

Sistemas Operativos

sesión 12: tuberías

Grado en Ingeniería Informática

Universidad Carlos III de Madrid

Agenda



Linux



Comunicación
con tuberías



Ejercicios



Agenda



Linux



Comunicación
con tuberías



Ejercicios



Contenidos



- Redirección
- Tuberías (pipes)

Contenidos



- **Redirección**
- Tuberías (pipes)

Ejemplo de redirección de entrada

f1.txt

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

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

grep ocho < f1

Ejemplo de redirección de salida

f1.txt		
uno,	dos,	tres
cuatro,	cinco,	seis
siete,	ocho,	nueve
diez,	once,	doce

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


grep ocho < f1 > s1

siete, ocho, nueve

Ejemplo de redirección de salida

f1.txt		
uno,	dos,	tres
cuatro,	cinco,	seis
siete,	ocho,	nueve
diez,	once,	doce

Dependiente del
intérprete de
mandatos usado

siete, ocho, nueve

grep ocho f1 1> s1

Ejemplo de redirección de **error**

f1.txt		
uno,	dos,	tres
cuatro,	cinco,	seis
siete,	ocho,	nueve
diez,	once,	doce

Dependiente del
intérprete de
mandatos usado

grep: f2: No existe el archivo o el directorio

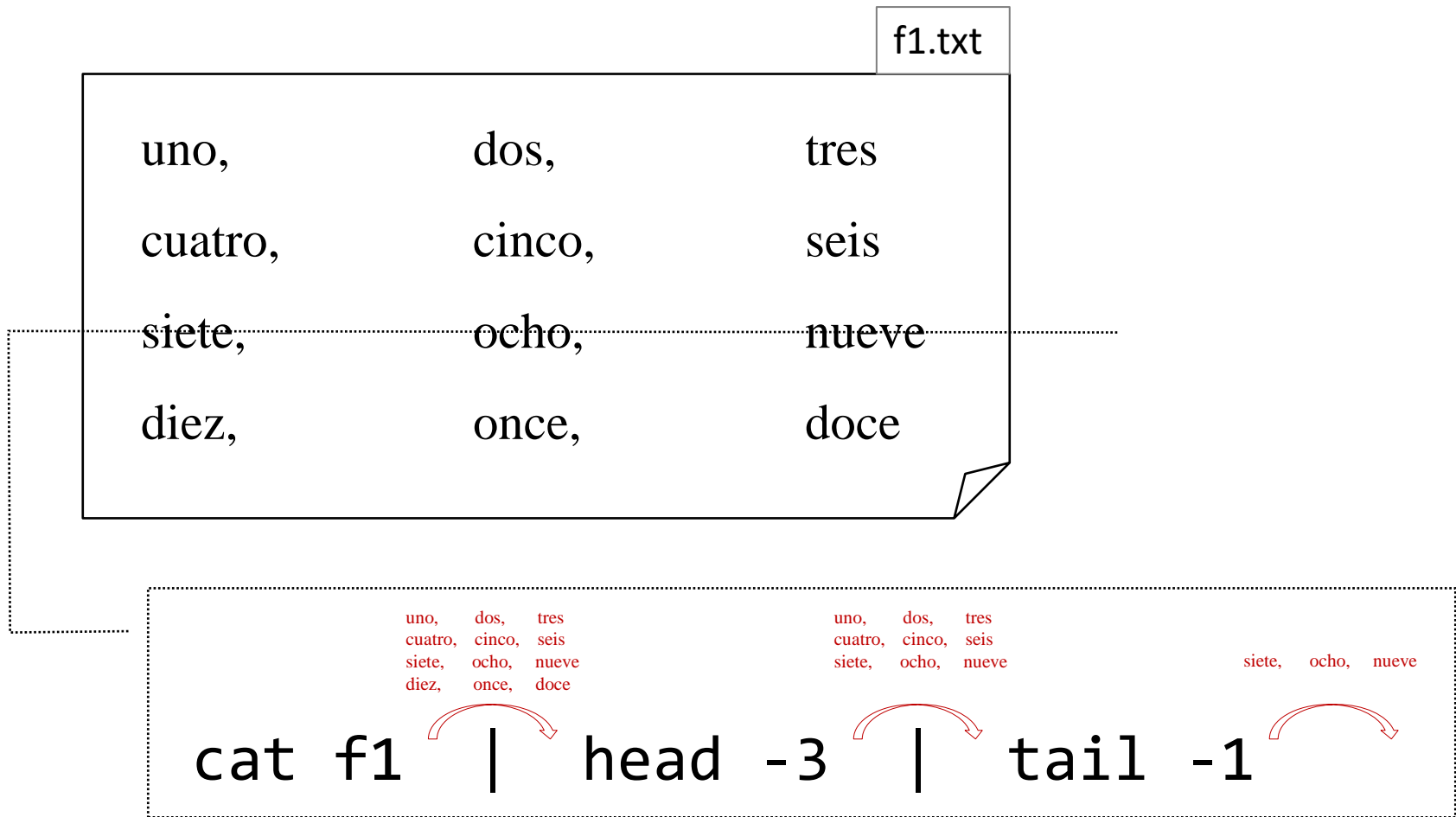
grep ocho xx 2> s1

Contenidos



- Redirección
- **Tuberías (pipes)**

Ejemplo de uso de tuberías



Agenda



Linux



**Comunicación
con tuberías**



Ejercicios



Contenidos



- Los descriptores de ficheros
 - Redirección y duplicado
- Los descriptores de ficheros y *fork()*
- Tuberías

Contenidos



- **Los descriptores de ficheros**
 - Redirección y duplicado
- Los descriptores de ficheros y *fork()*
- Tuberías

