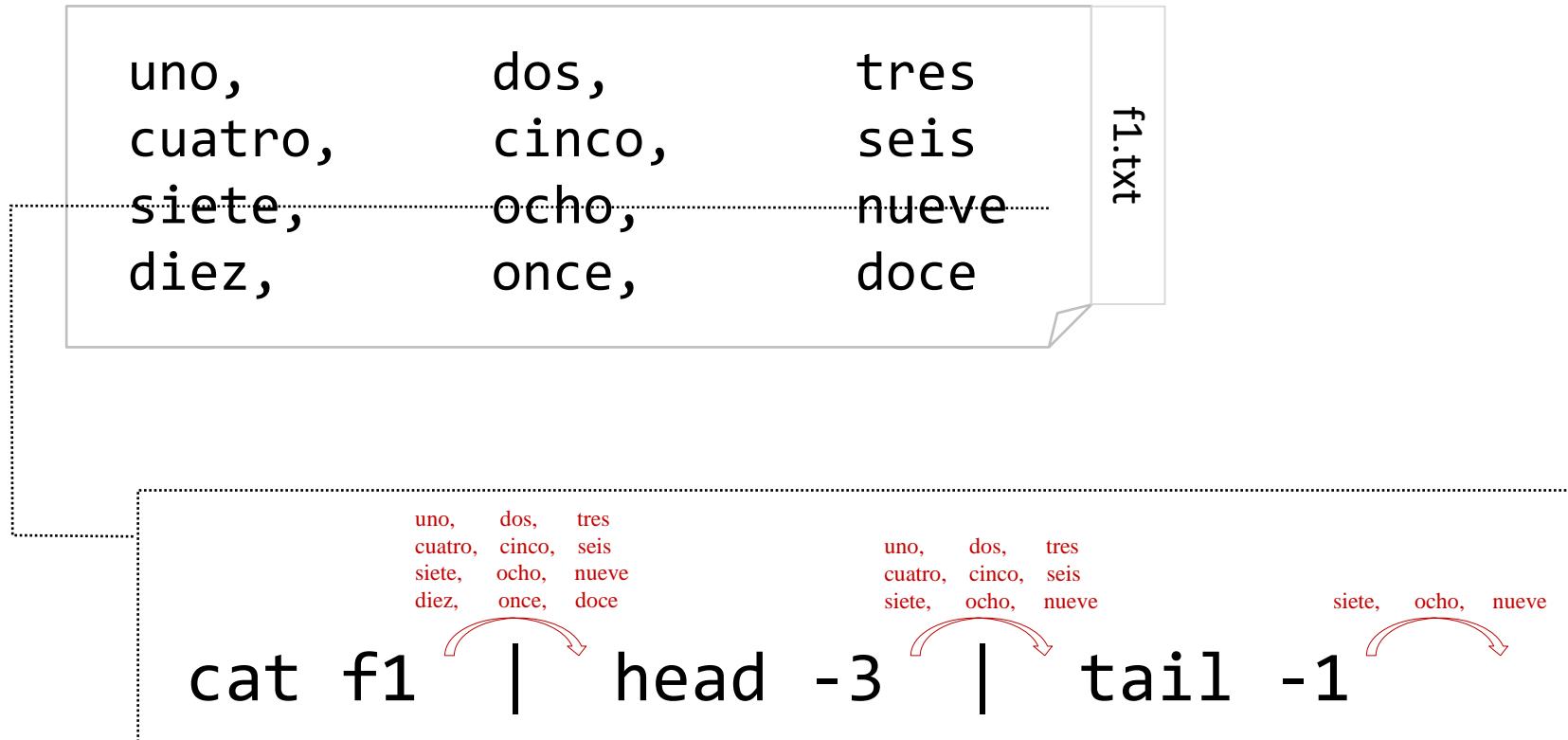
f1.txt

Dependiente del
intérprete de
mandatos usado

grep: xx: No existe el archivo o el directorio

grep ocho xx 2> s1

Ejemplo de uso de tuberías



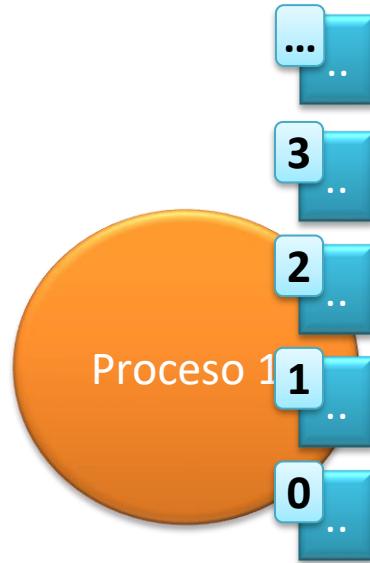
Contenidos

1. Señales y excepciones.
2. Temporizadores.
3. Entorno de un proceso.
4. Comunicación de procesos con tuberías (pipes).
Paso de mensajes local.

- Interprete de mandatos: redirección y tuberías
- **Los descriptores de ficheros**
 - **Modelo: abstracción ofrecida**
 - **Redirección**
 - **Duplicación**
 - ***fork()***
- Tuberías o *pipes*

Descriptores de ficheros

modelo | redirección | duplicación | *fork()*



Los descriptores de ficheros son el índice de la tabla que hay por proceso que identifica los posibles ficheros (o dispositivos) con los que comunicarse

Descriptores de ficheros

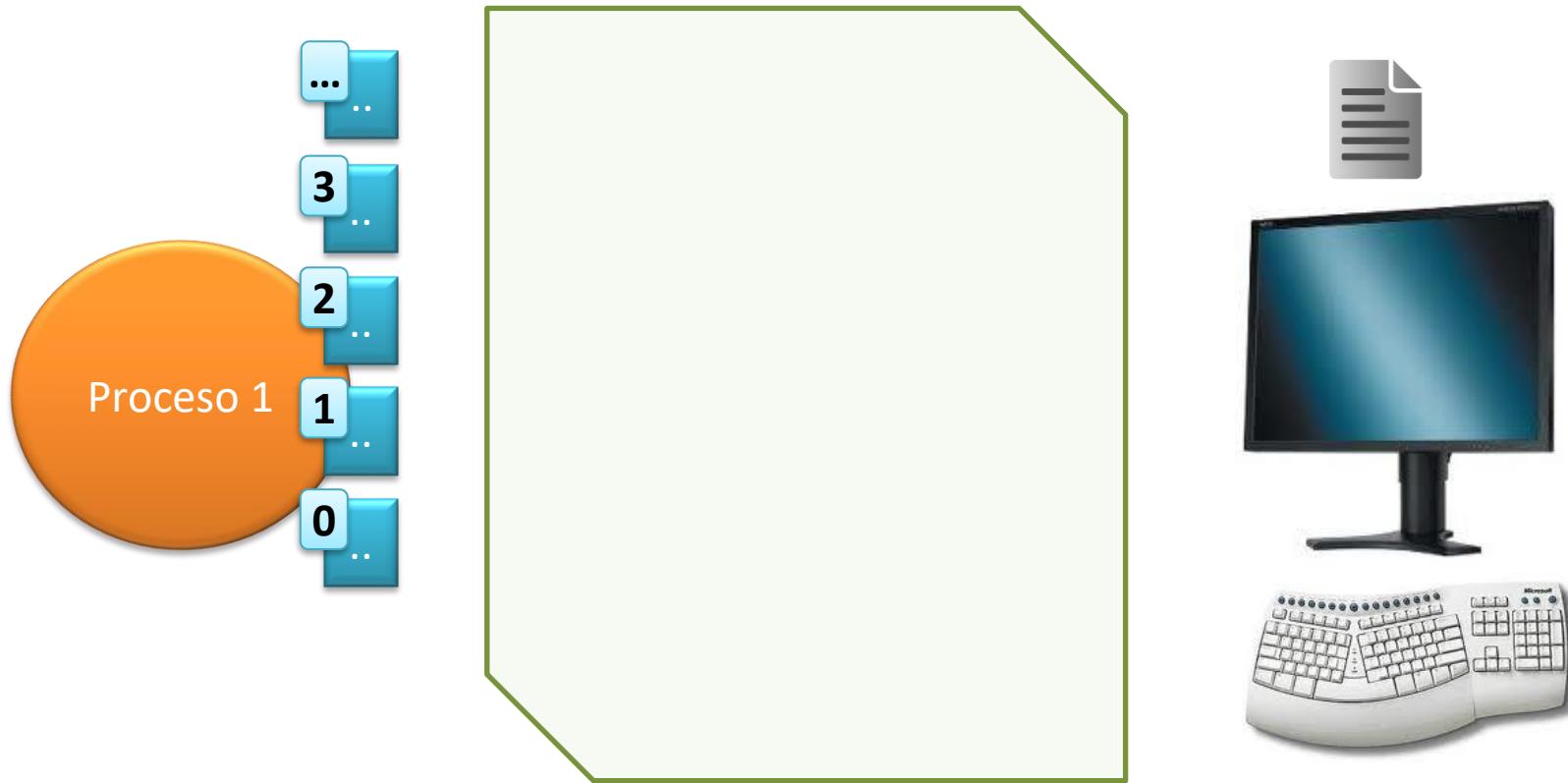
modelo | redirección | duplicación | *fork()*



Por defecto se utilizan los tres primeros para la entrada estándar, salida estándar y salida de error respectivamente.

Descriptores de ficheros

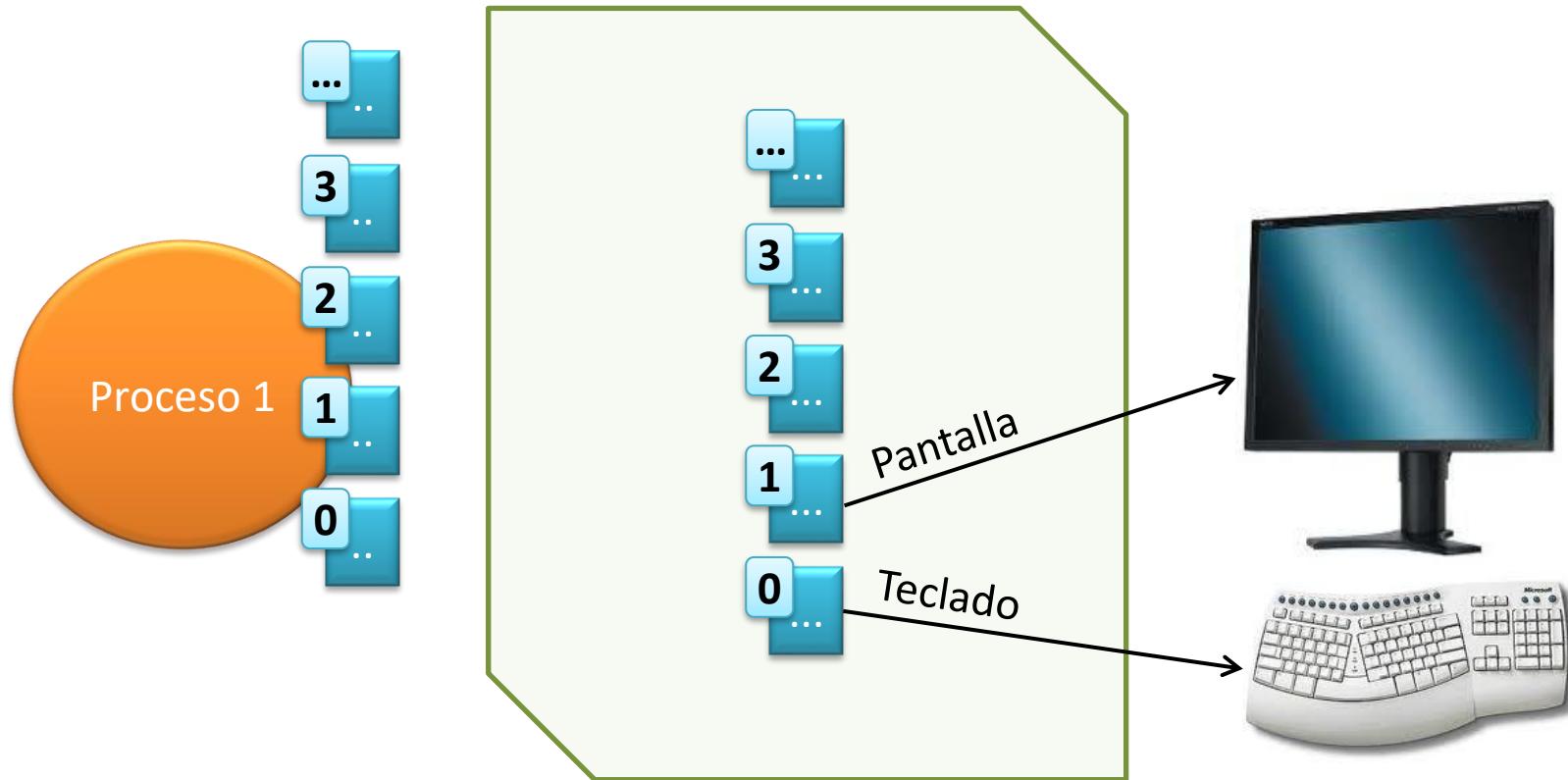
modelo | redirección | duplicación | *fork()*



Los descriptores de ficheros son una abstracción ofrecida por el sistema operativo para referenciar los dispositivos reales. Igual que una llave numerada para una consigna.

Descriptores de ficheros

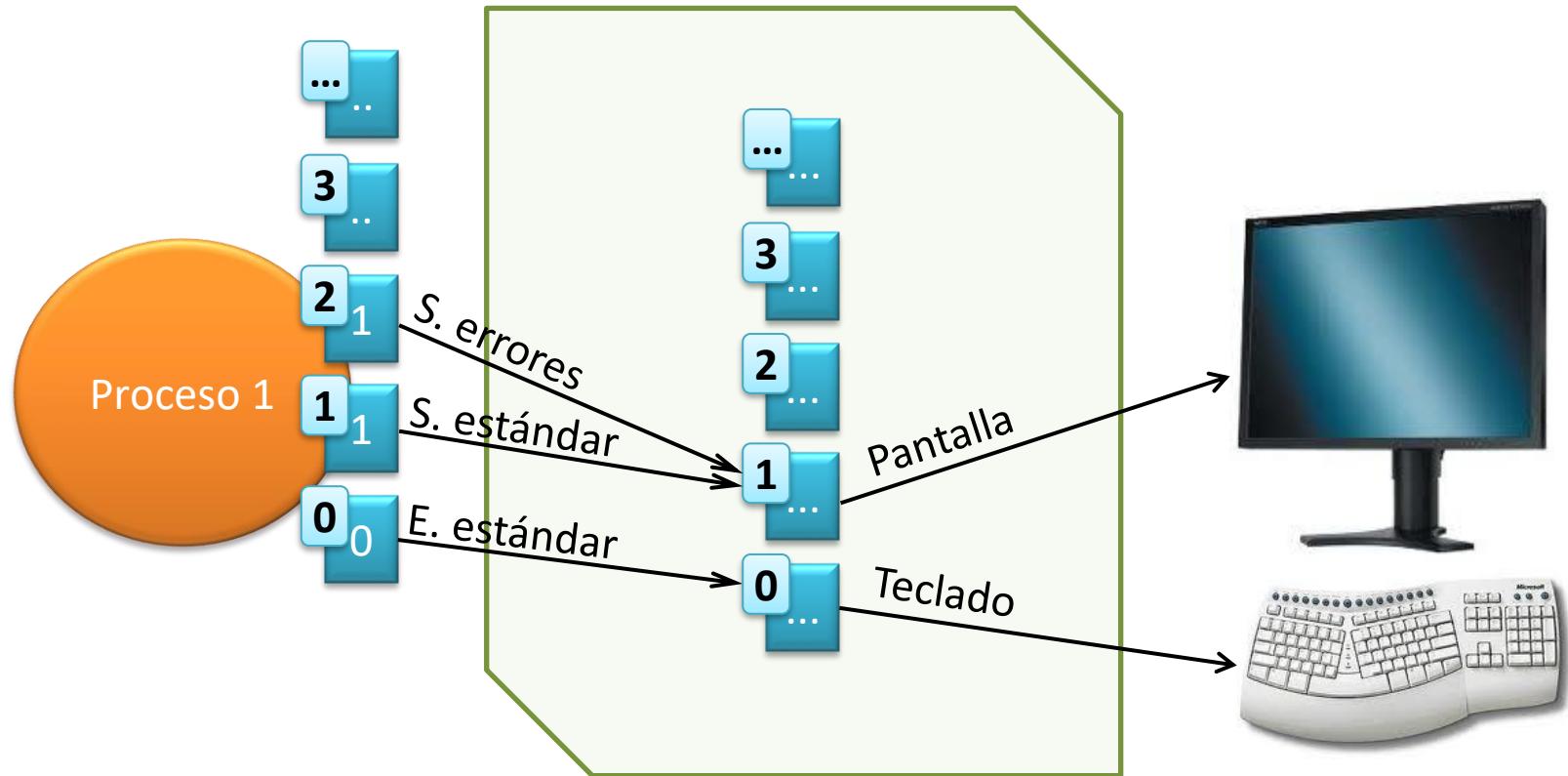
modelo | redirección | duplicación | *fork()*



El sistema operativo mantiene una tabla interna con la información real de contacto con los dispositivos y ficheros con los que los procesos piden comunicarse...