Descriptores de ficheros



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

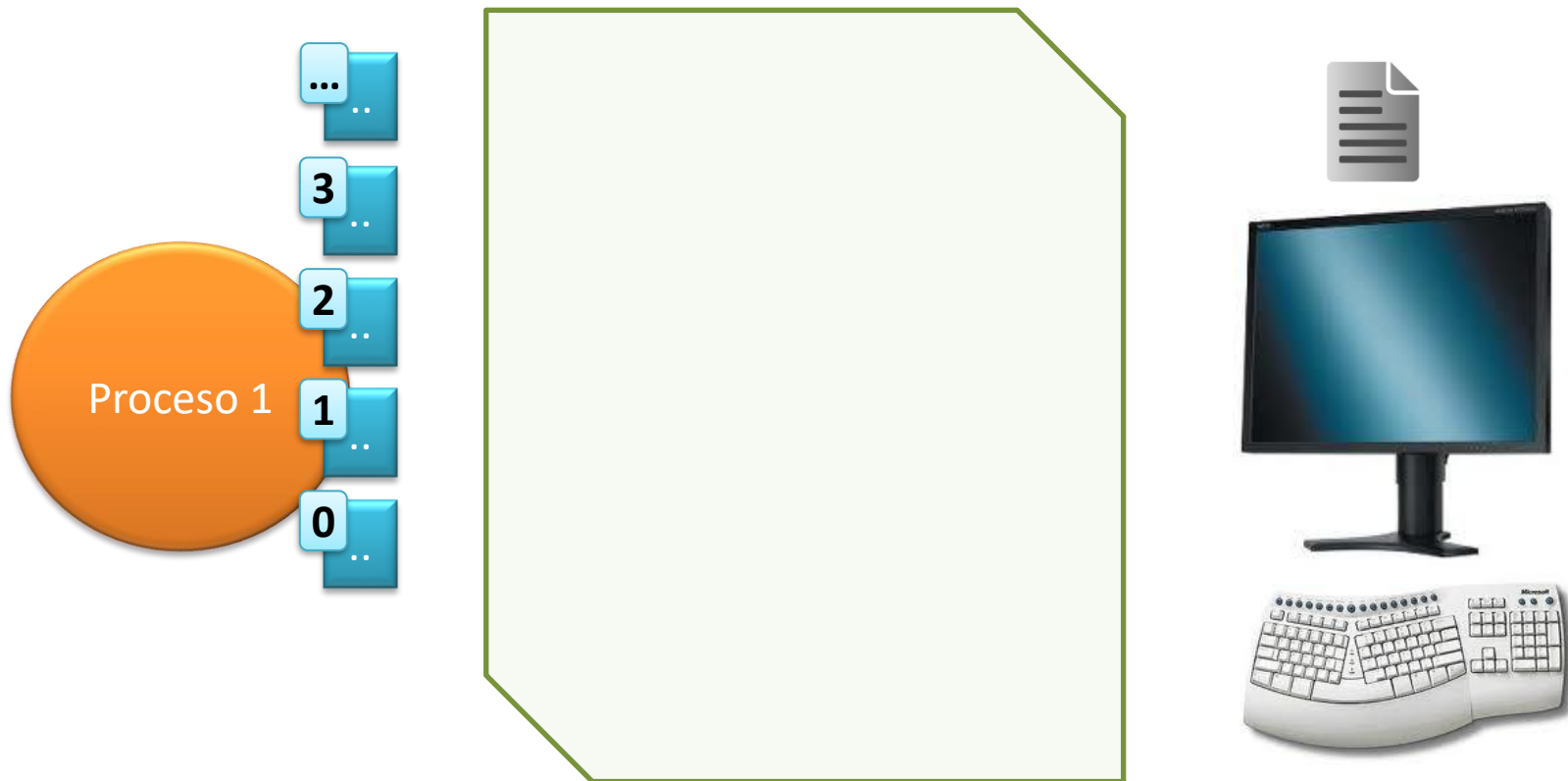
Descriptores de ficheros



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

Descriptores de ficheros

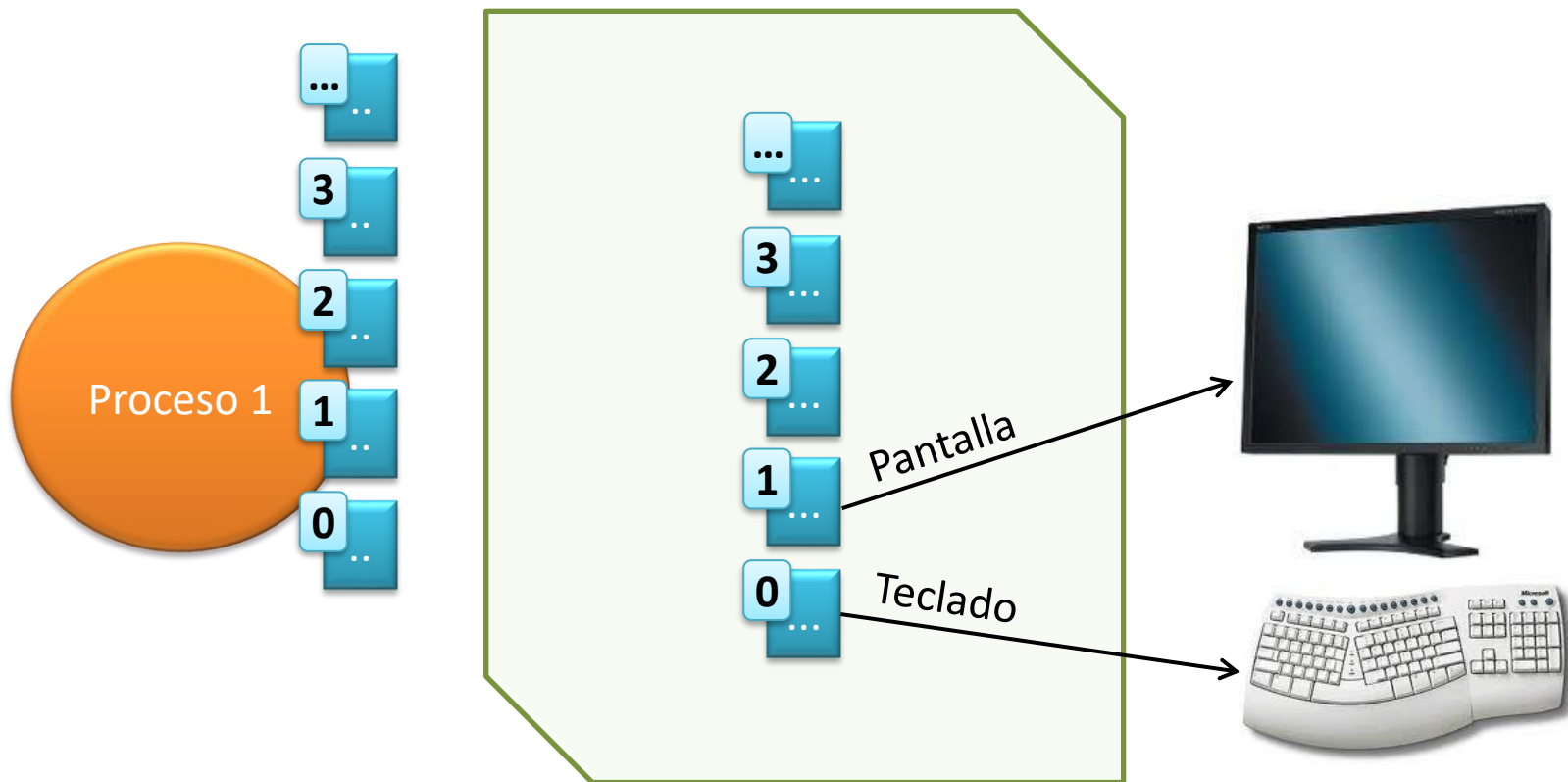
abstracción ofrecida



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

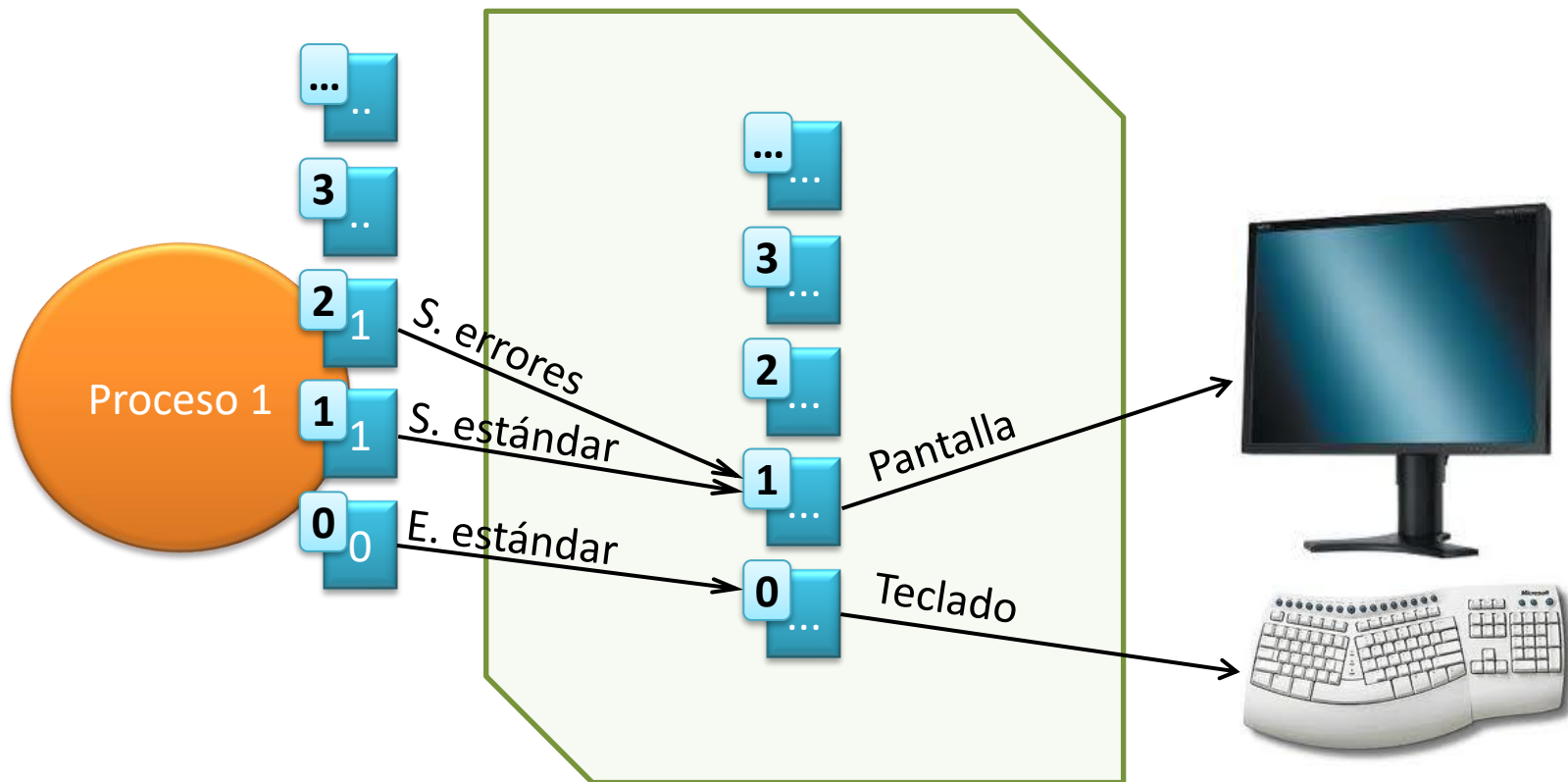
abstracción ofrecida



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

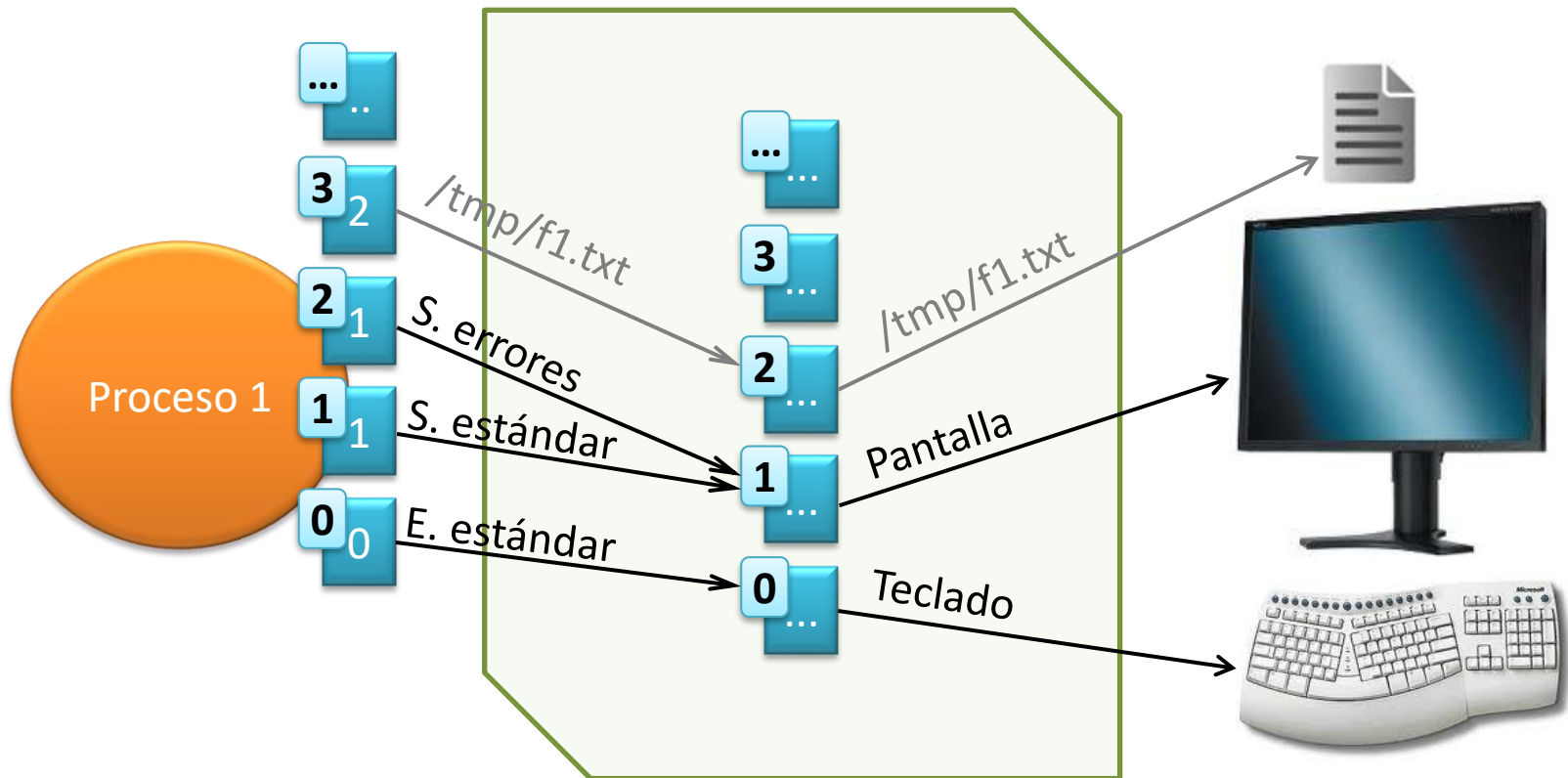
abstracción ofrecida



...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

abstracción ofrecida



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.

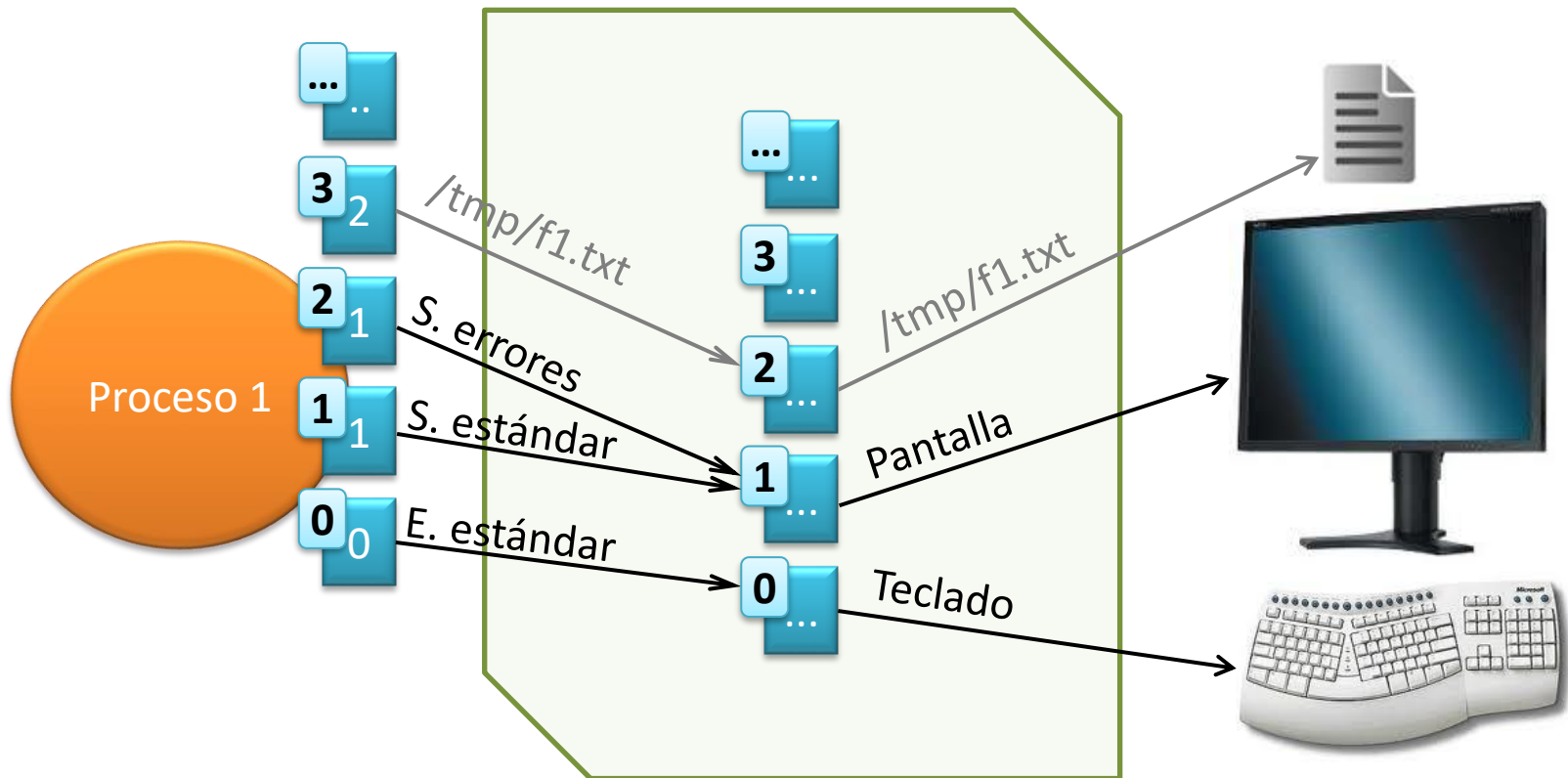
Contenidos



- Los descriptores de ficheros
 - Redirección y duplicado
- Los descriptores de ficheros y *fork()*
- Tuberías