Descriptores de ficheros

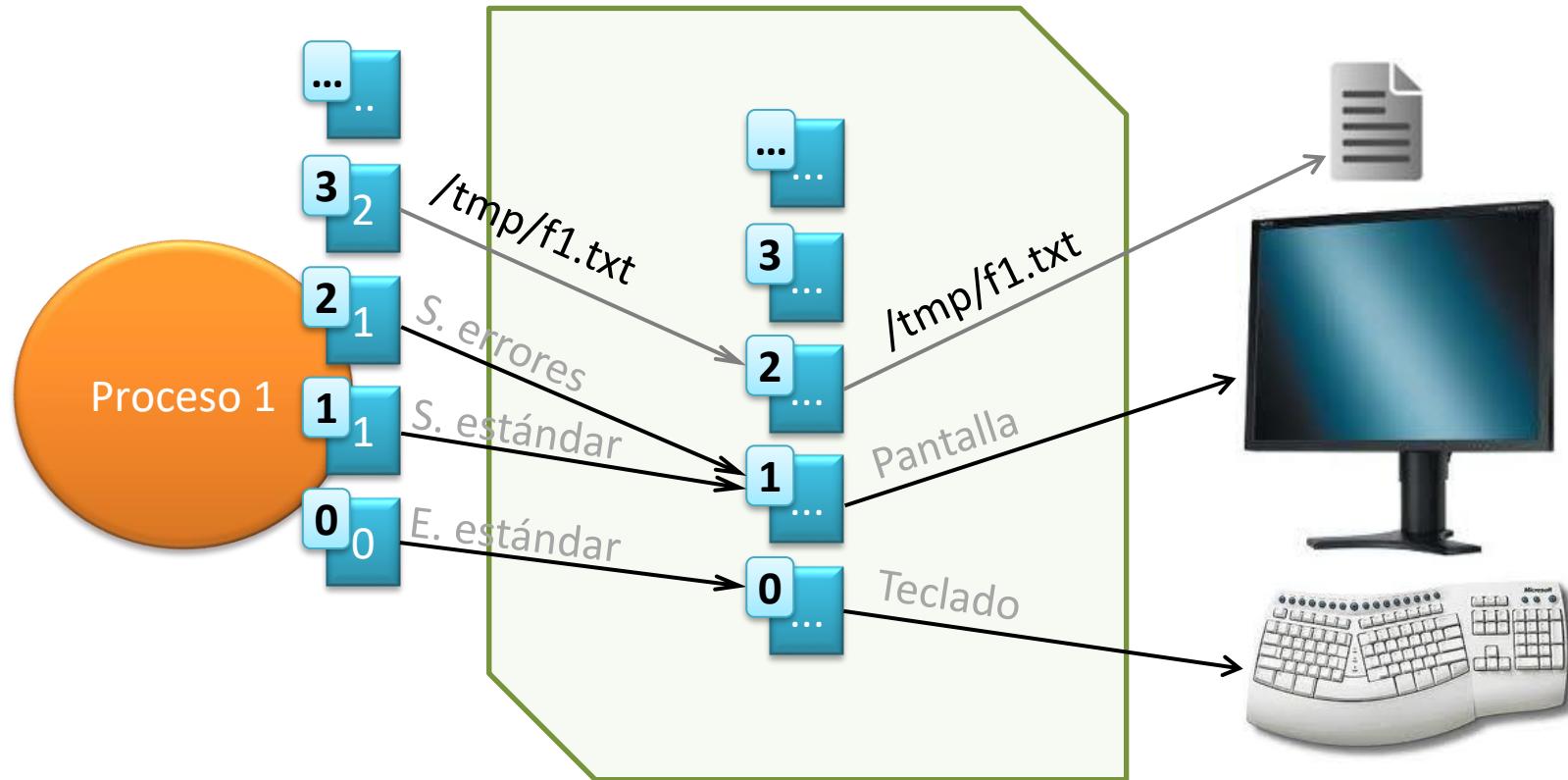
modelo | redirección | duplicación | *fork()*



...Y los descriptores de ficheros son el índice de la tabla que hay por proceso, cuyo contenido es a su vez el índice de la tabla interna del sistema operativo.

Descriptores de ficheros

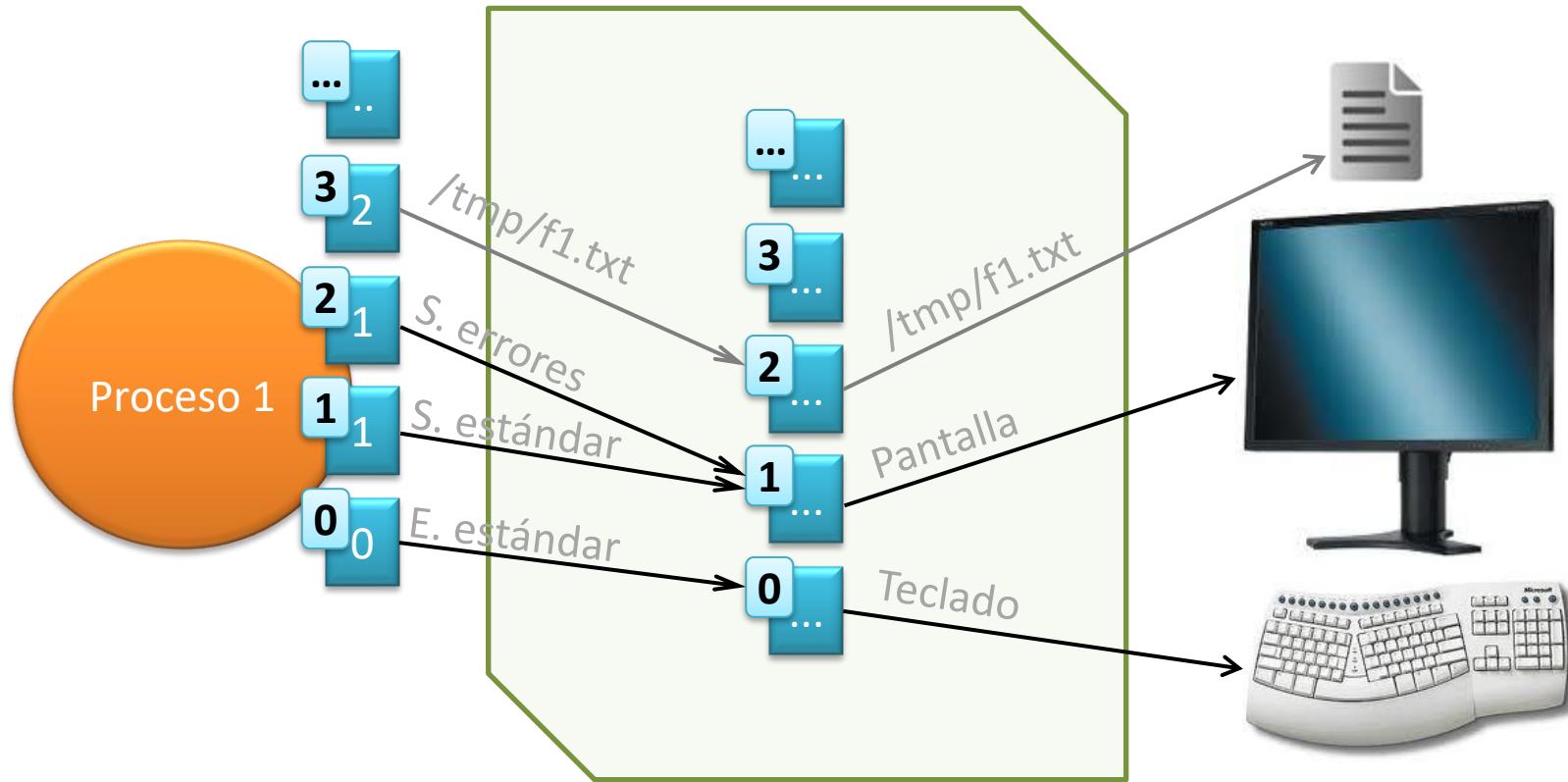
modelo | redirección | duplicación | *fork()*



Cuando se pide un nuevo descriptor de ficheros (al abrir un fichero) se busca el primero hueco libre de la tabla y el índice de esa posición es el descriptor asignado.

Descriptores de ficheros

modelo | **redirección** | duplicación | *fork()*

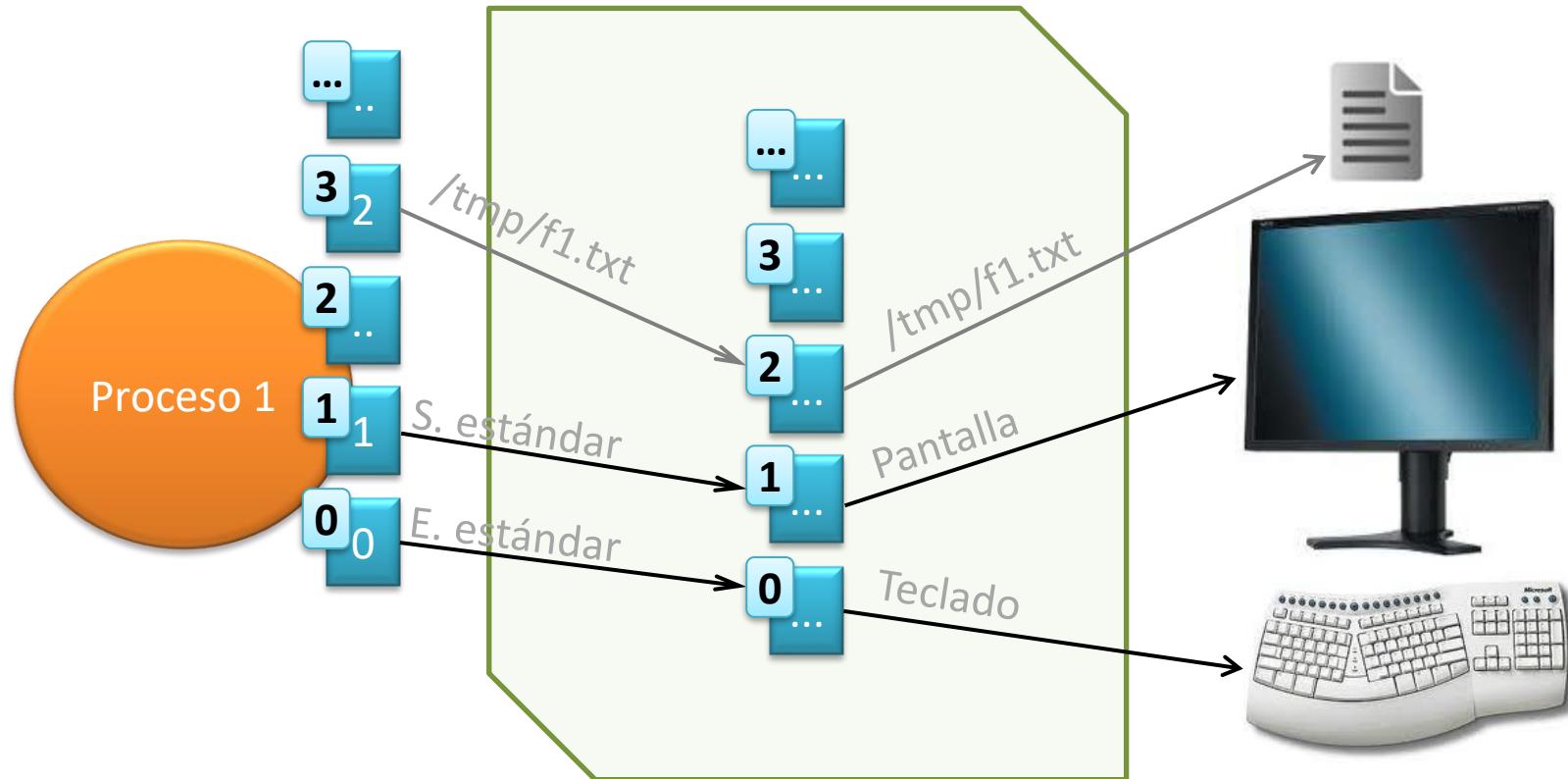


1. `close(2);`
2. `open("/tmp/errores.txt");`

?

Descriptores de ficheros

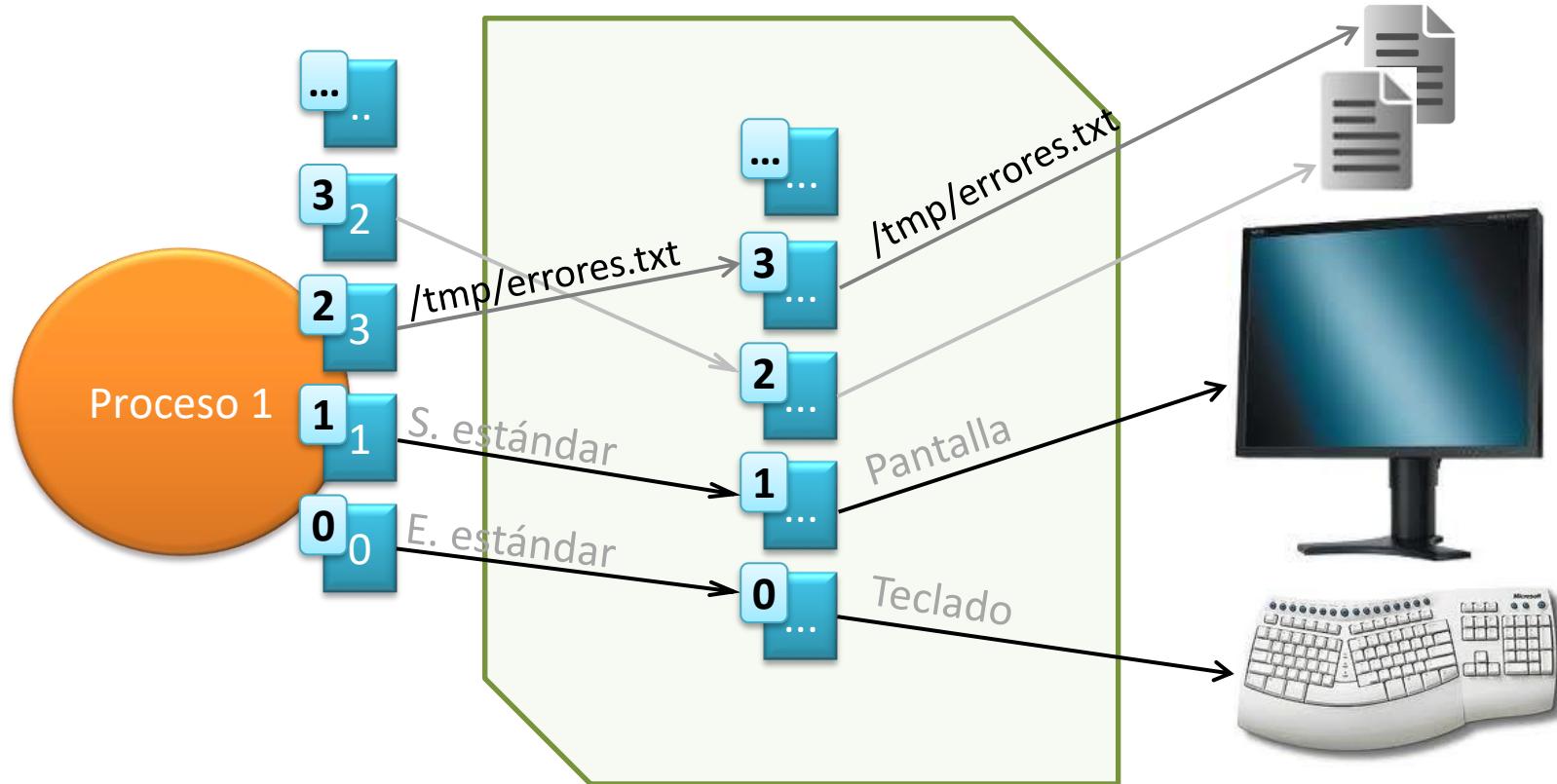
modelo | **redirección** | duplicación | *fork()*



1. **close(2);**
2. **open("/tmp/errores.txt");**

Descriptores de ficheros

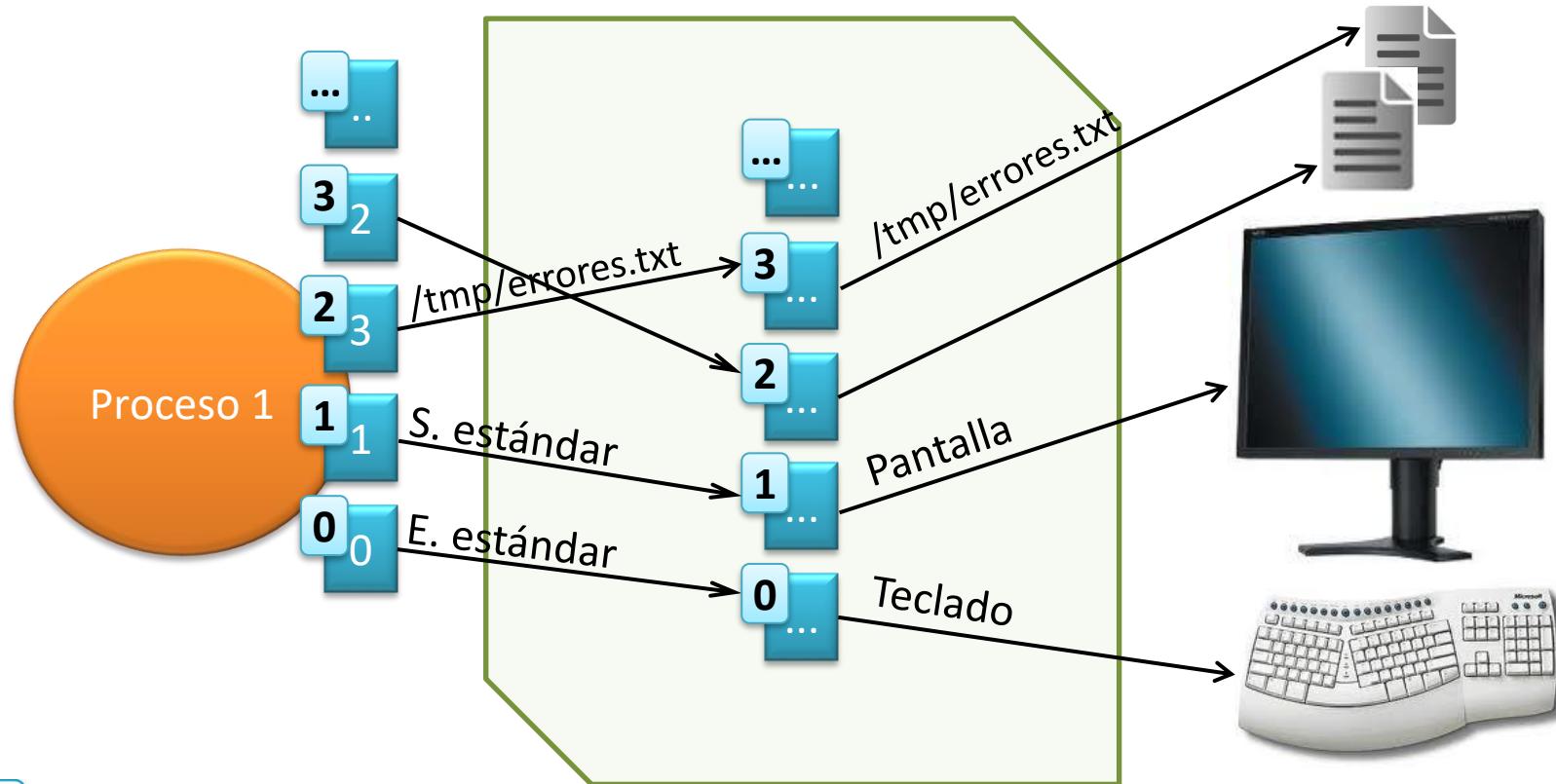
modelo | **redirección** | duplicación | *fork()*



- 1. `close(2);`
- 2. `open("/tmp/errores.txt");`

Descriptores de ficheros

modelo | **redirección** | duplicación | *fork()*

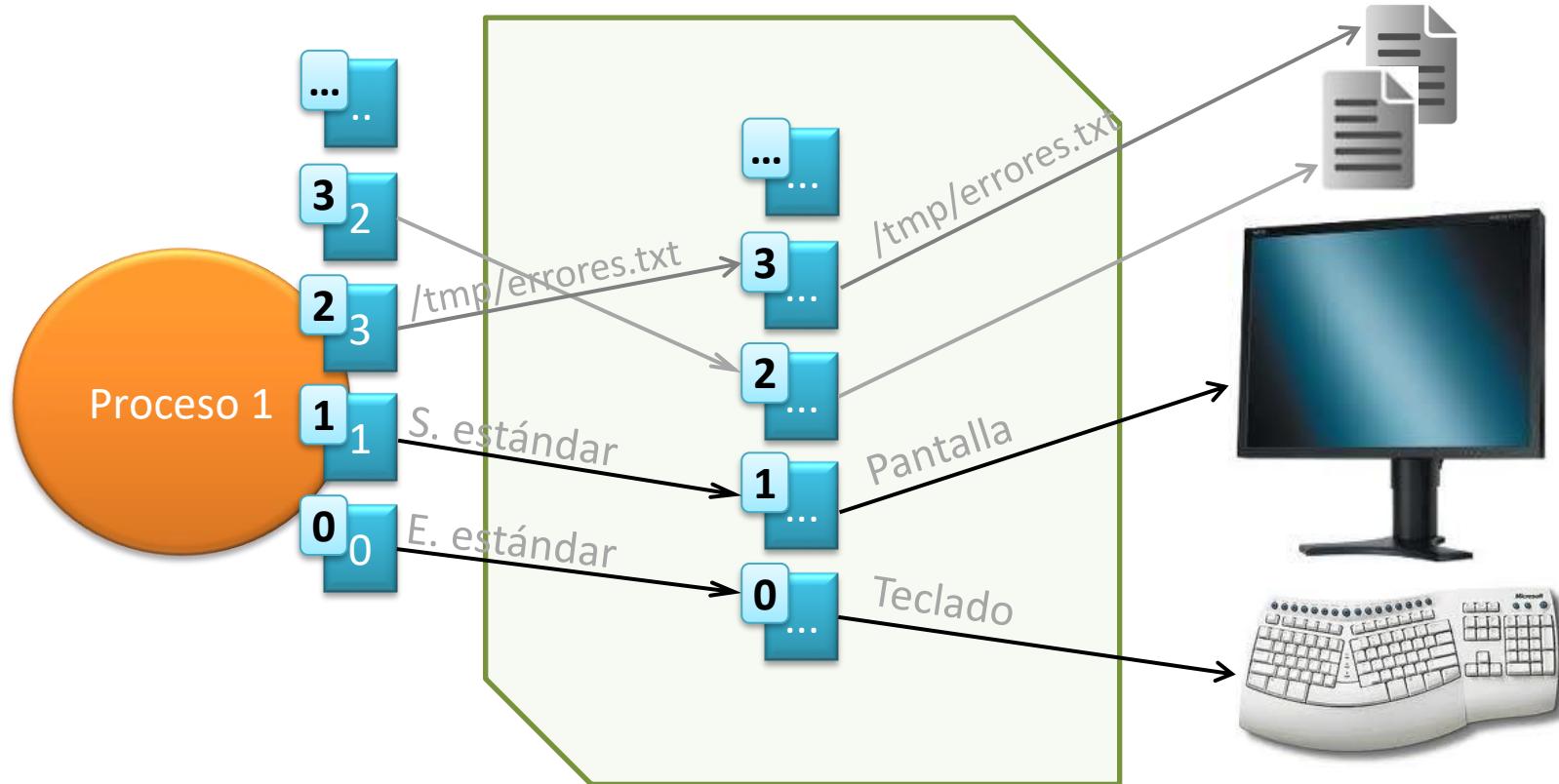


`close(2) + open("/tmp/errores.txt")`

Es posible cambiar el archivo asociado a un descriptor (redirección).

Descriptores de ficheros

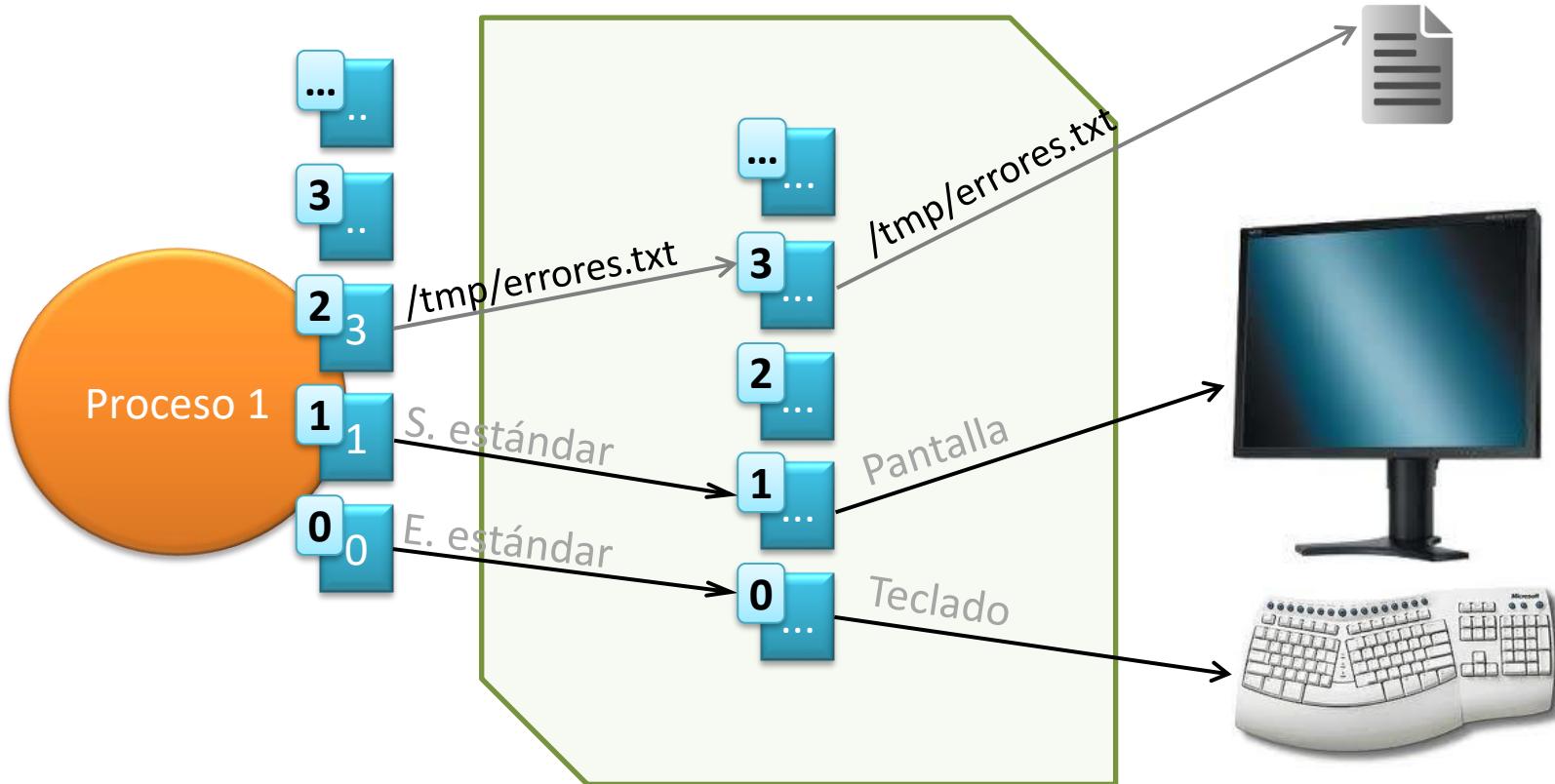
modelo | redirección | **duplicación** | *fork()*



1. `close(3);`
2. `dup(2);` ?

Descriptores de ficheros

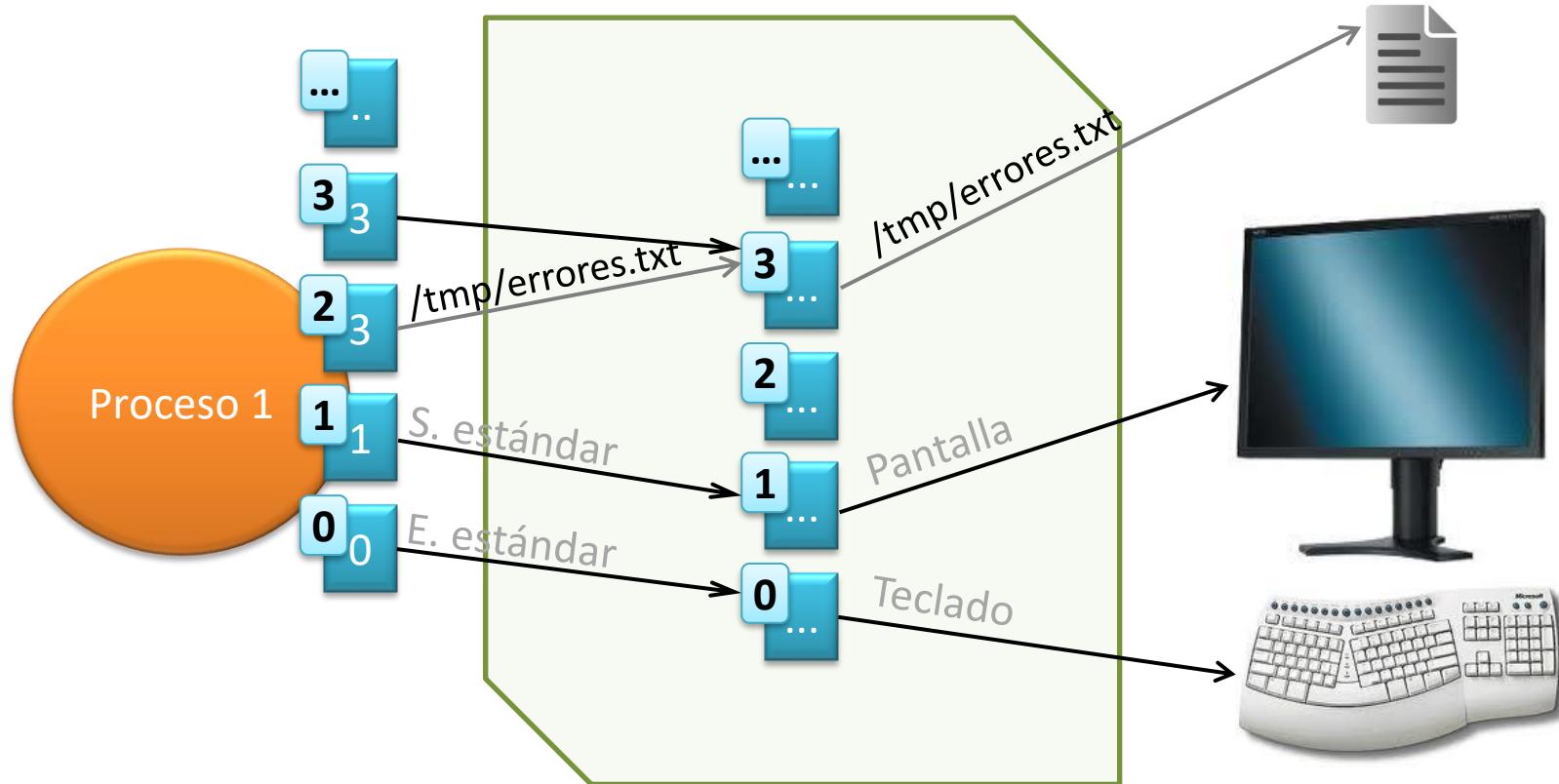
modelo | redirección | **duplicación** | *fork()*



1. **close(3)**;
2. **dup(2)**;

Descriptores de ficheros

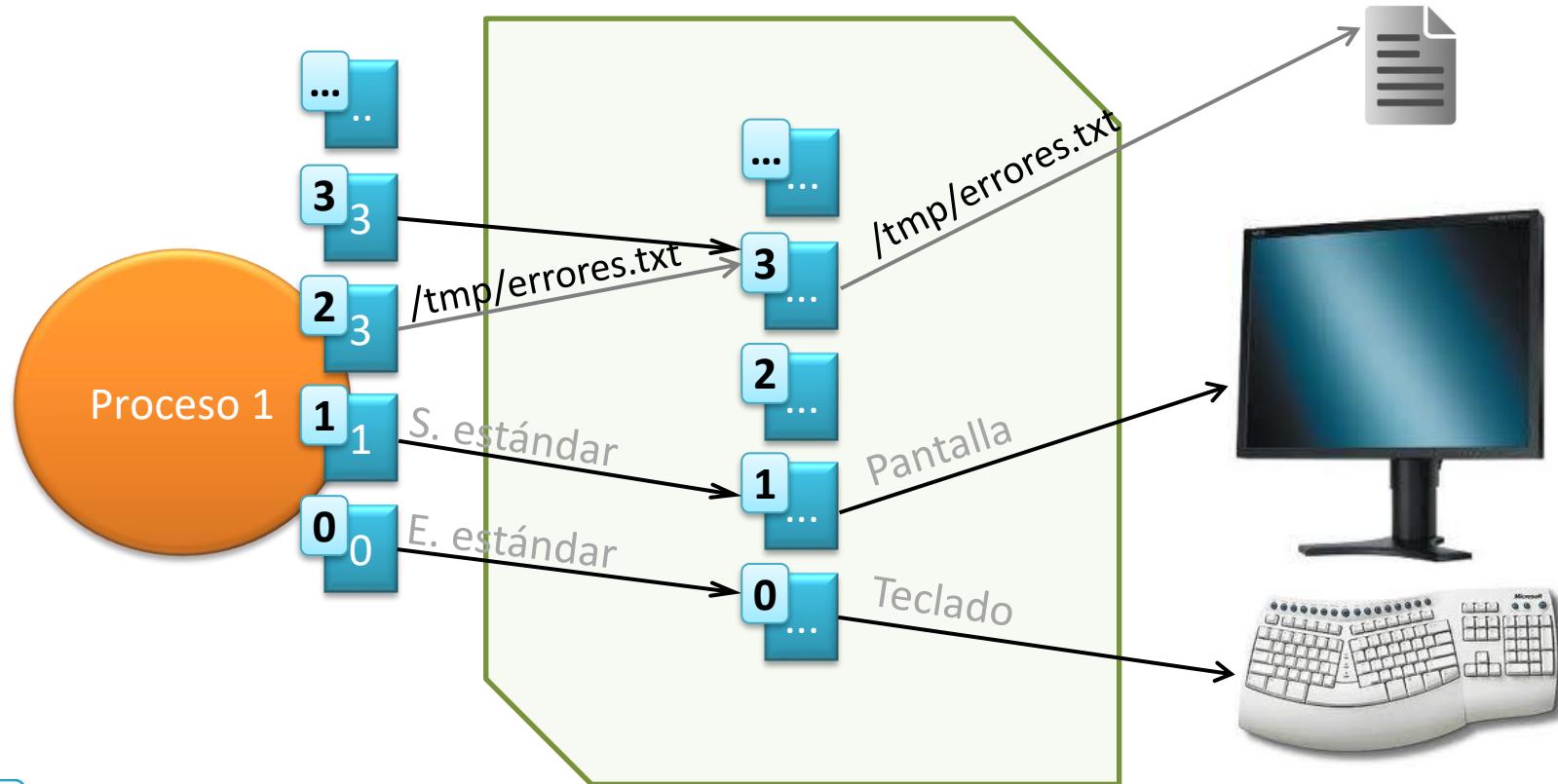
modelo | redirección | **duplicación** | *fork()*



- 1. `close(3);`
- 2. **`dup(2);`**

Descriptores de ficheros

modelo | redirección | **duplicación** | *fork()*

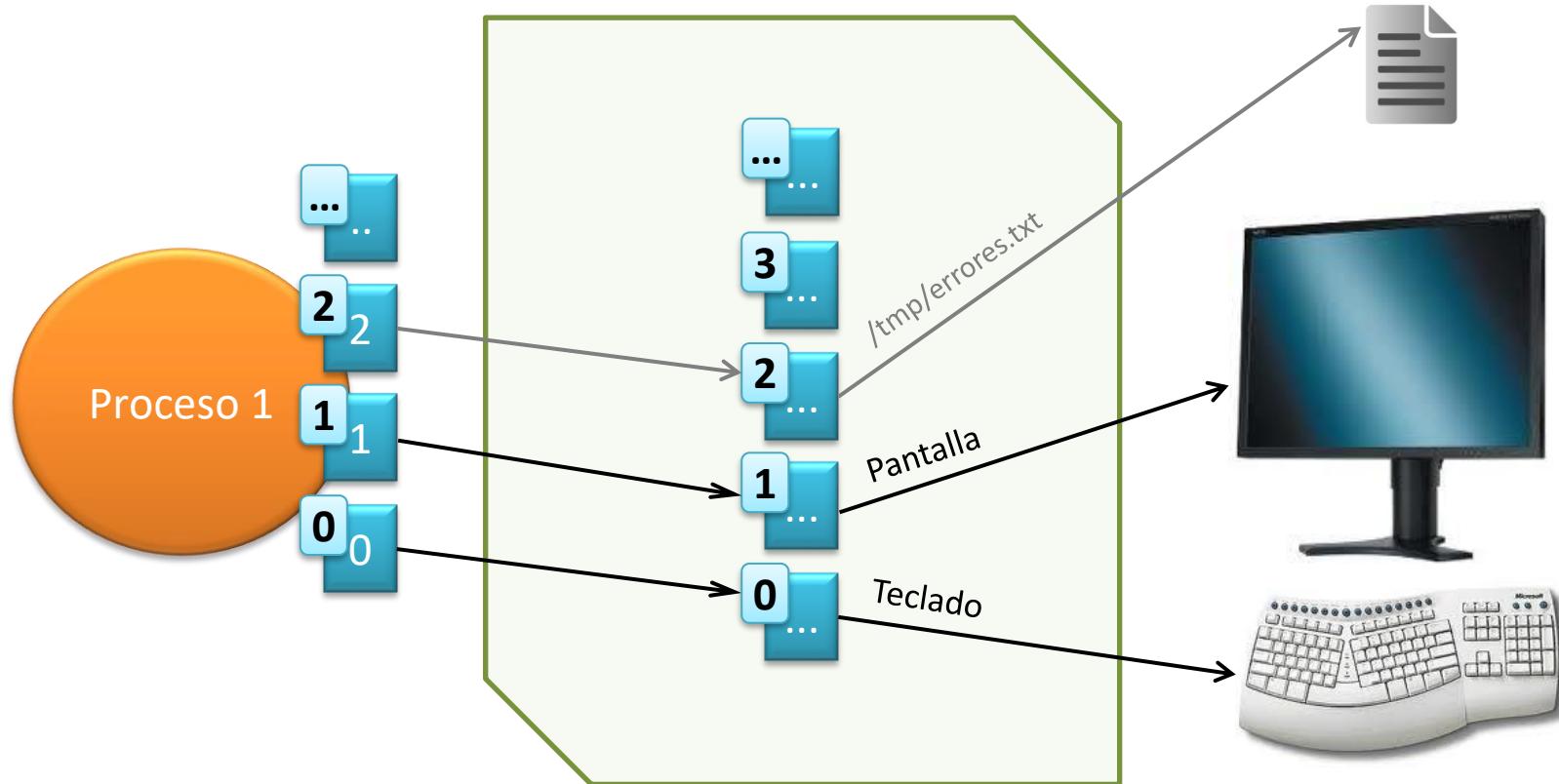


`close(3) + dup(2)`

Permite acceder a un mismo fichero desde dos descriptores diferentes (duplicación).