Descriptores de ficheros

redirección a fichero

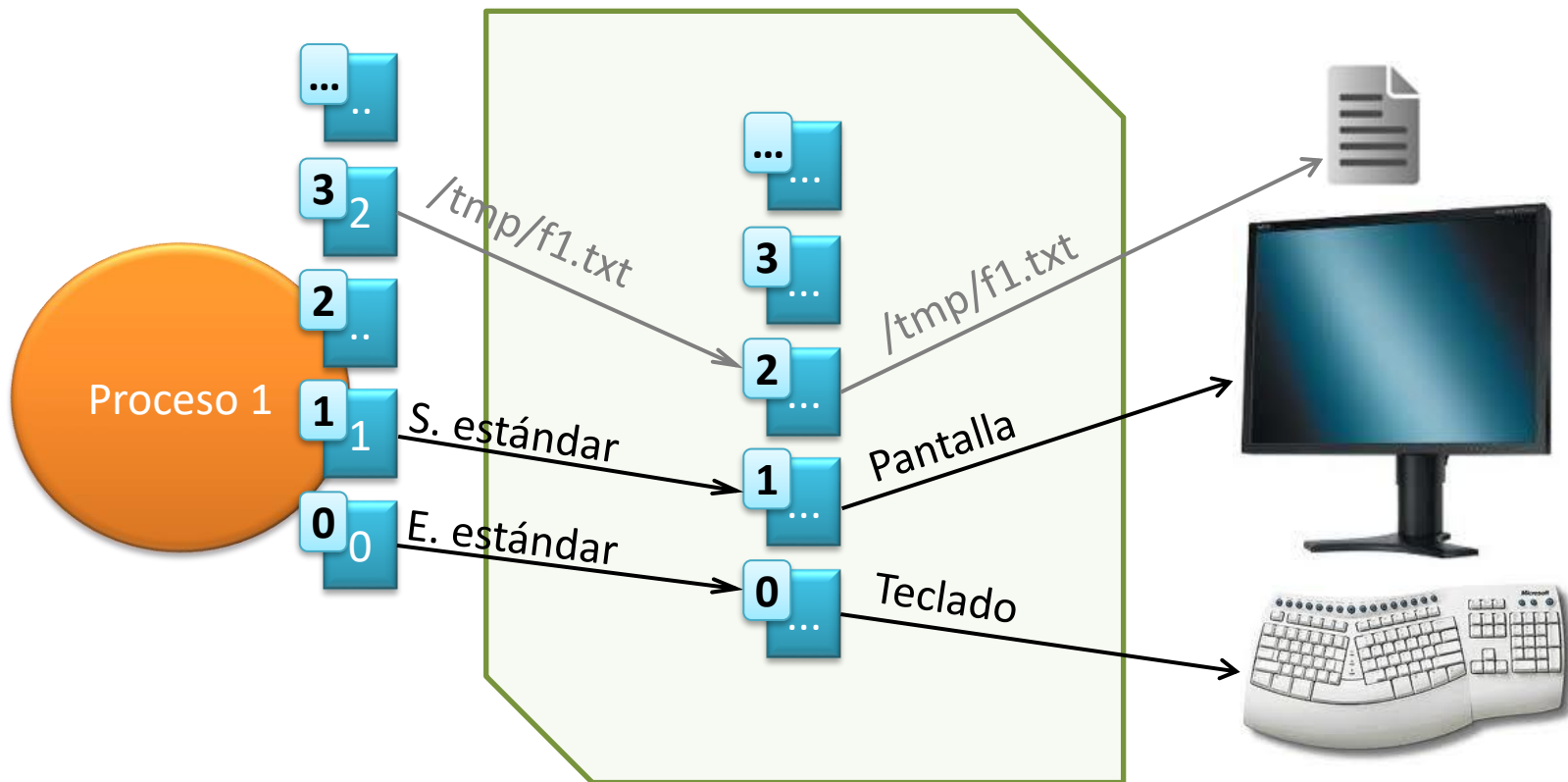


1. `close(2) ;`
2. `open("/tmp/errores.txt") ;`



Descriptores de ficheros

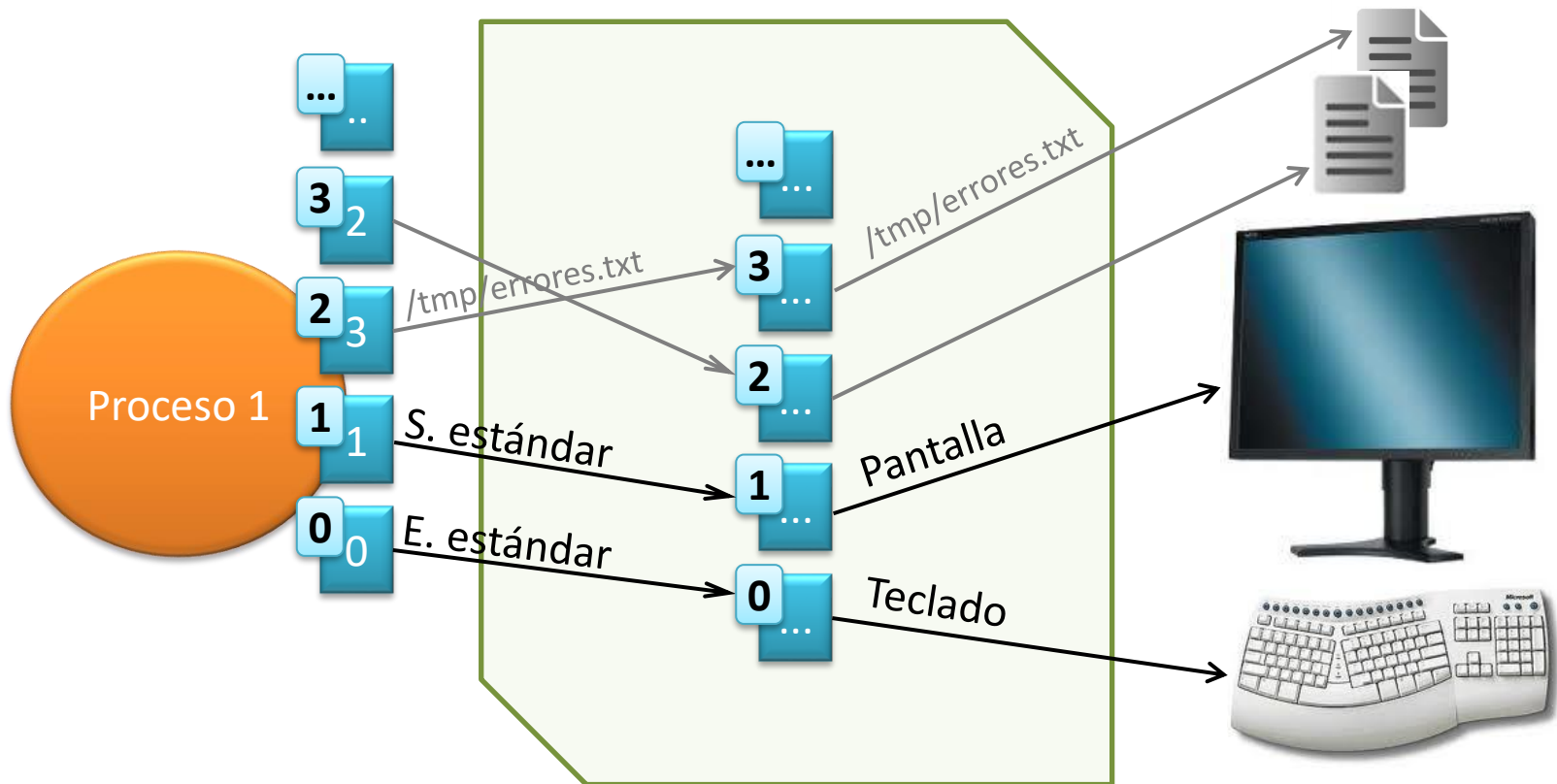
redirección a fichero



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

Descriptores de ficheros

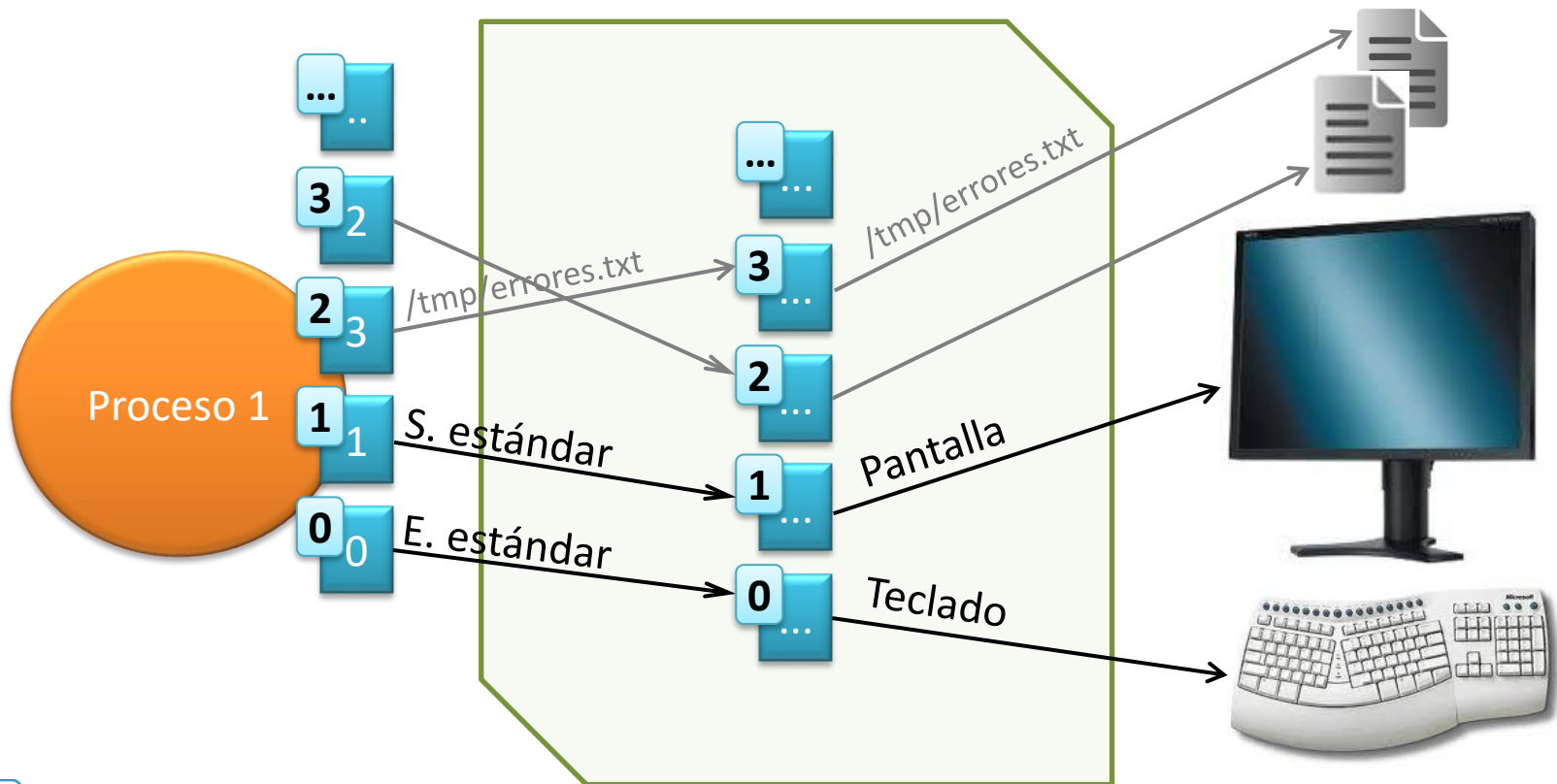
redirección a fichero



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

Descriptores de ficheros

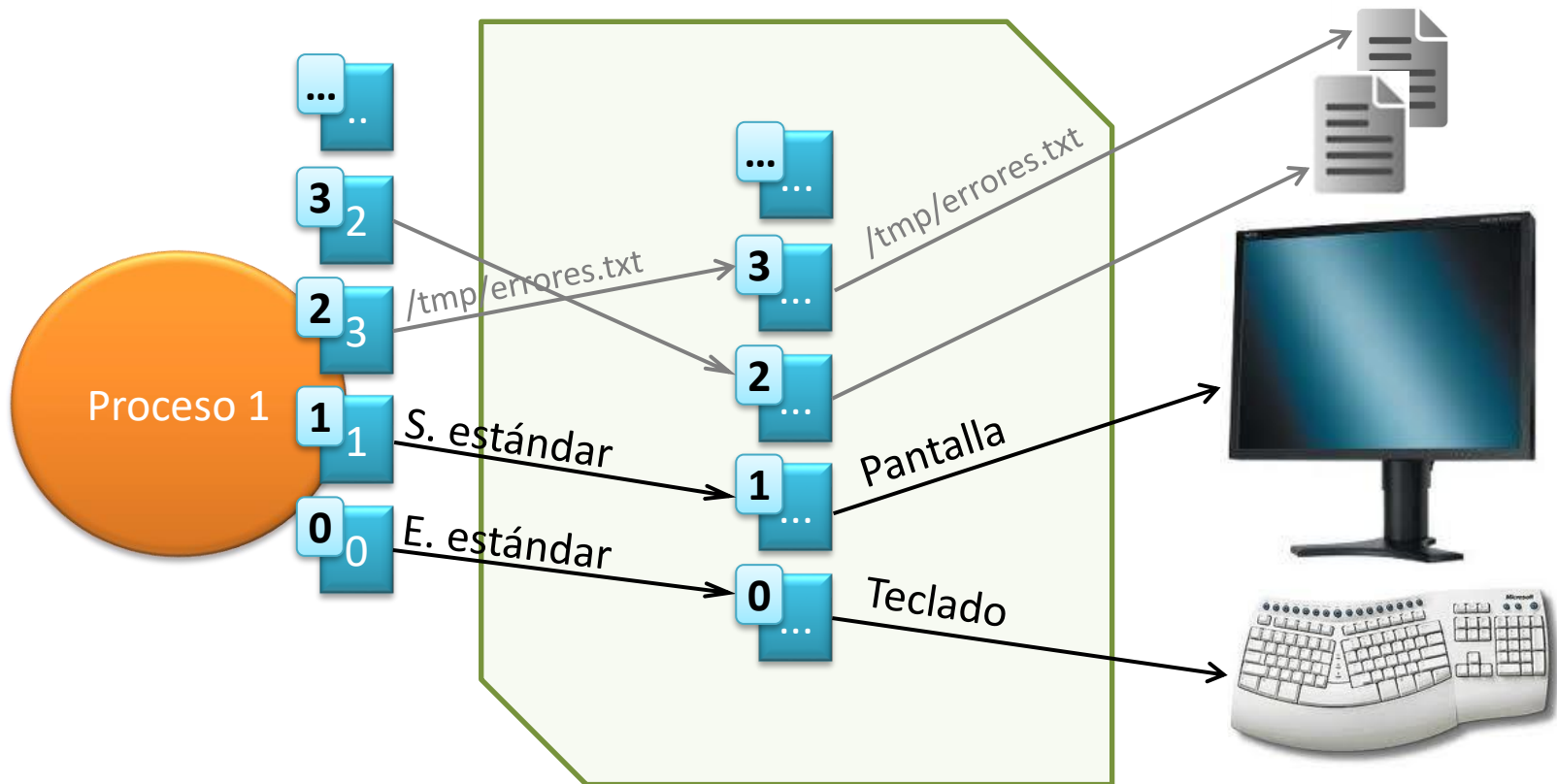
redirección a fichero



`close(2) + open("/tmp/errores.txt")`
Es posible cambiar el archivo asociado a un descriptor.