Descriptores de ficheros

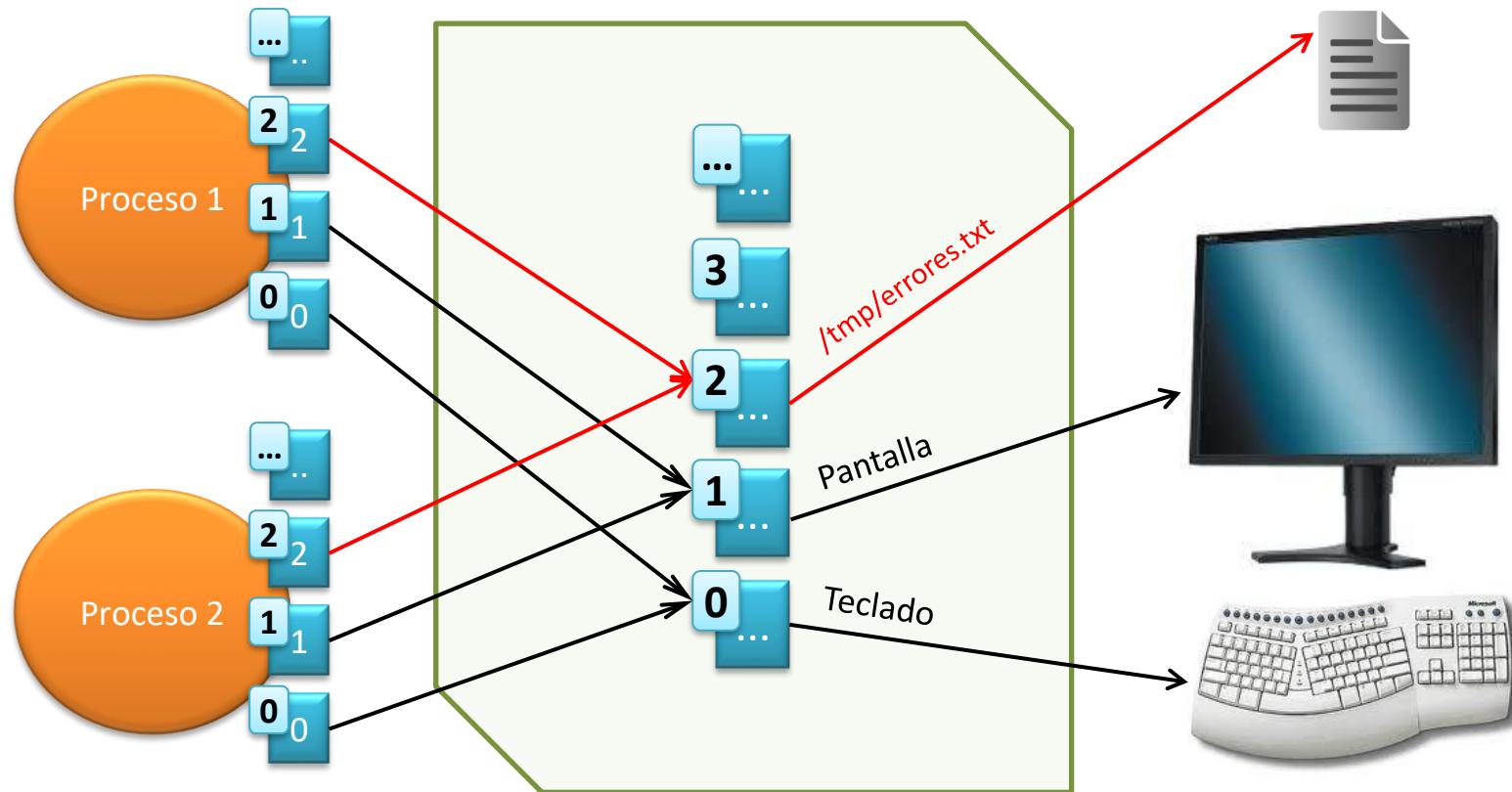
modelo | redirección | duplicación | **fork()**



fork() crea un duplicado del hijo

Descriptores de ficheros

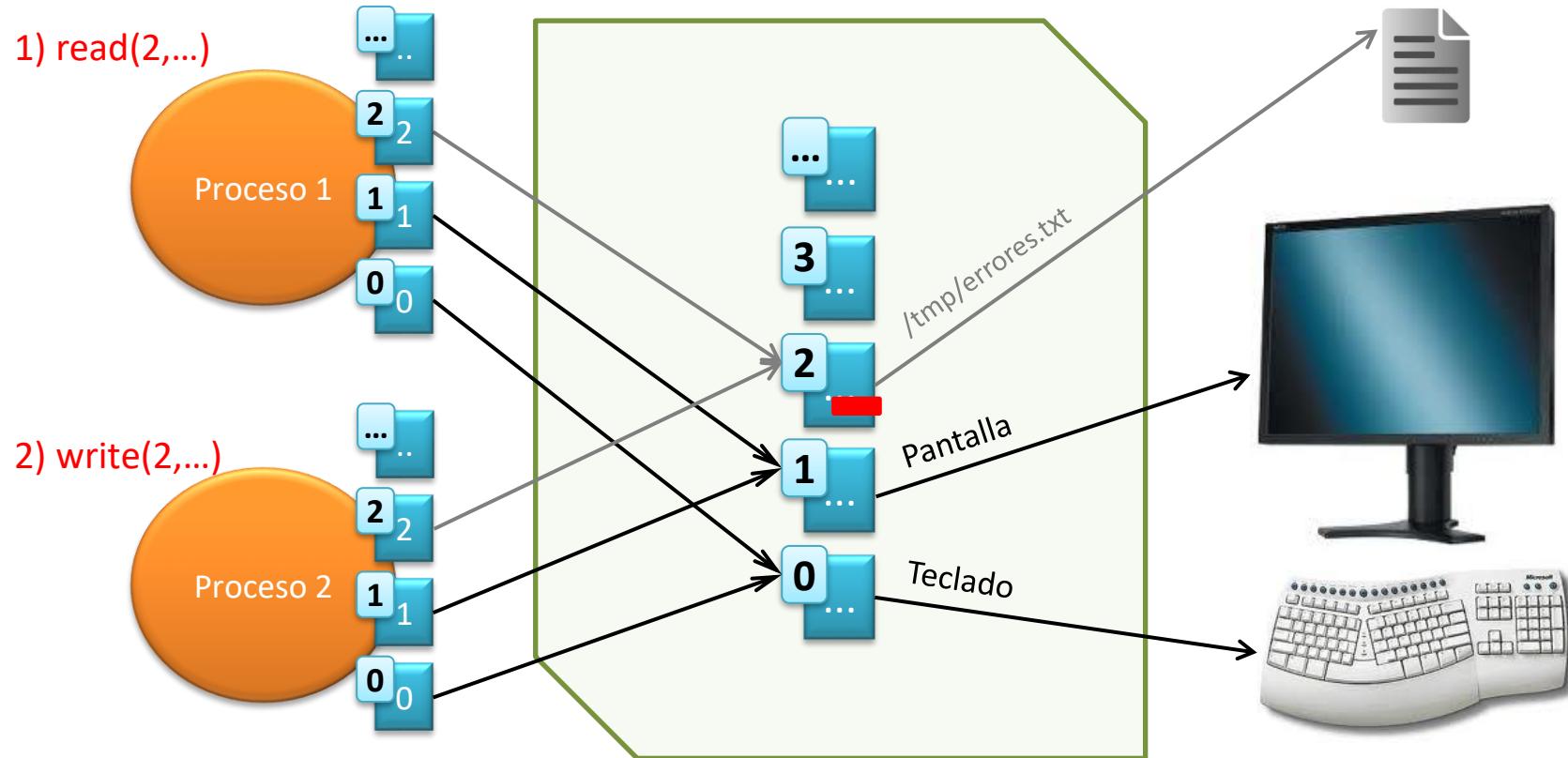
modelo | redirección | duplicación | **fork()**



- Ambos tienen descriptores iguales (redirecciones antes del `fork()` se heredan)
- Ambos referencia los mismos elementos (posición L/E después del `fork()` común)

Descriptores de ficheros

modelo | redirección | duplicación | **fork()**



- Ambos tienen descriptores iguales (redirecciones antes del `fork()` se heredan)
- **Ambos referencia los mismos elementos (posición L/E después del `fork()` común)**

Contenidos

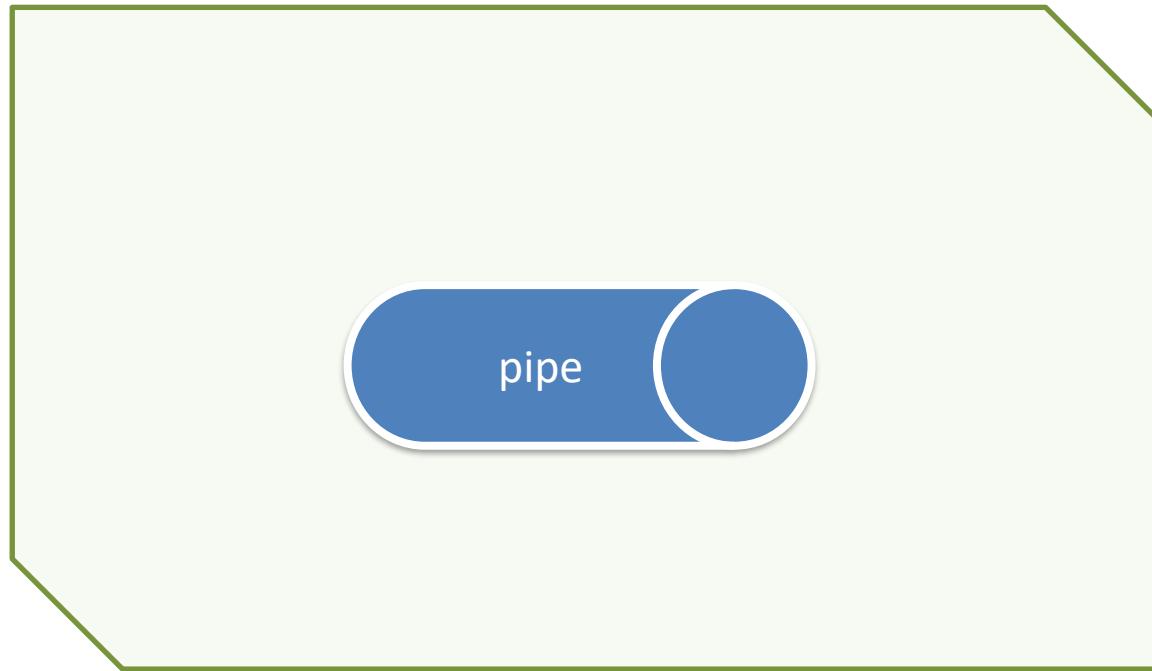
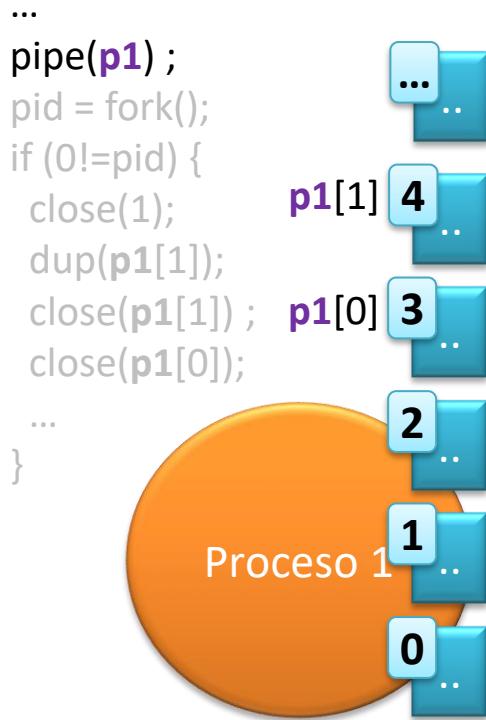
1. Señales y excepciones.
2. Temporizadores.
3. Entorno de un proceso.
4. Comunicación de procesos con tuberías (*pipes*).
Paso de mensajes local.

- Interprete de mandatos: redirección y tuberías
- Los descriptores de ficheros
 - Modelo: abstracción ofrecida
 - Redirección
 - Duplicación
 - *fork()*
- **Tuberías o *pipes***

Tuberías

(1) creación + (2) fork() + (3) redirección + (4) limpieza

```
int p1[2] ;
```



Una tubería es un fichero especial que se crea con la llamada al sistema *pipe()*

Tuberías

(1) creación + (2) fork() + (3) redirección + (4) limpieza

```
int p1[2] ;
```

```
...
```

```
pipe(p1) ;
```

```
pid = fork();  
if (0!=pid) {
```

```
close(1);
```

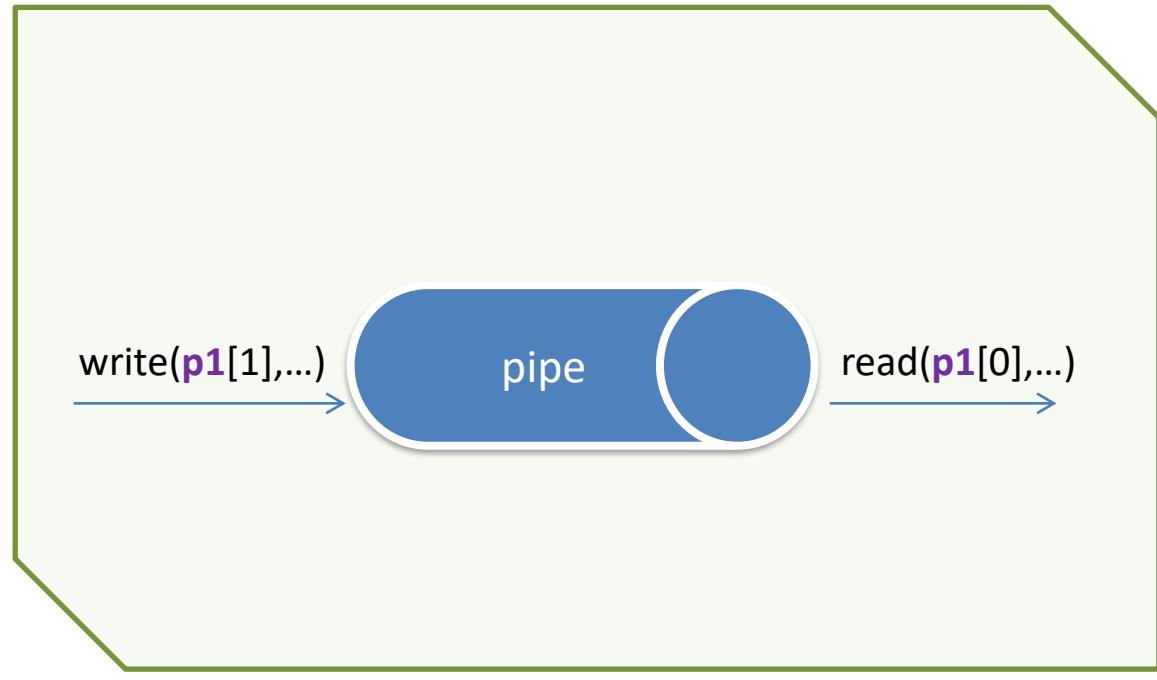
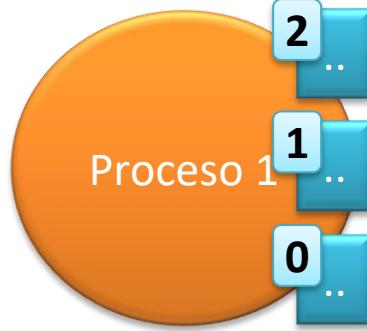
```
dup(p1[1]);
```

```
close(p1[1]);
```

```
close(p1[0]);
```

```
...
```

```
}
```



Una tubería es un fichero especial que se crea con la llamada al sistema *pipe()*
Dicha llamada crea la tubería y reserva dos descriptores de ficheros: lectura y escritura

Tuberías

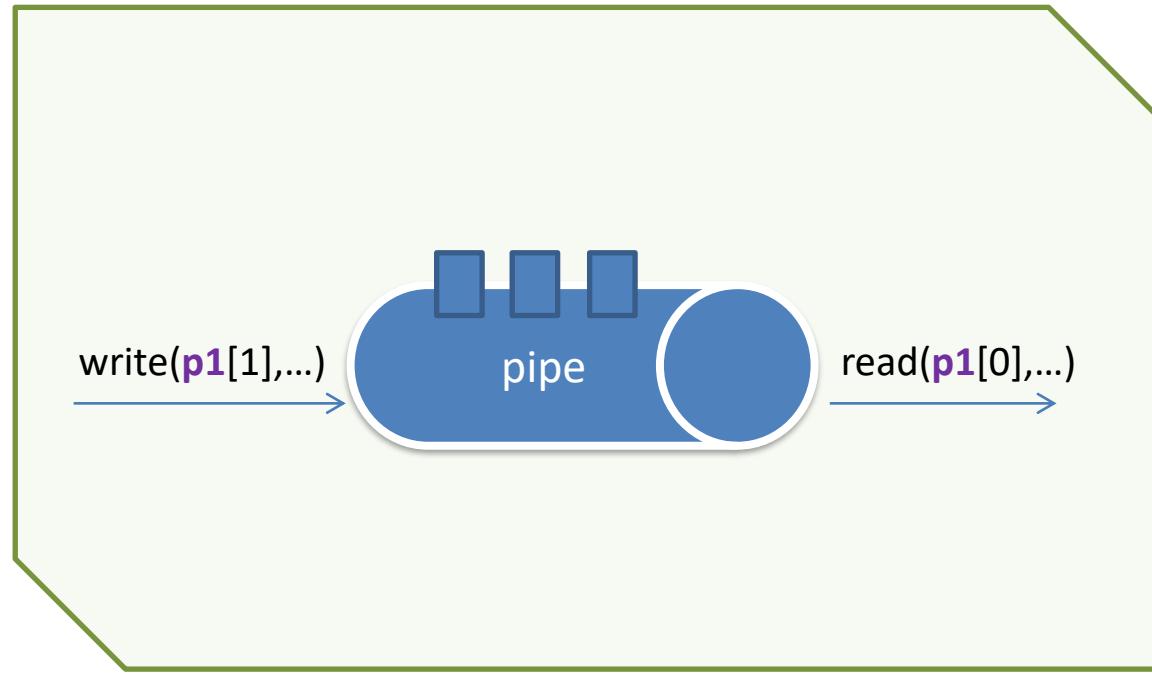
(1) creación + (2) fork() + (3) redirección + (4) limpieza

```
int p1[2] ;
```

```
...
pipe(p1);
pid = fork();
if (0!=pid) {
    close(1);
    dup(p1[1]);
    close(p1[1]);
    close(p1[0]);
}
...
}
```

Diagram illustrating the state of Process 1's file descriptors after the fork operation:

- Original fd 1 (stdout) is closed.
- New fd 4 is created for the pipe's write end.
- New fd 3 is created for the pipe's read end.
- Fds 0 and 2 remain closed.



- Si se escribe en una tubería llena, se bloquea la ejecución del proceso hasta poder escribir.
- Si se lee de una tubería vacía, se bloquea la ejecución del proceso hasta poder leer algo.

Tuberías

(1) creación + (2) fork() + (3) redirección + (4) limpieza

```
int p1[2] ;
```

```
...
```

```
pipe(p1) ;
```

```
pid = fork();  
if (0!=pid) {
```

```
close(1);
```

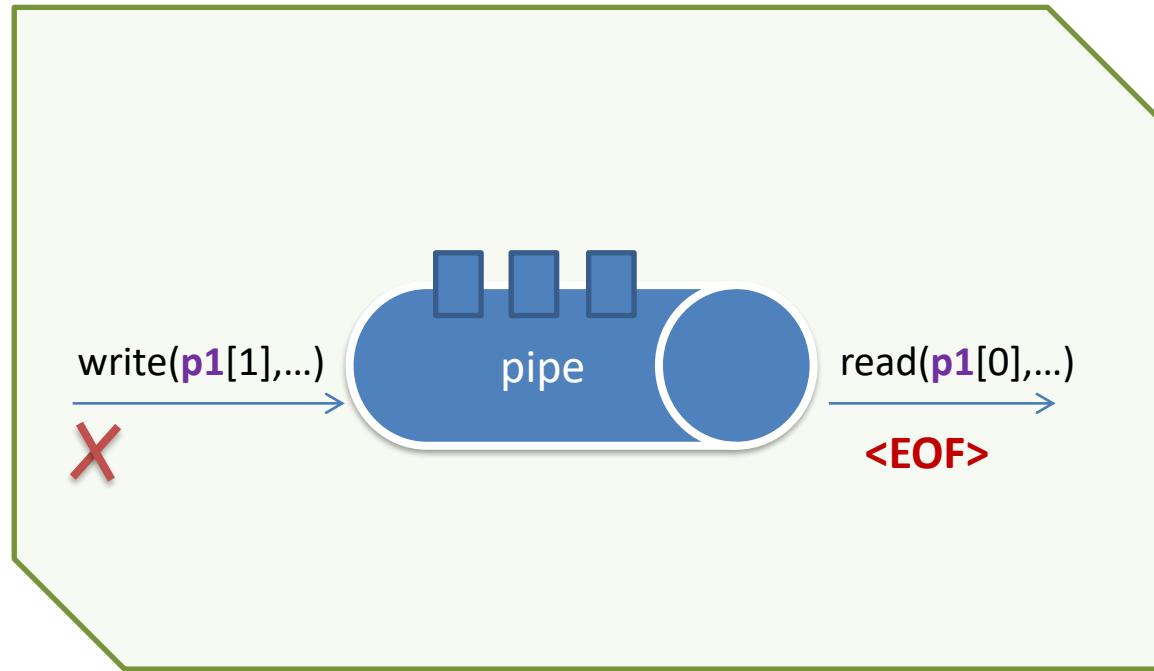
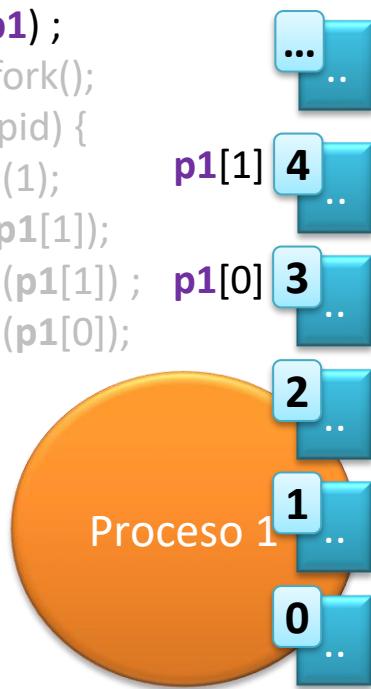
```
dup(p1[1]);
```

```
close(p1[1]);
```

```
close(p1[0]);
```

```
...
```

```
}
```



- Cuando todos los posibles procesos escritores en el pipe cierren la parte de escritura, entonces se manda un final de fichero (EOF) a los lectores.

Tuberías

(1) creación + (2) fork() + (3) redirección + (4) limpieza

```
int p1[2] ;
```

```
...
```

```
pipe(p1) ;
```

```
pid = fork();
```

```
if (0!=pid) {
```

```
close(1);
```



```
dup(p1[1]);
```



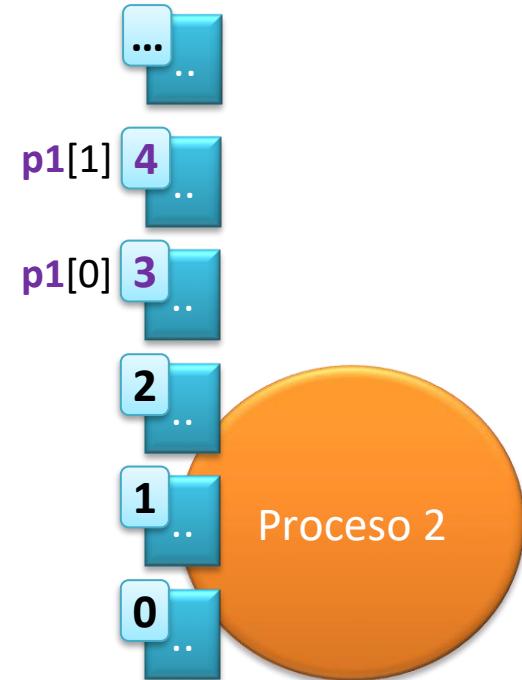
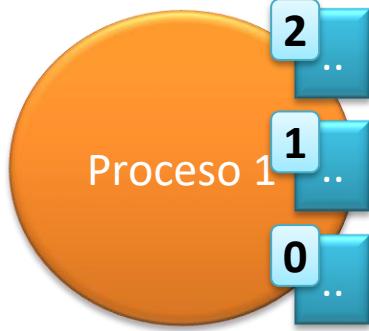
```
close(p1[1]);
```



```
close(p1[0]);
```



```
}
```



pipe() + fork() -> padre e hijo ven la misma tubería

Tuberías

(1) creación + (2) **fork()** + (3) redirección + (4) limpieza

```
int p1[2] ;
```

```
...
```

```
pipe(p1) ;
```

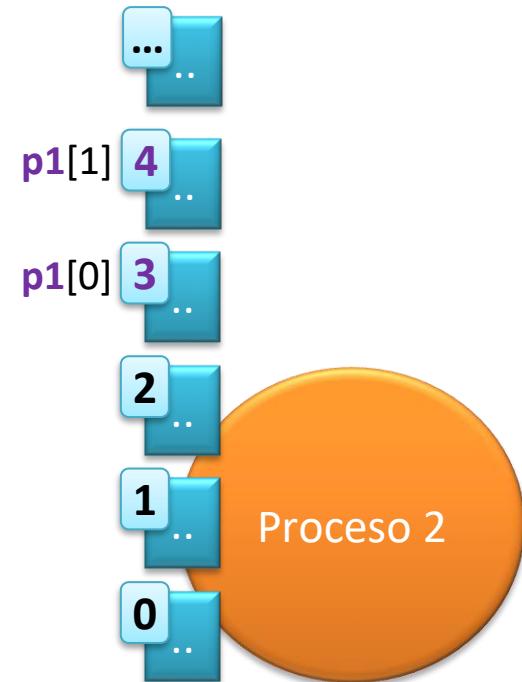
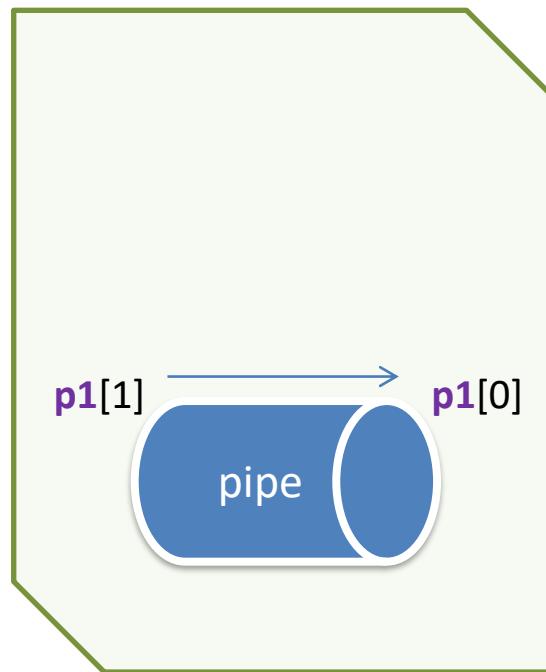
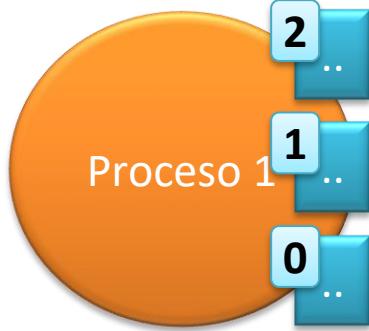
```
pid = fork();
```

```
if (0!=pid) {
```

```
close(1);  
dup(p1[1]);  
close(p1[1]);
```

```
p1[0] ...  
close(p1[0]);
```

```
}
```



pipe() + fork() -> padre e hijo ven la misma tubería
-> ambos podrían leer y escribir en ella

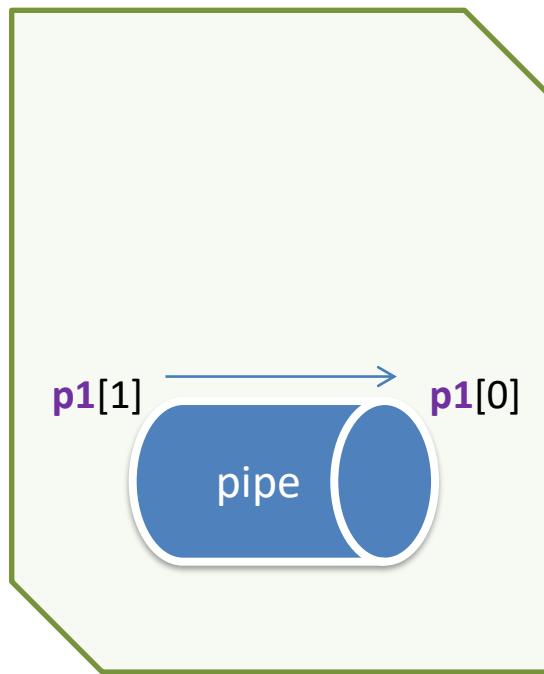
Tuberías

(1) creación + (2) fork() + (3) redirección + (4) limpieza

```
int p1[2] ;
```

```
...
pipe(p1);
pid = fork();
if (0!=pid) {
    close(1);
    dup(p1[1]);
    close(p1[1]);
    p1[0] = ...;
    close(p1[0]);
}
...
}
```

The diagram shows Process 1 as an orange circle. It has five blue rounded rectangles representing file descriptors. From bottom to top, they are labeled 0, 1, 2, 3, and 4. Descriptions next to them are: 0 .., 1 .., 2 .., 3 .., and 4 .. respectively. A red 'X' is placed to the right of the 1, 2, 3, and 4 boxes, indicating they are closed. Only descriptor 0 is still open. An arrow points from descriptor 1 to a blue pipe component.



```
else {
    close(0);
    dup(p1[0]);
    close(p1[0]);
    close(p1[1]);
    ...
}
```

The diagram shows Process 2 as an orange circle. It has five blue rounded rectangles representing file descriptors. From bottom to top, they are labeled 0, 1, 2, 3, and 4. Descriptions next to them are: 0 .., 1 .., 2 .., 3 .., and 4 .. respectively. A red 'X' is placed to the right of the 0, 1, 2, and 3 boxes, indicating they are closed. Only descriptor 4 is still open. An arrow points from descriptor 0 to the pipe component.

Redirección de la salida estándar en el padre...

Redirección de la entrada estándar en el hijo...

Tuberías

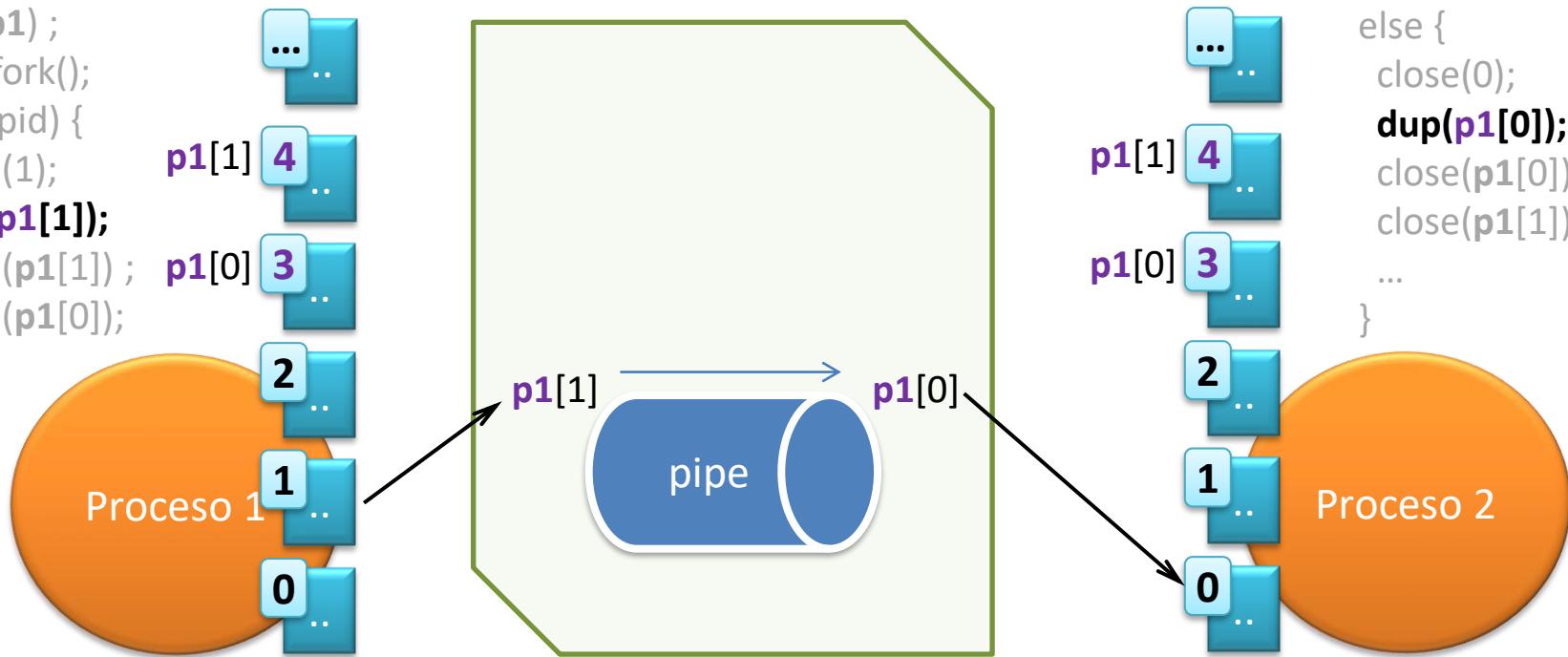
(1) creación + (2) fork() + (3) redirección + (4) limpieza

```
int p1[2] ;
```

```
...
```

```
pipe(p1) ;  
pid = fork();  
if (0!=pid) {  
    close(1);  
    dup(p1[1]);  
    close(p1[1]);  
    close(p1[0]);  
}
```

```
...  
}
```



Redirección de la salida estándar en el padre...