Descriptores de ficheros

duplicación de descriptor

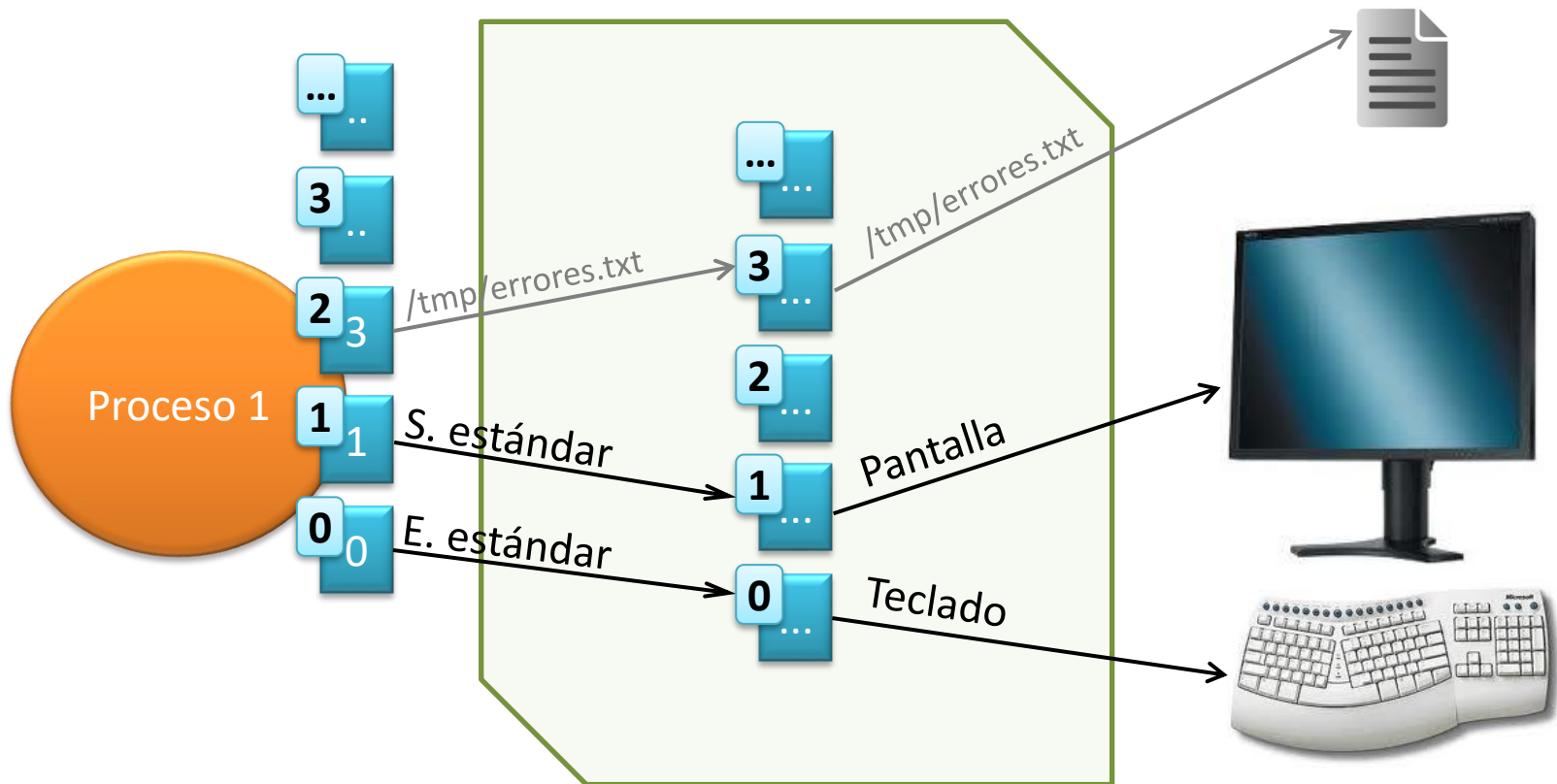


1. `close(3);`
2. `dup(2);`



Descriptores de ficheros

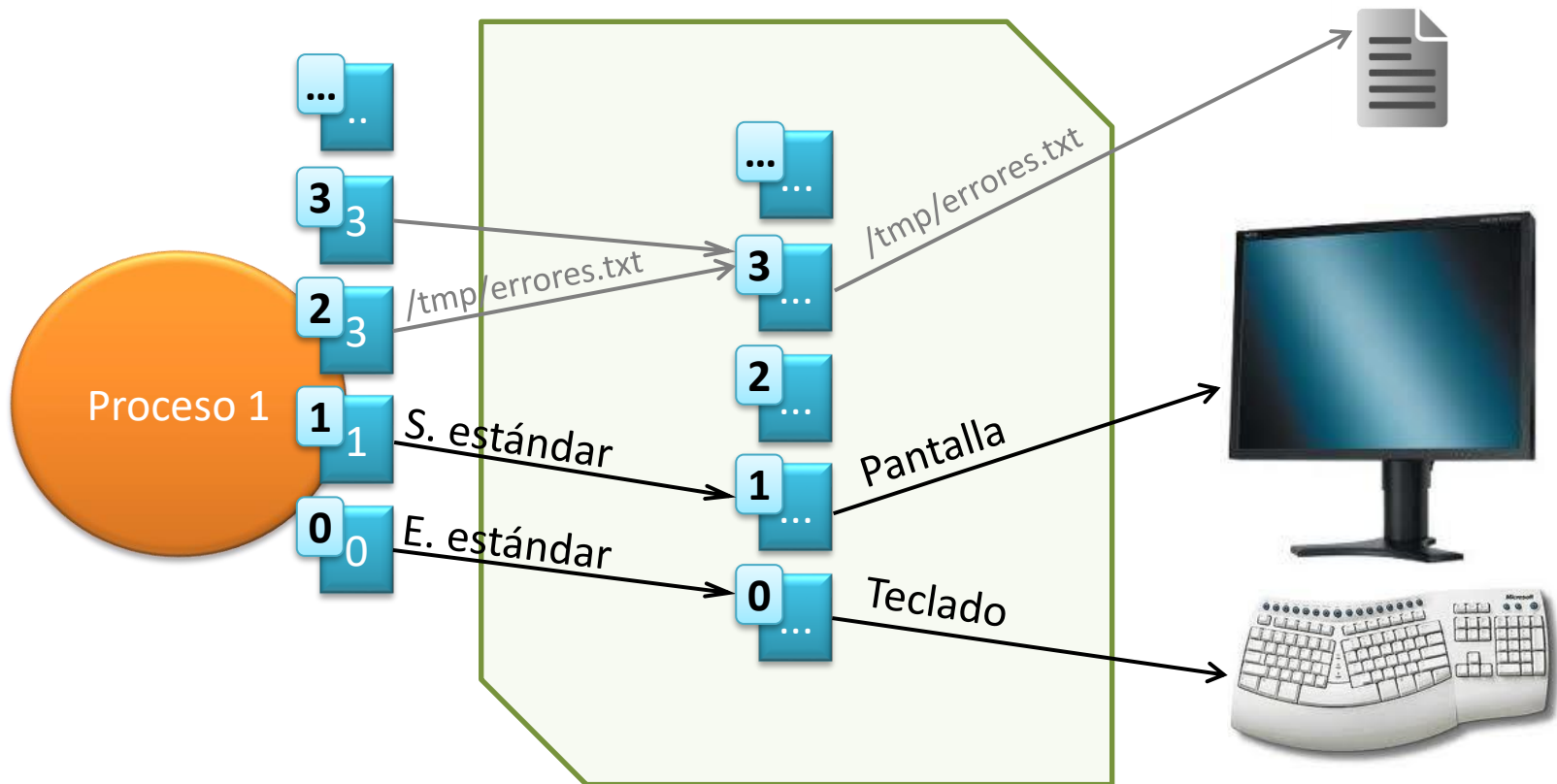
duplicación de descriptor



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

Descriptores de ficheros

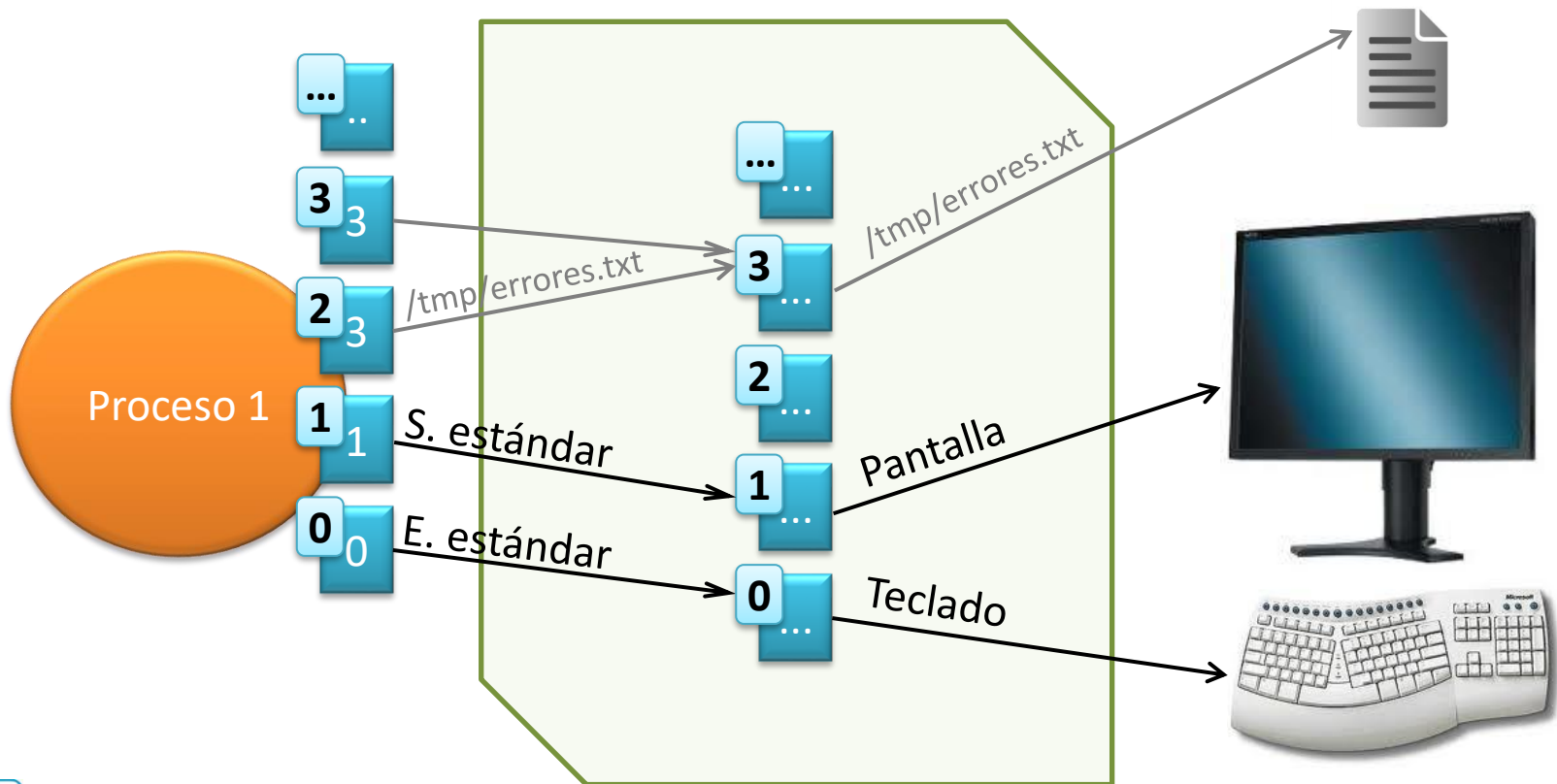
duplicación de descriptor



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

Descriptores de ficheros

duplicación de descriptor



`close(3) + dup(2)`

Permite acceder a un mismo fichero desde dos descriptores diferentes

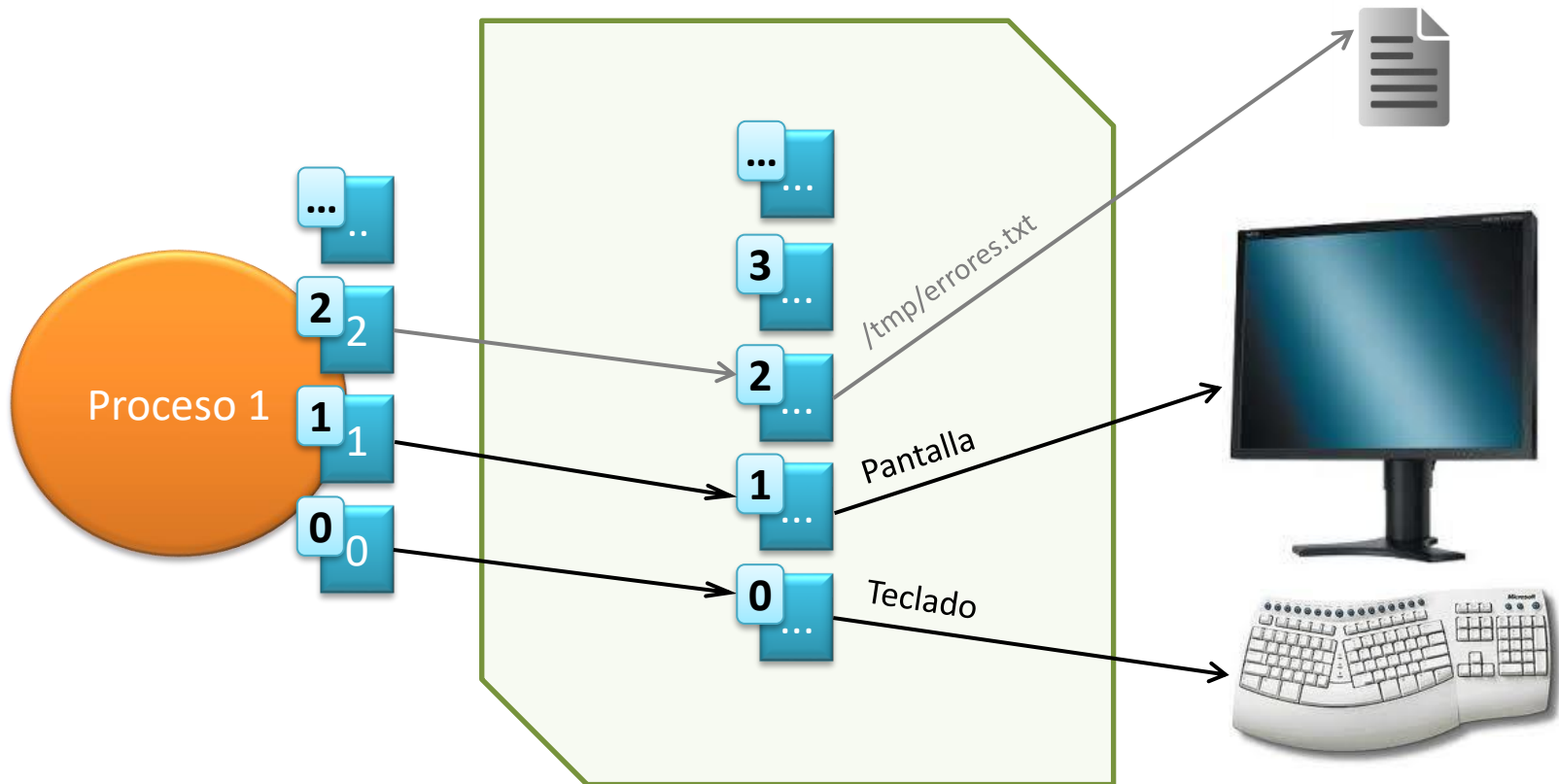
Contenidos



- Los descriptores de ficheros
 - Redirección y duplicado
- **Los descriptores de ficheros y *fork()***
- Tuberías