Redirección de la entrada estándar en el hijo...

Tuberías

(1) creación + (2) fork() + (3) redirección + (4) limpieza

```
int p1[2] ;
```

```
...
```

```
pipe(p1) ;
```

```
pid = fork();
```

```
if (0!=pid) {
```

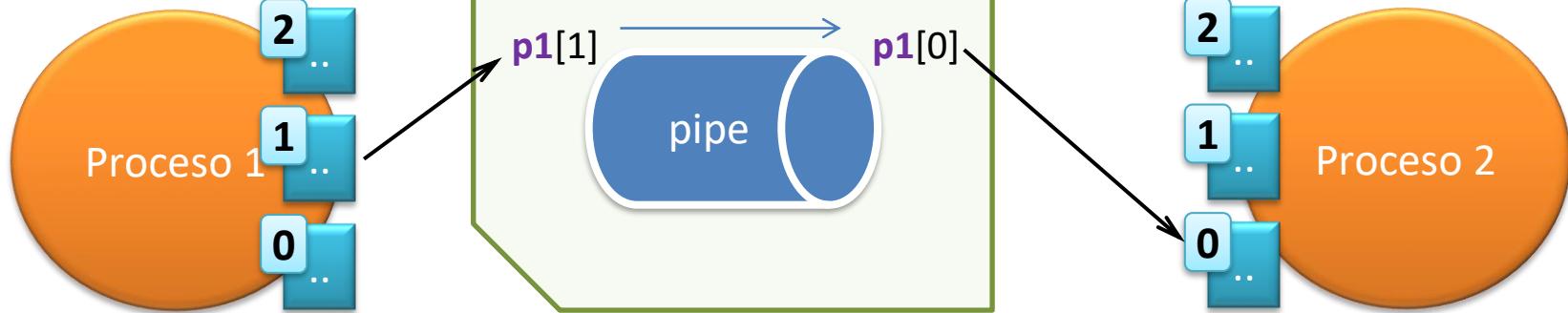
```
close(1);
```

```
dup(p1[1]);
```

```
close(p1[1]);
```

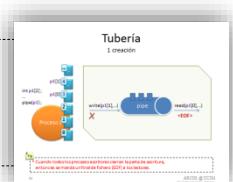
```
close(p1[0]);
```

```
}
```



Cierre de los descriptores que no se usan en el padre...

Cierre de los descriptores que no se usan en el hijo...



Descriptores de ficheros

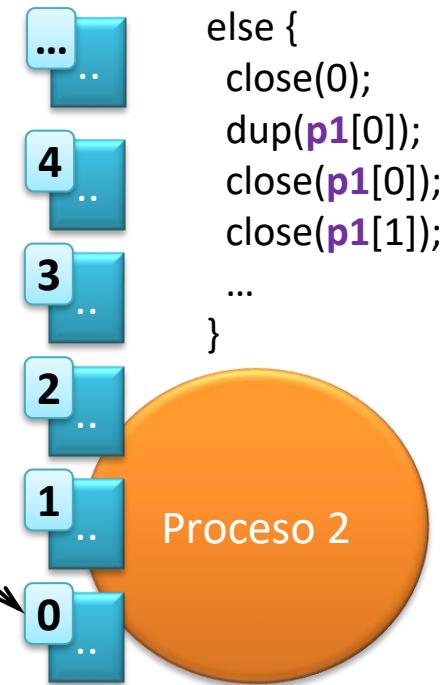
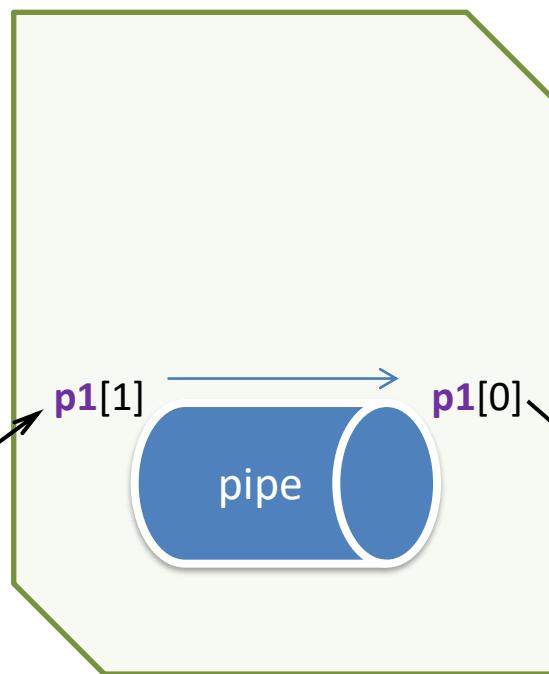
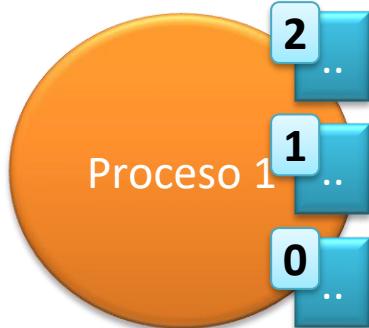
resumen

```
int p1[2] ;
```

```
...
```

```
pipe(p1) ;  
pid = fork();  
if (0!=pid) {  
    close(1);  
    dup(p1[1]);  
    close(p1[1]) ;  
    close(p1[0]);  
}
```

```
}
```



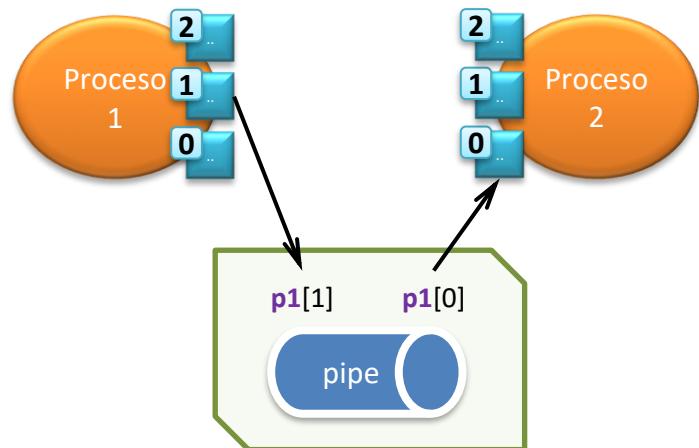
Descriptoros de ficheros

resumen

```
int p1[2] ;  
...  
pipe(p1) ;  
pid = fork();  
if (0!=pid) {  
    close(1);  
    dup(p1[1]);  
    close(p1[1]) ;  
    close(p1[0]);  
    ...  
} else {  
    close(0);  
    dup(p1[0]);  
    close(p1[0]);  
    close(p1[1]);  
    ...  
}  
}
```

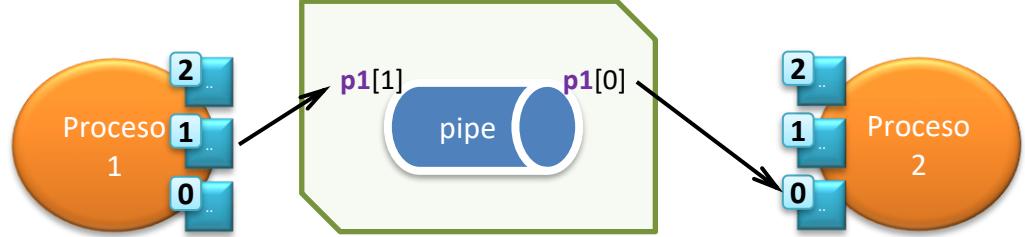
1) Creación
2) *fork()*
3) Redirección (padre)
4) Limpieza (padre)

3) Redirección (hijo)
4) Limpieza (hijo)



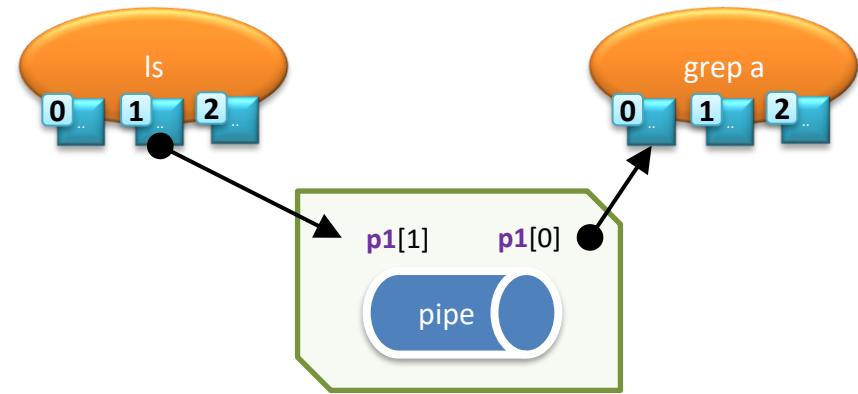
Tuberías

limitaciones



- **Semi-duplex:**
 - En un sentido: los datos son escritos por un proceso en un extremo de la tubería y leídos por otro proceso desde el otro extremo del mismo.
- Solo se pueden utilizar entre **procesos emparentados**, que tengan un ancestro en común.
- La **lectura es destructiva**.

Ejemplo 1: "ls | grep a"

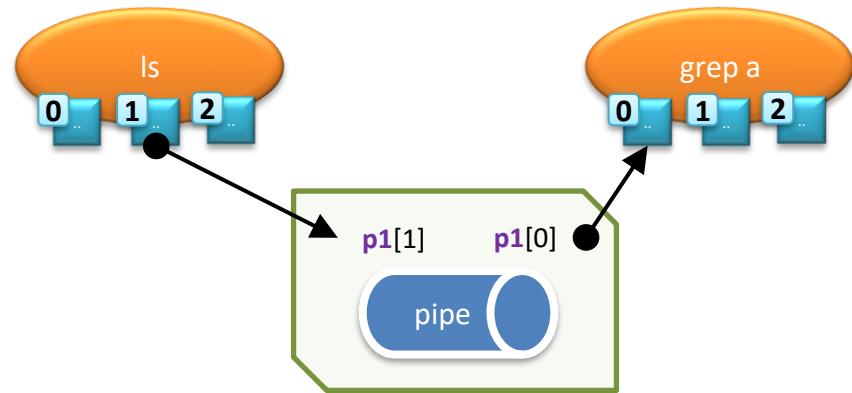


Ejemplo 1: "ls | grep a"

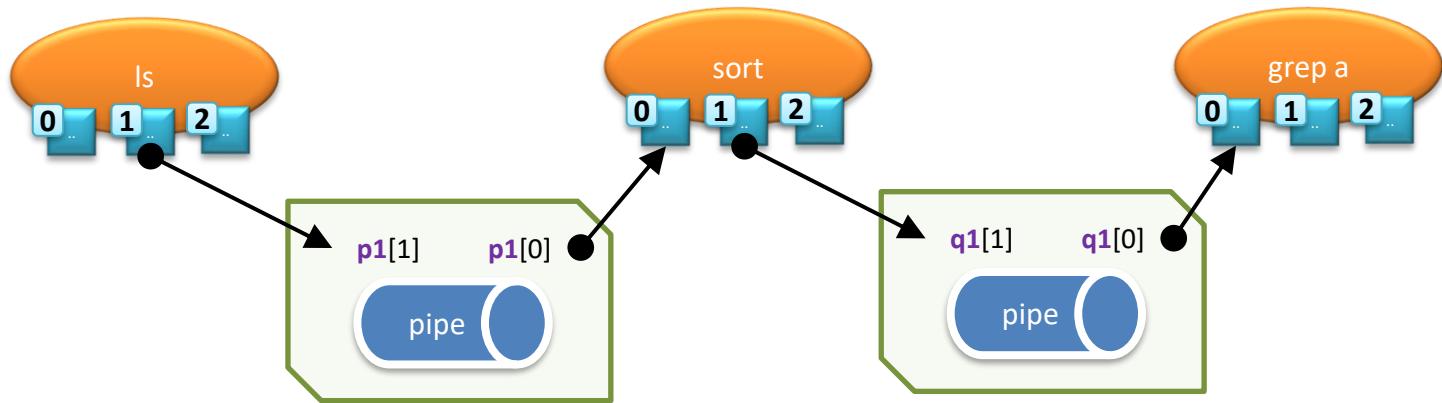
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main (int argc, char *argv[])
{
    int fd[2];

    pipe(fd);
    if (fork() !=0) { /* código del padre */
        close(STDIN_FILENO);
        dup(fd[STDIN_FILENO]);
        close(fd[STDIN_FILENO]);
        close(fd[STDOUT_FILENO]);
        execvp("grep", "grep", "a", NULL);
    } else { /* código del hijo */
        close(STDOUT_FILENO);
        dup(fd[STDOUT_FILENO]);
        close(fd[STDOUT_FILENO]);
        close(fd[STDIN_FILENO]);
        execvp("ls", "ls", NULL);
    }
    return 0;
}
```

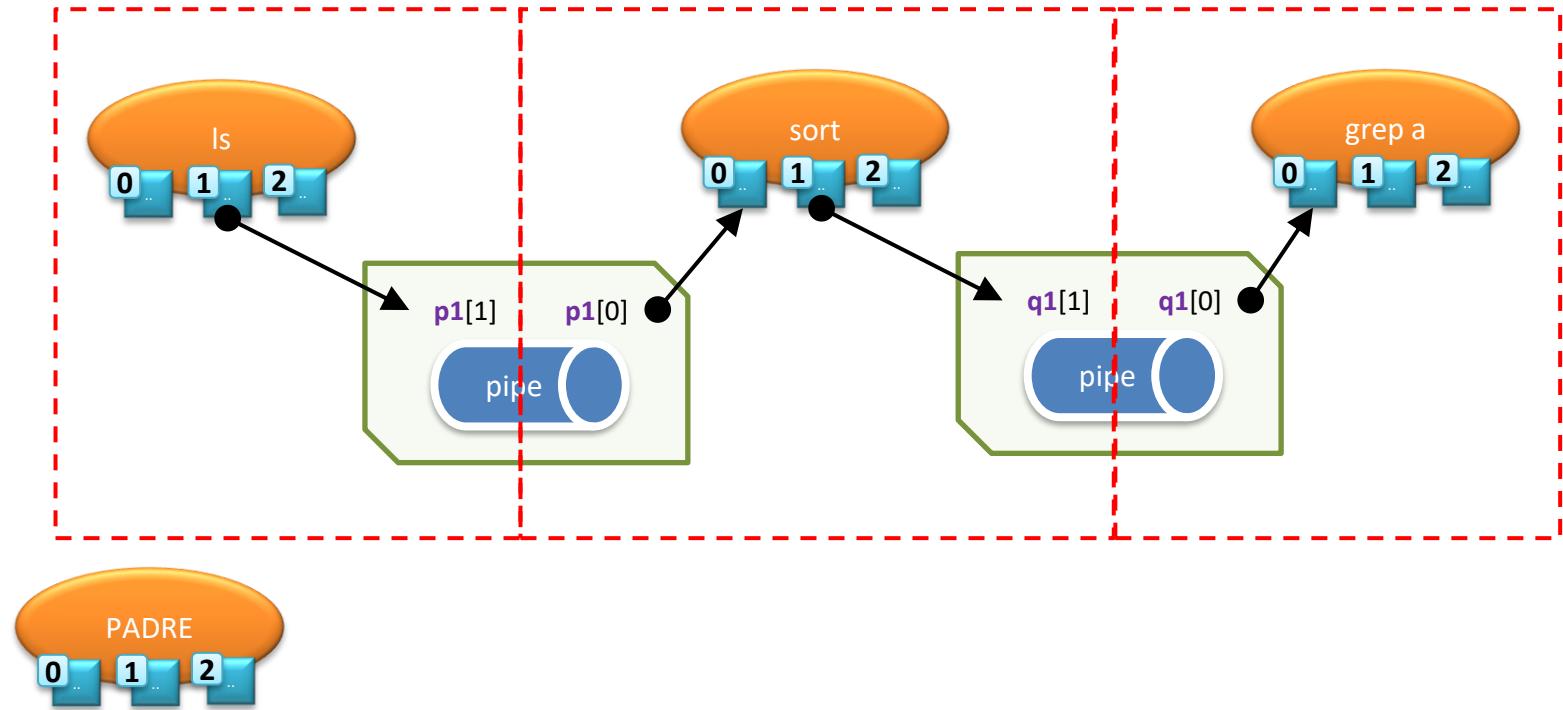


Ejemplo 2: "ls | sort | grep a"



Es posible trabajar con varias tuberías (*pipes*) de forma similar.
Se trabaja con un proceso “PADRE” que se encarga de ir creando la estructura.

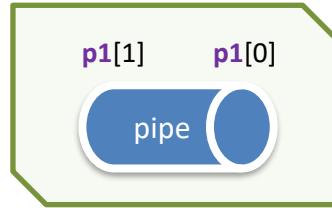
Ejemplo 2: "ls | sort | grep a"



La idea es aplicar un “divide y vencerás” trabajando cada tuberías y sus procesos asociados.
Tres tipos de casos: primer proceso, procesos intermedios y último proceso.

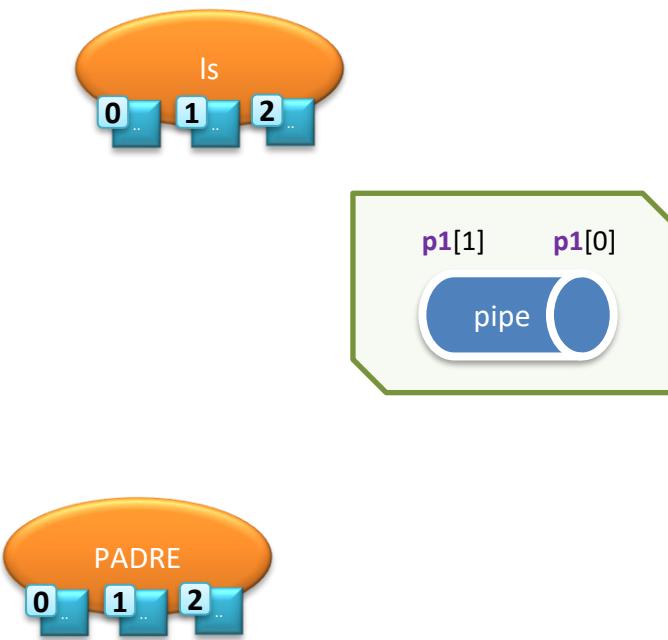
Ejemplo 2: "ls | sort | grep a"

(1/4) creación



Habría que hacer los 4 pasos para cada tubería (**pipe**, fork, redirección y limpieza)

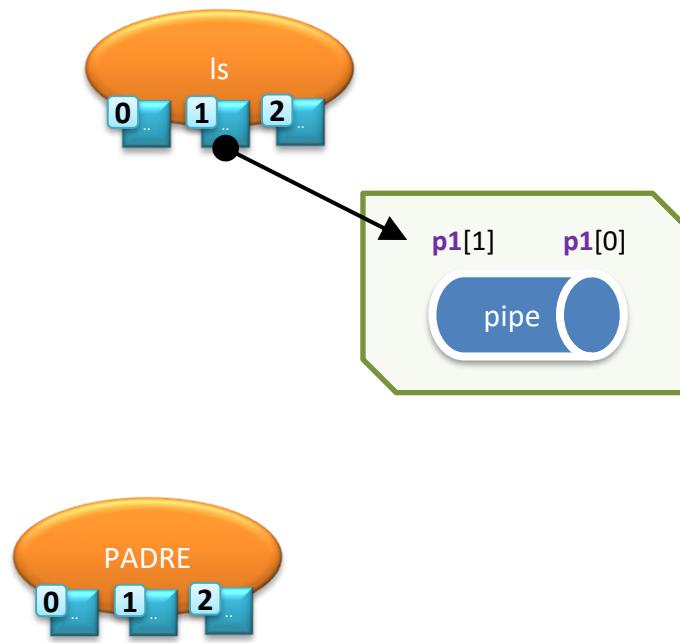
Ejemplo 2: "ls | sort | grep a" **(2/4) fork()**



Habría que hacer los 4 pasos para cada tubería (pipe, **fork**, redirección y limpieza)

Ejemplo 2: "ls | sort | grep a"

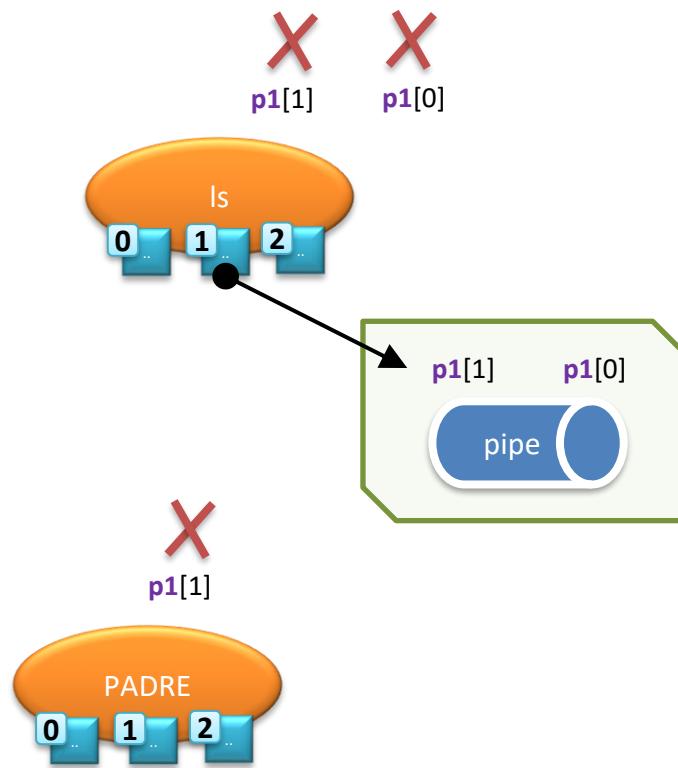
(3/4) redirección



Habría que hacer los 4 pasos para cada tubería (pipe, fork, **redirección** y limpieza)
El primer proceso solo utiliza la tubería en la redirección de salida.

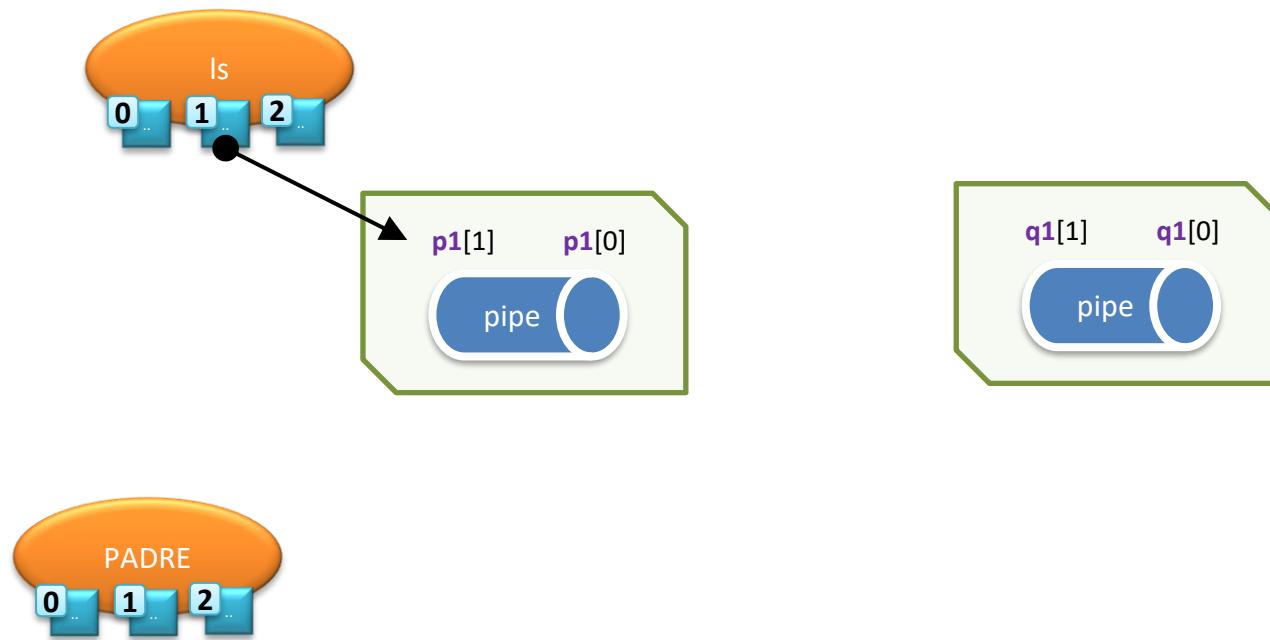
Ejemplo 2: "ls | sort | grep a"

(4/4) limpieza



Habría que hacer los 4 pasos para cada tubería (pipe, fork, redirección y **limpieza**)
El padre no haría todavía limpieza total para que el siguiente hijo tenga acceso a “p1[0]”

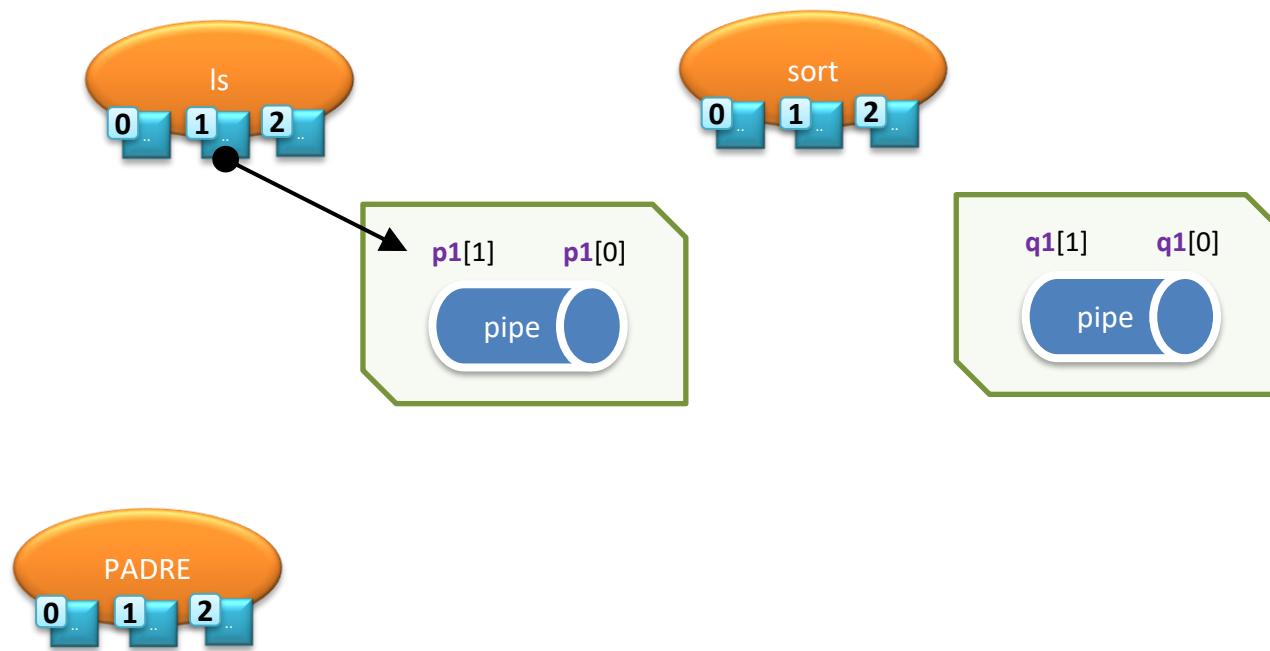
Ejemplo 2: "ls | sort | grep a" **(1/4) creación**



Habría que hacer los 4 pasos para cada tubería (**creación**, fork, redirección y limpieza)

Ejemplo 2: "ls | sort | grep a"

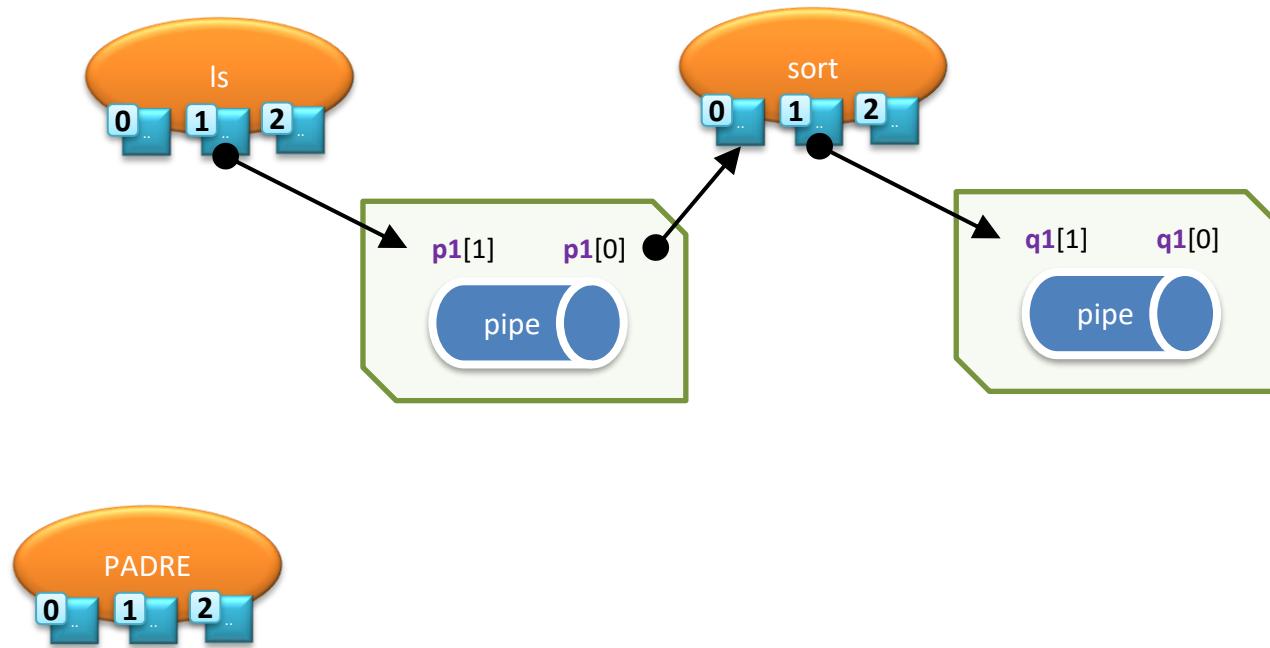
(2/4) fork()