Descriptores de ficheros

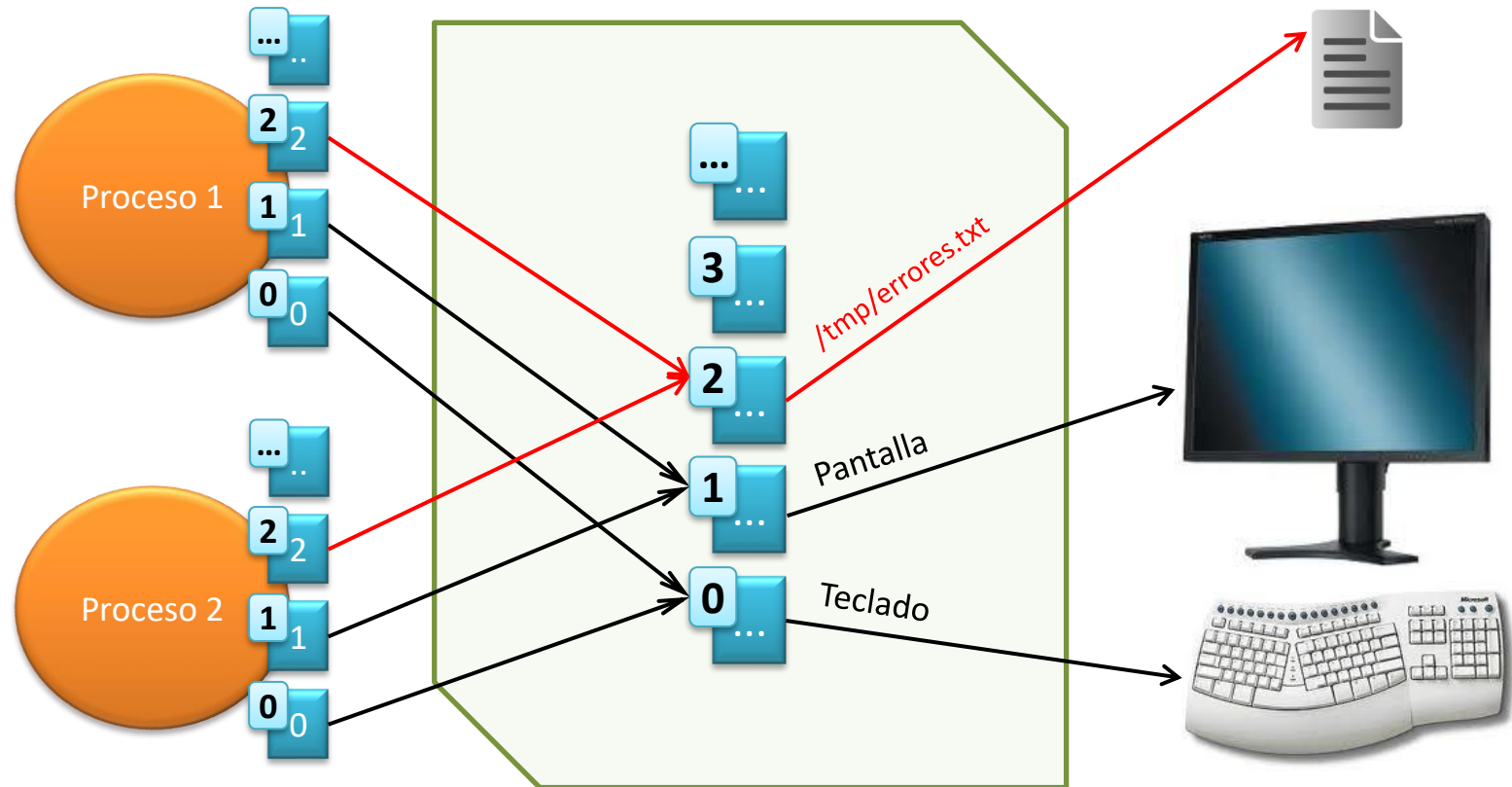
llamada fork()



`fork()` crea un duplicado del hijo

Descriptores de ficheros

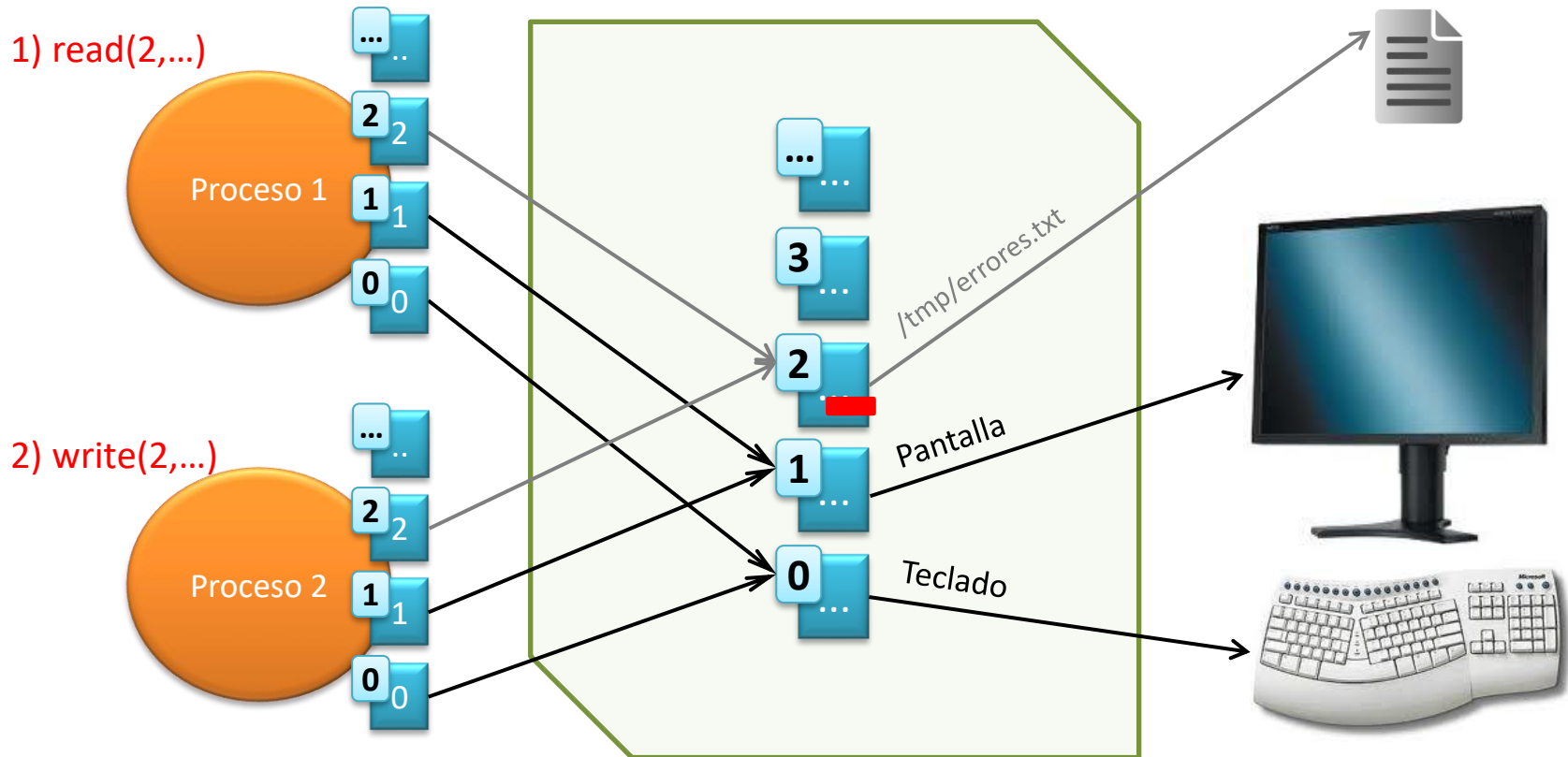
llamada `fork()`



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

Descriptores de ficheros

llamada `fork()`



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

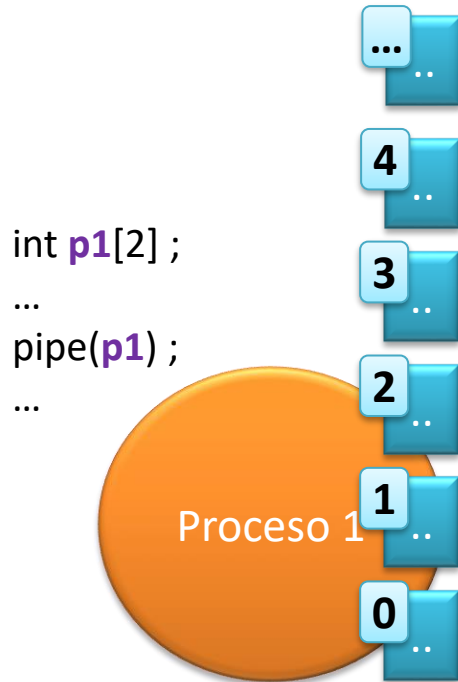
Contenidos



- Los descriptores de ficheros
 - Redirección y duplicado
- Los descriptores de ficheros y *fork()*
- **Tuberías**