Habría que hacer los 4 pasos para cada tubería (creación, **fork**, redirección y limpieza)

Ejemplo 2: "ls | sort | grep a"

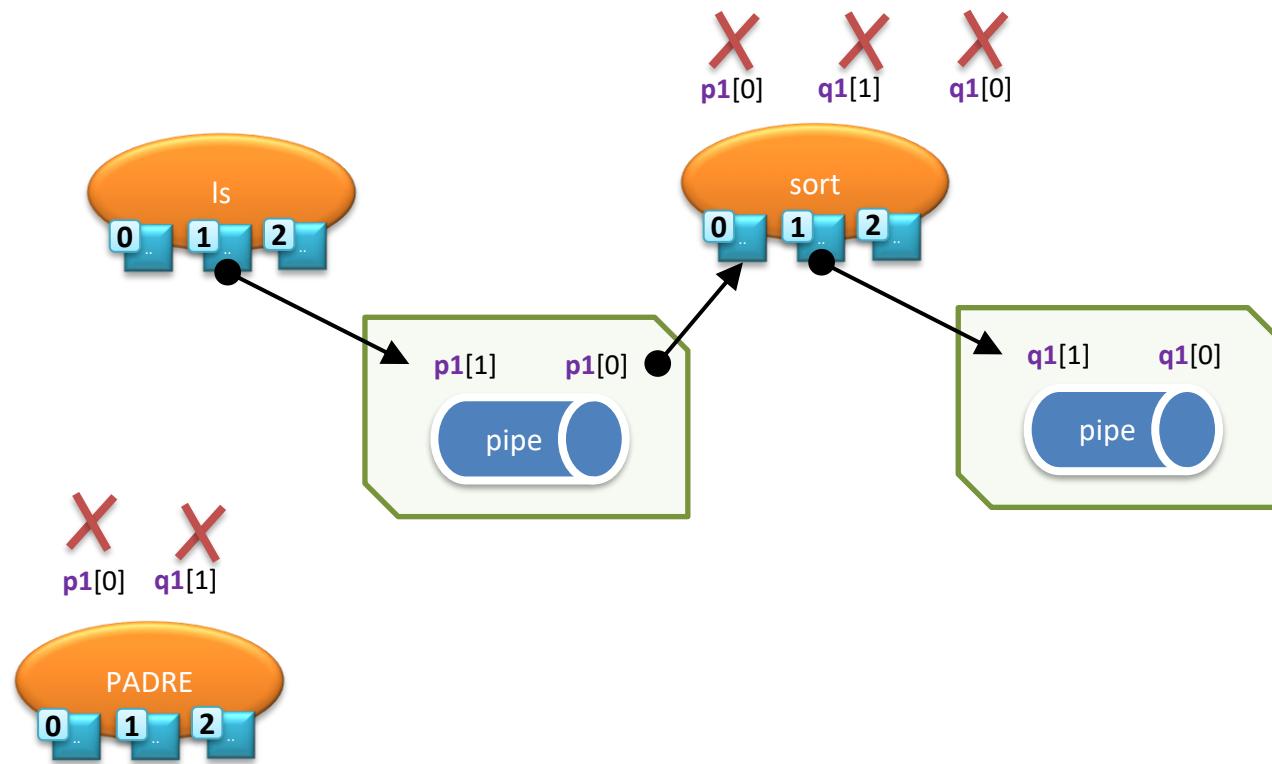
(3/4) redirección



Habría que hacer los 4 pasos para cada tubería (creación, fork, **redirección** y limpieza)
Los procesos intermedios redireccionan a tubería su entrada y salida estándar.

Ejemplo 2: "ls | sort | grep a"

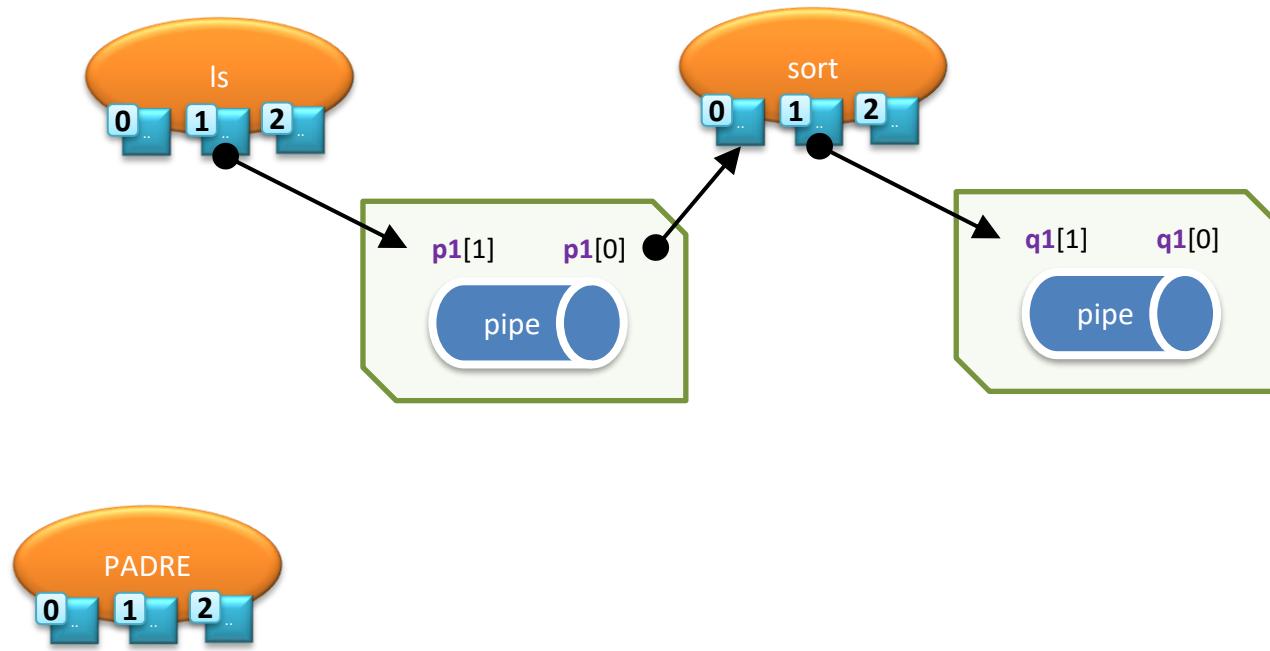
(4/4) limpieza



Habría que hacer los 4 pasos para cada tubería (creación, fork, redirección y **limpieza**)
El padre no haría todavía limpieza total para que el siguiente hijo tenga acceso a “q1[0]”

Ejemplo 2: "ls | sort | grep a"

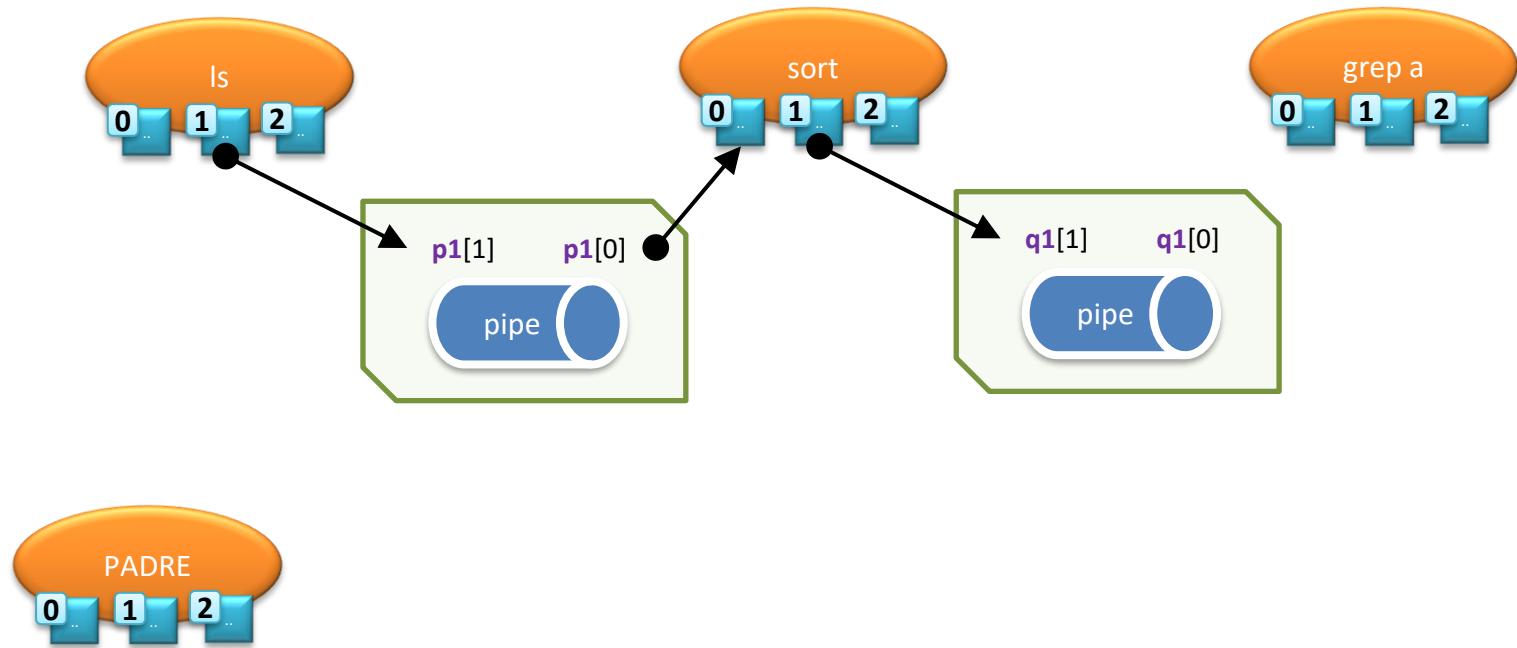
(1/4) creación



Habría que hacer los 4 pasos para cada tubería (**creación**, fork, redirección y limpieza)
El último proceso usa la tubería ya creada (hay n procesos y n-1 tuberías)

Ejemplo 2: "ls | sort | grep a"

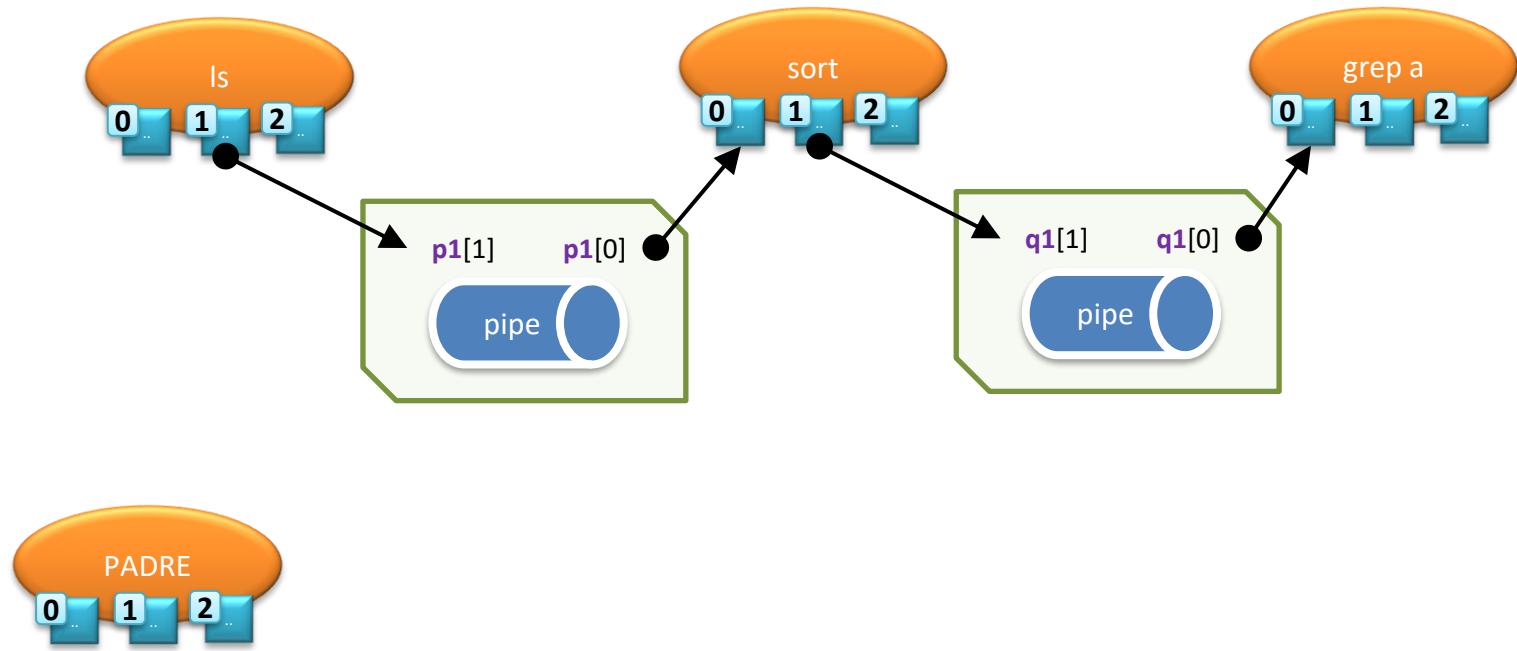
(2/4) fork()



Habría que hacer los 4 pasos para cada tubería (creación, **fork**, redirección y limpieza)

Ejemplo 2: "ls | sort | grep a"

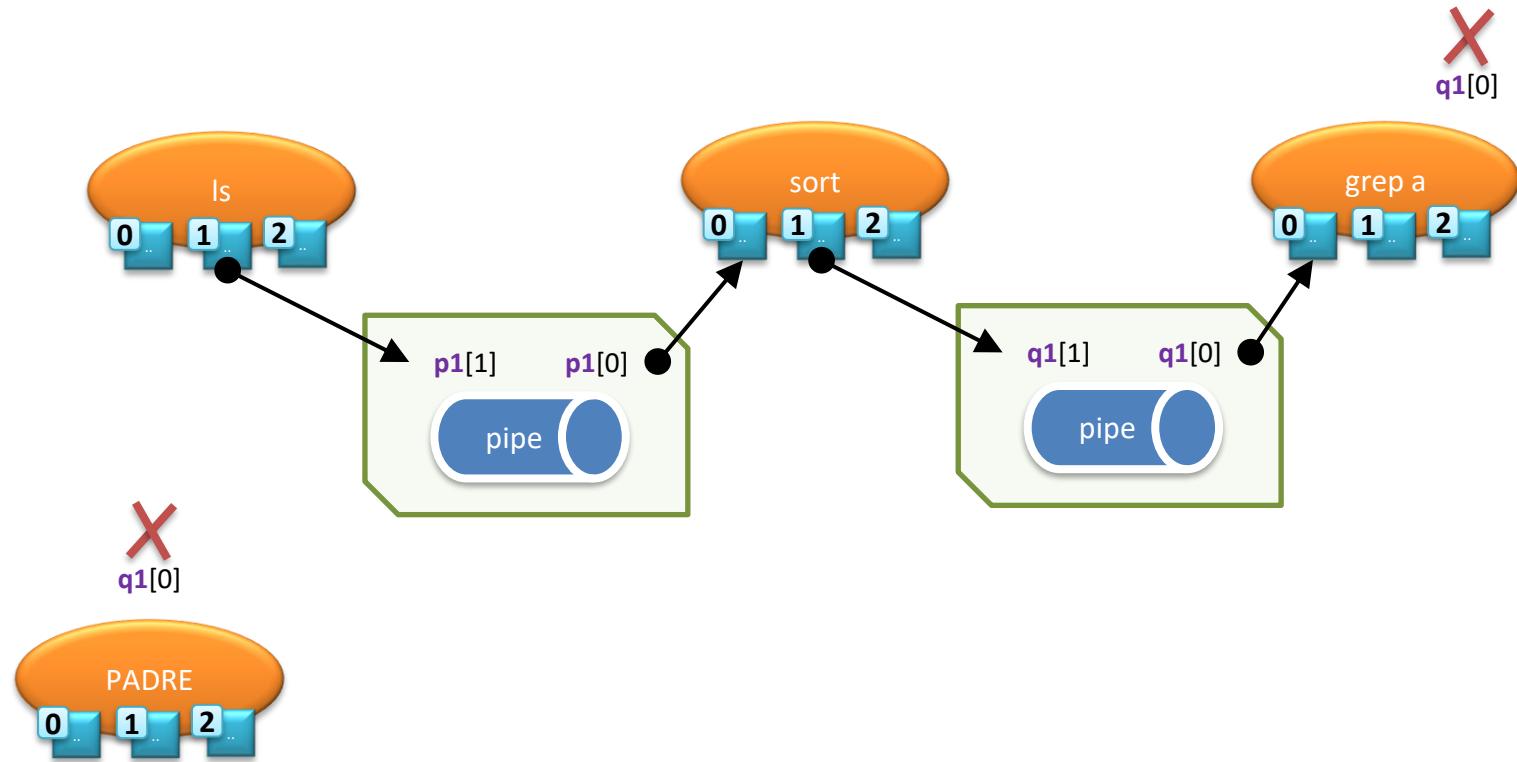
(3/4) redirección



Habría que hacer los 4 pasos para cada tubería (creación, fork, **redirección** y limpieza)
El último proceso usa la tubería para redirección en su entrada estándar.

Ejemplo 2: "ls | sort | grep a"

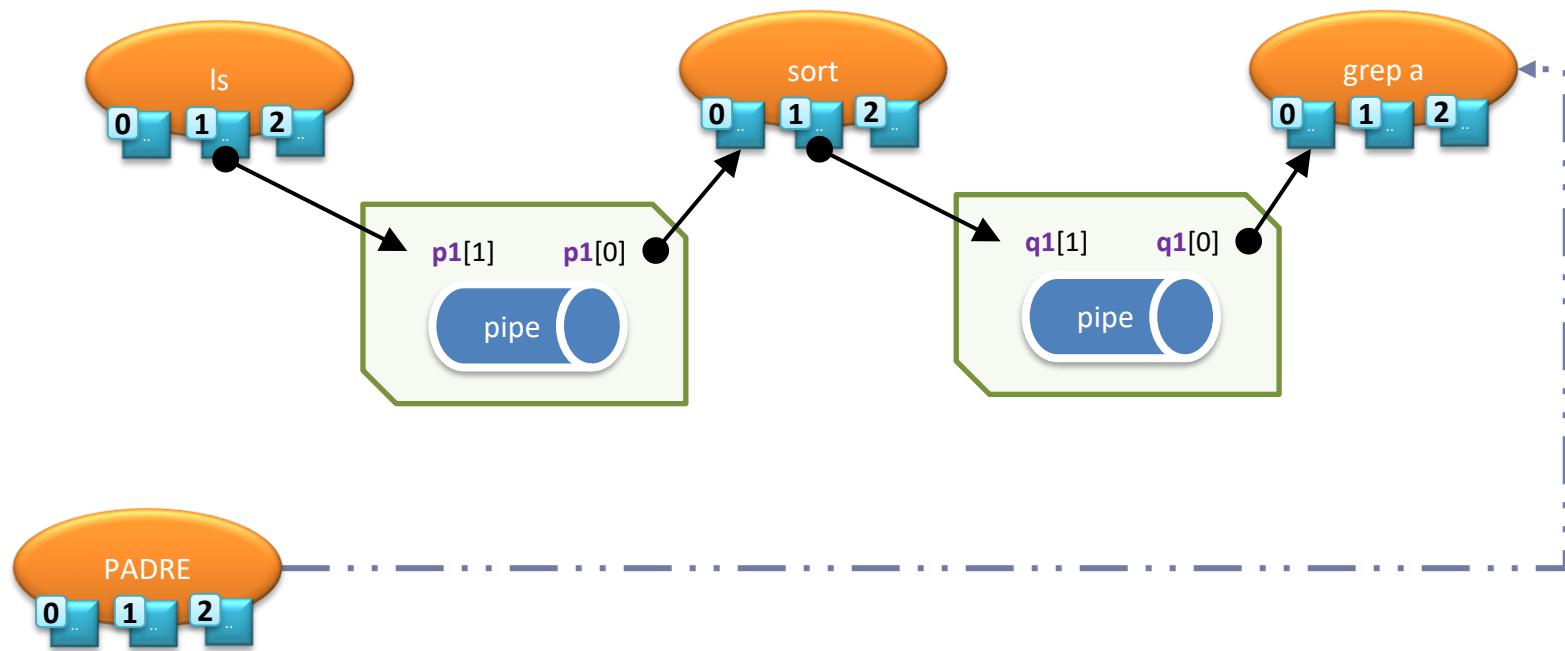
(4/4) limpieza



Habría que hacer los 4 pasos para cada tubería (creación, fork, redirección y **limpieza**)
El proceso padre cierra todos los descriptores restantes.

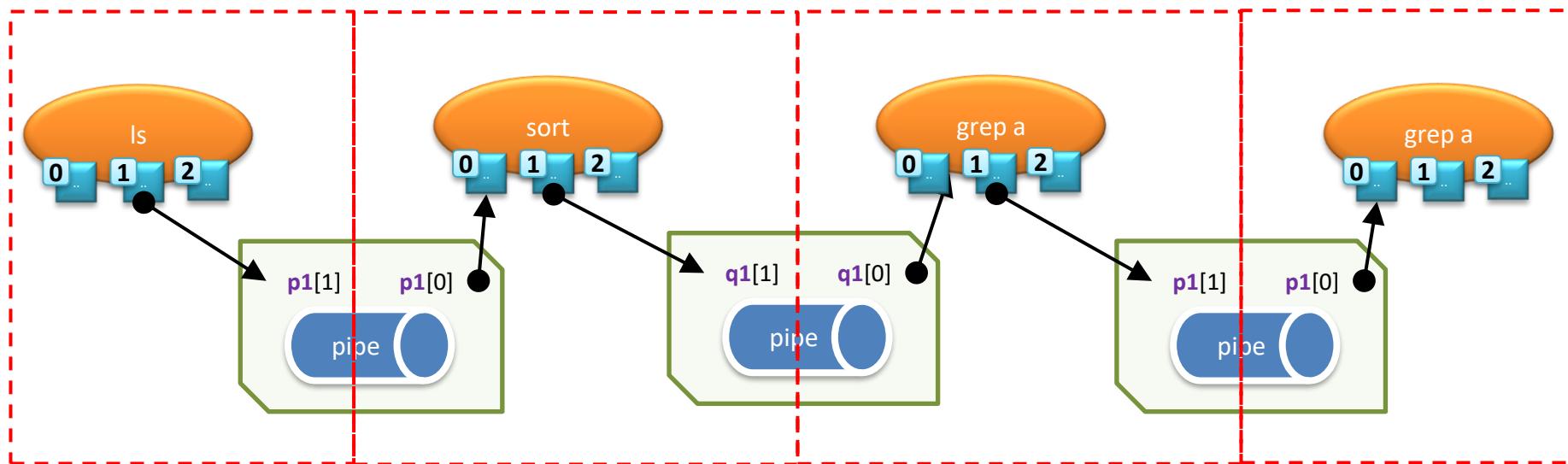
Ejemplo 2: "ls | sort | grep a"

wait por el último hijo creado



Si no es una tarea de fondo (*background*) entonces el proceso padre hace un `wait(...)` esperando por el último proceso (que se transforma en “grep a”)

Ejemplo 3: "ls | sort | uniq | grep a"



Con más tuberías el proceso es similar al descrito anteriormente.

Tres tipos de casos: primer proceso, procesos intermedios y último proceso.

Grupo ARCOS
Universidad Carlos III de Madrid

Lección 3

Señales, excepciones y pipes

Sistemas Operativos
Ingeniería Informática