Tubería

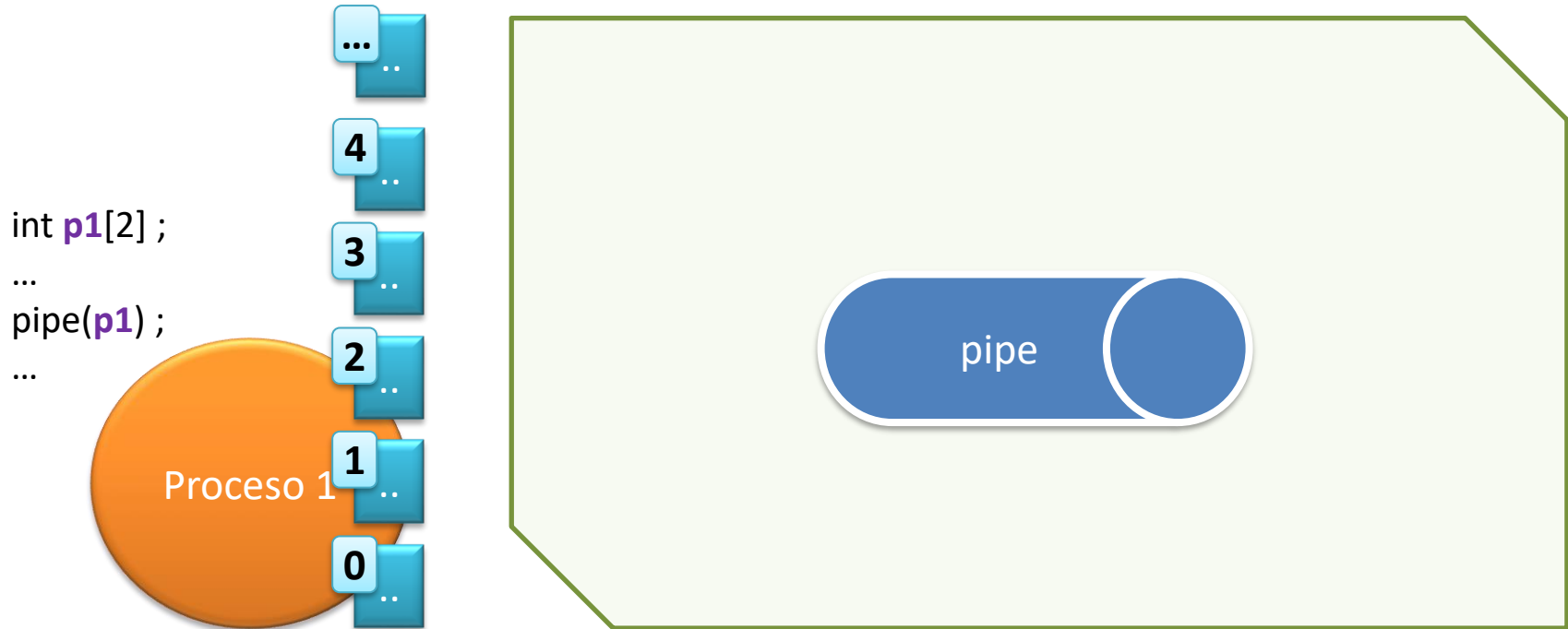
1 creación



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

Tubería

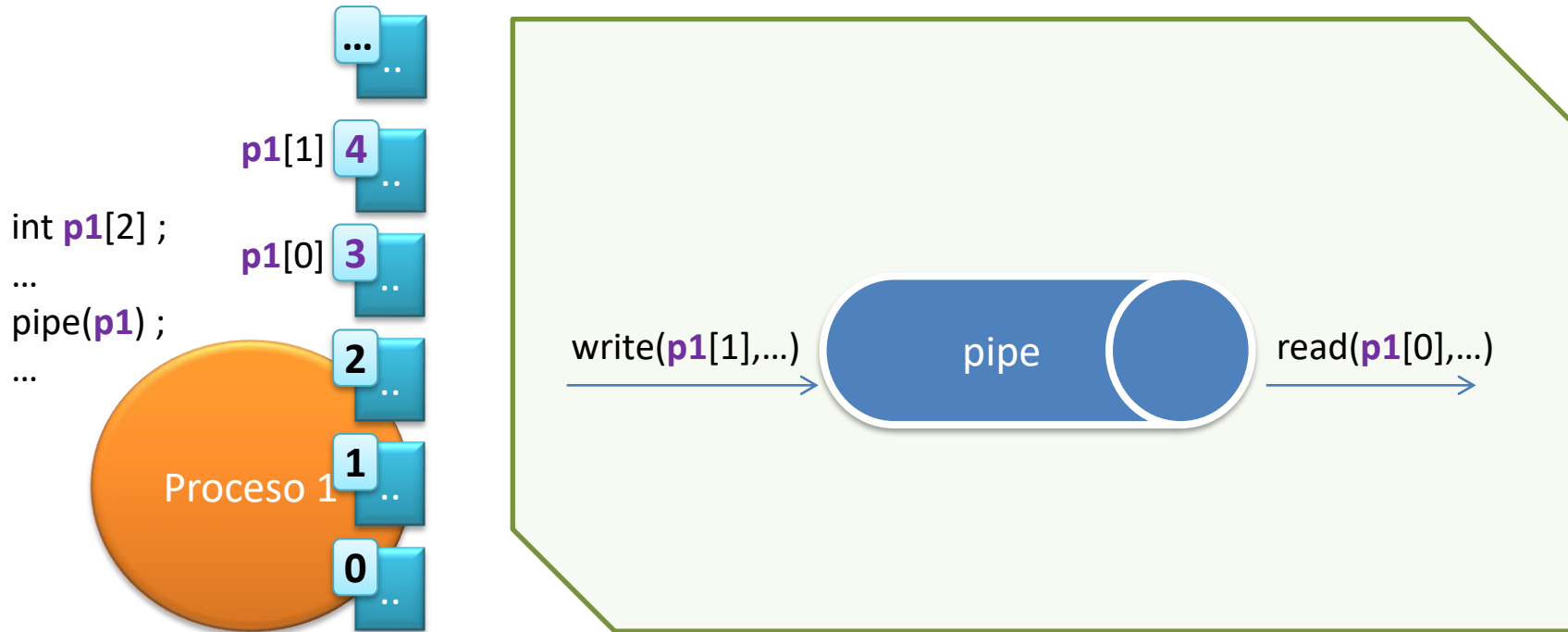
1 creación



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

Tubería

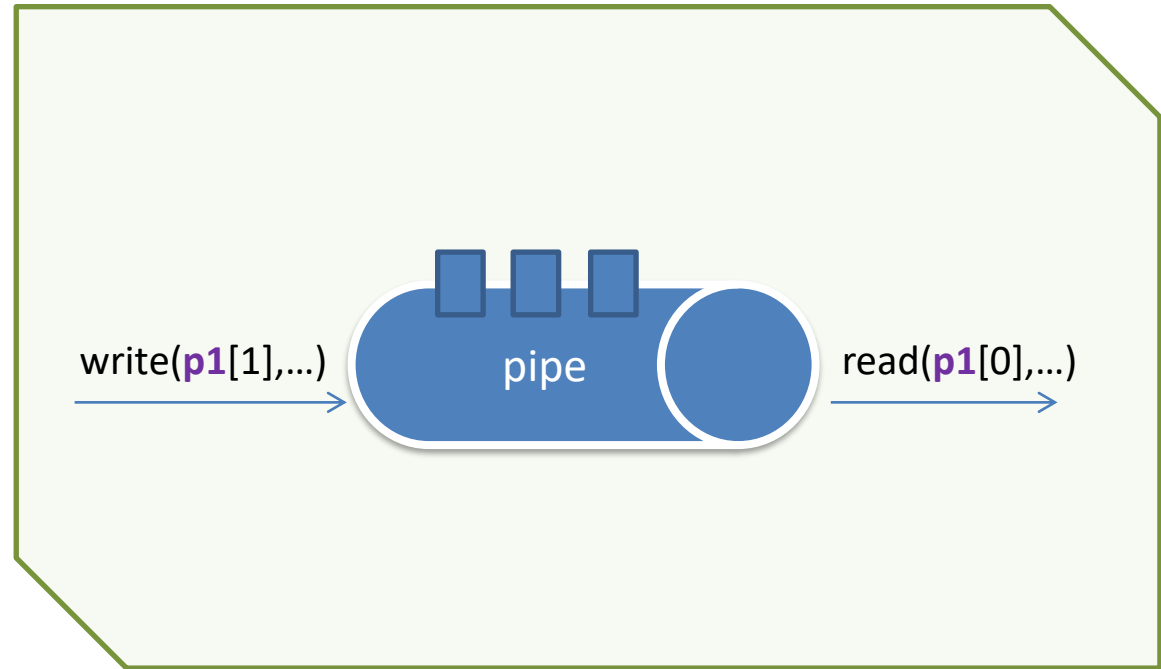
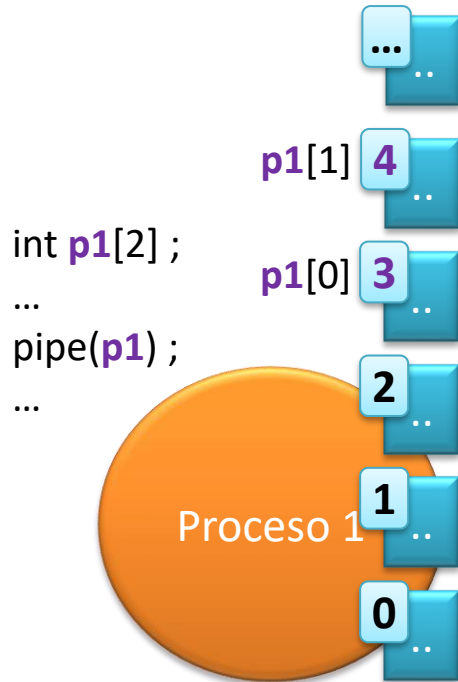
1 creación



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

Tubería

1 creación

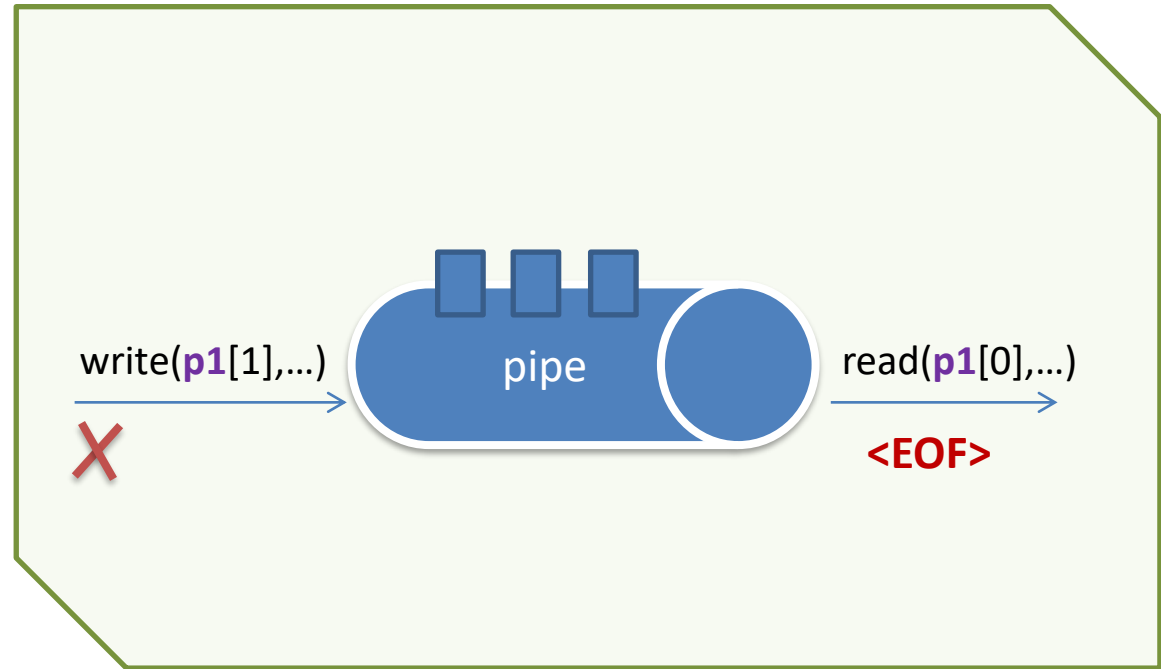
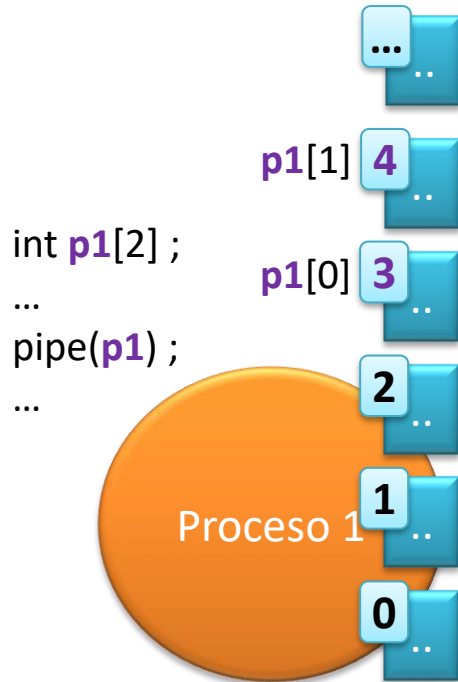


Tip

- 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.

Tubería

1 creación

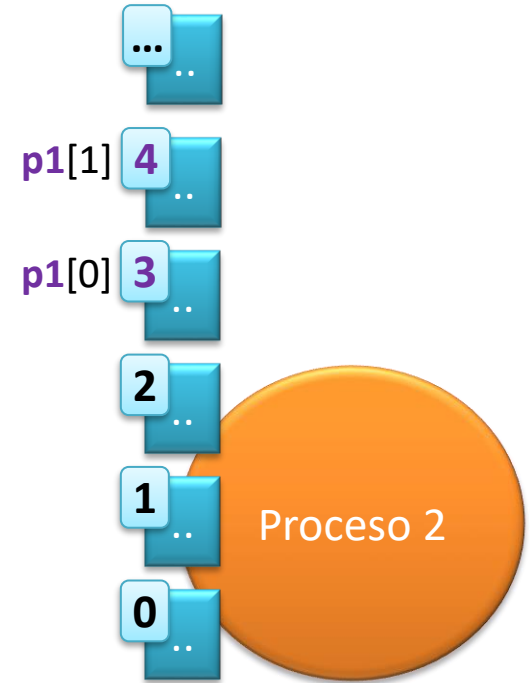
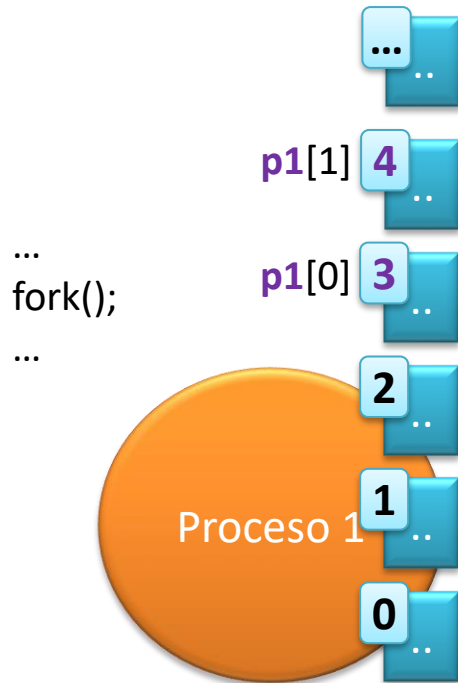


Tip

- Cuando todos los procesos escritores cierran la parte de escritura, entonces se manda un final de fichero (EOF) a los lectores.

Tubería

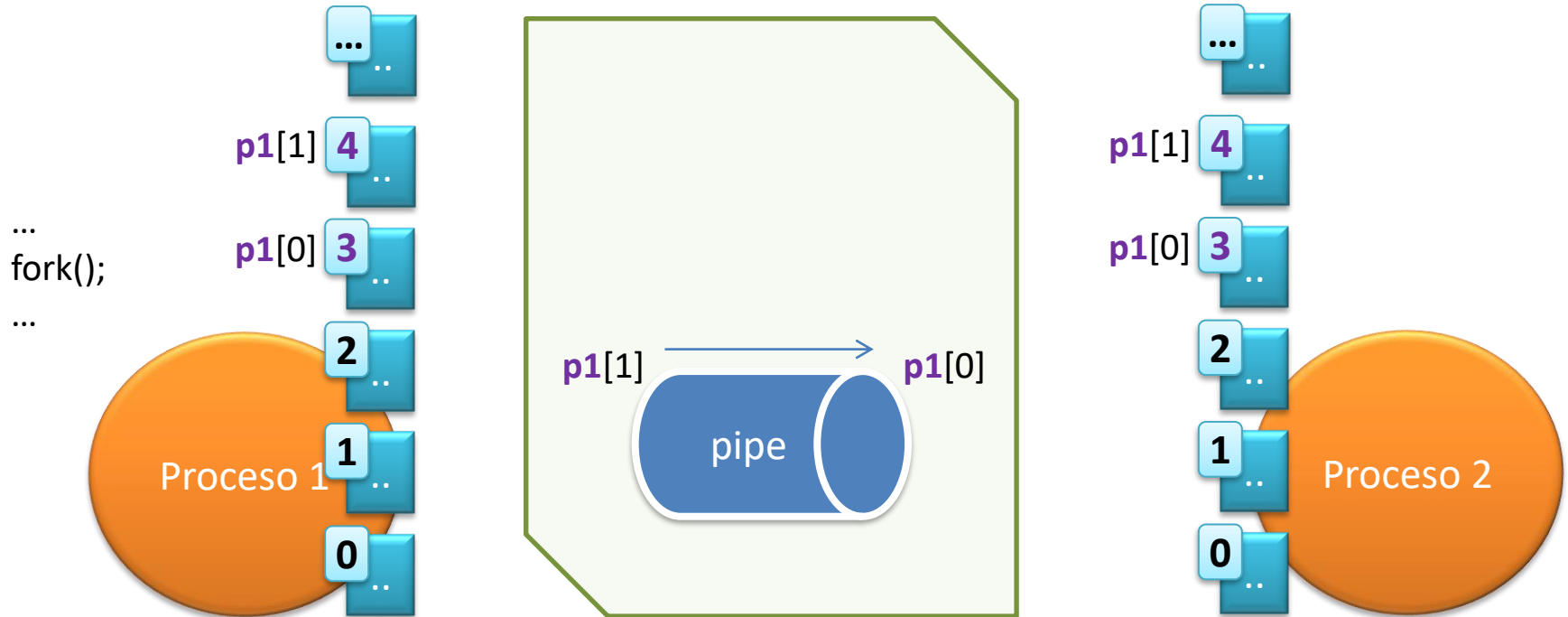
2 fork()



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

Tubería

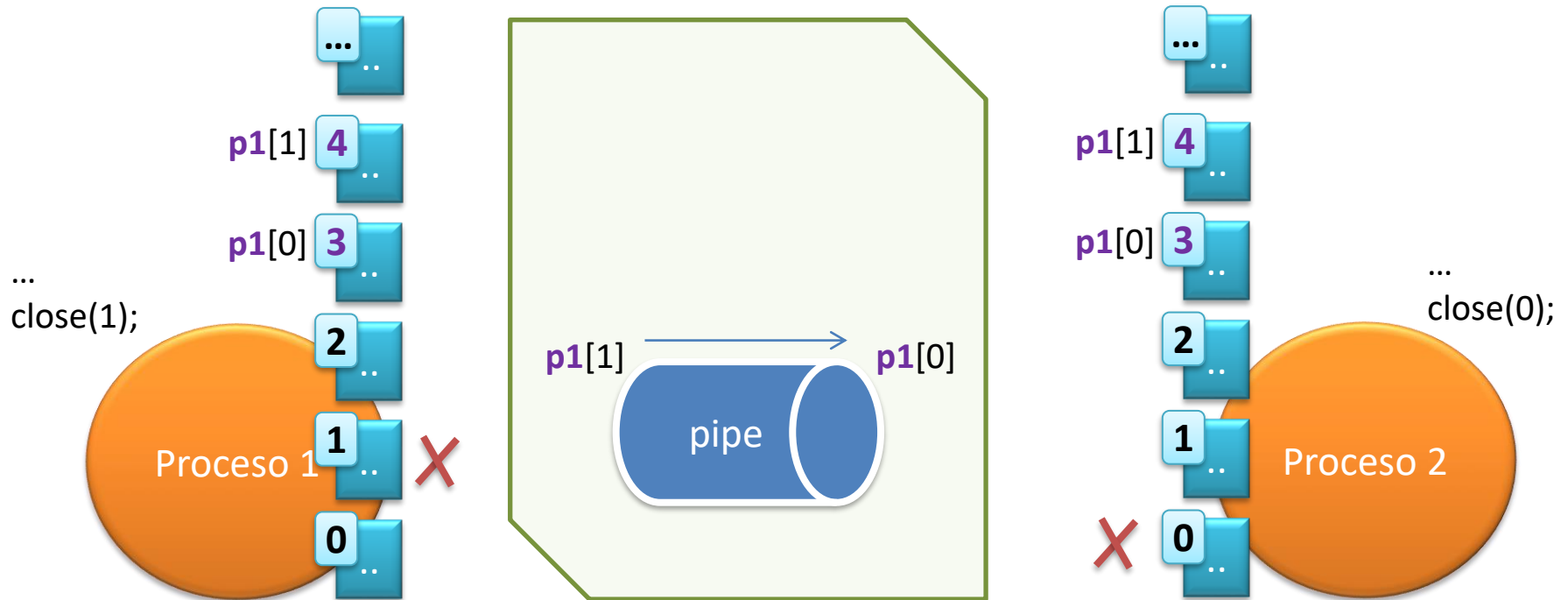
2 fork()



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

Tubería

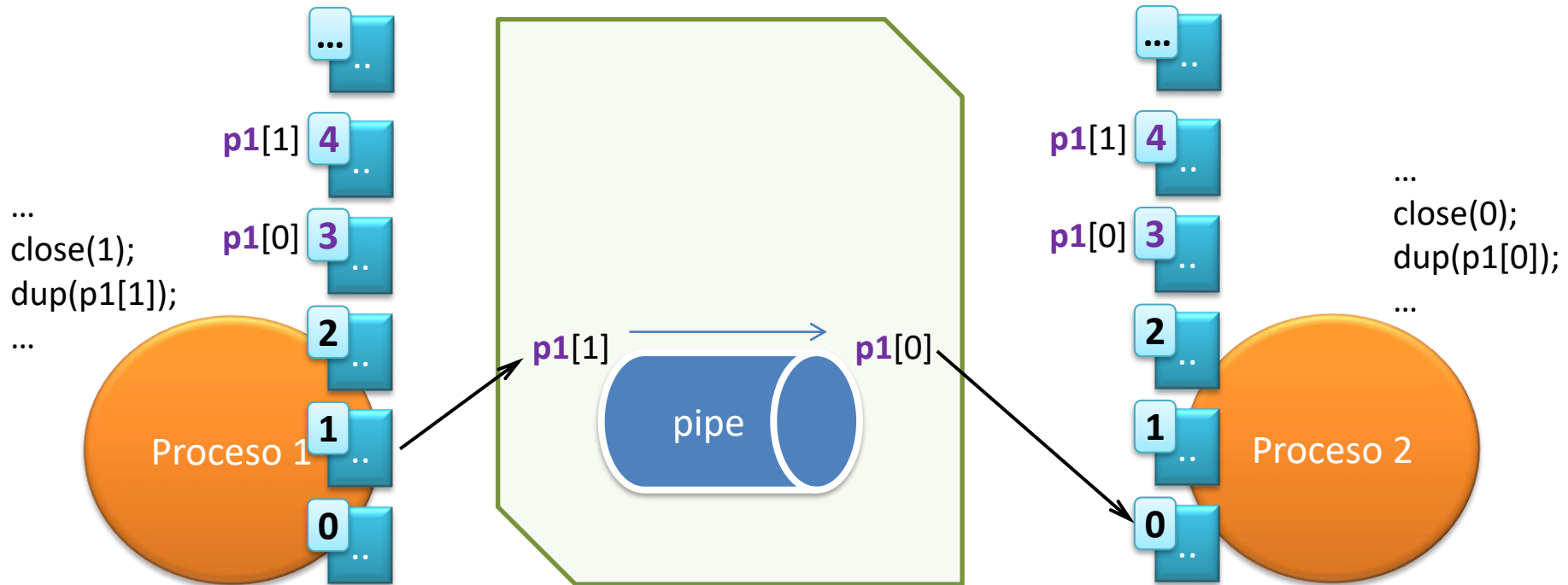
3 redirección



Redirección de la salida estándar en el padre...
Redirección de la entrada estándar en el hijo...

Tubería

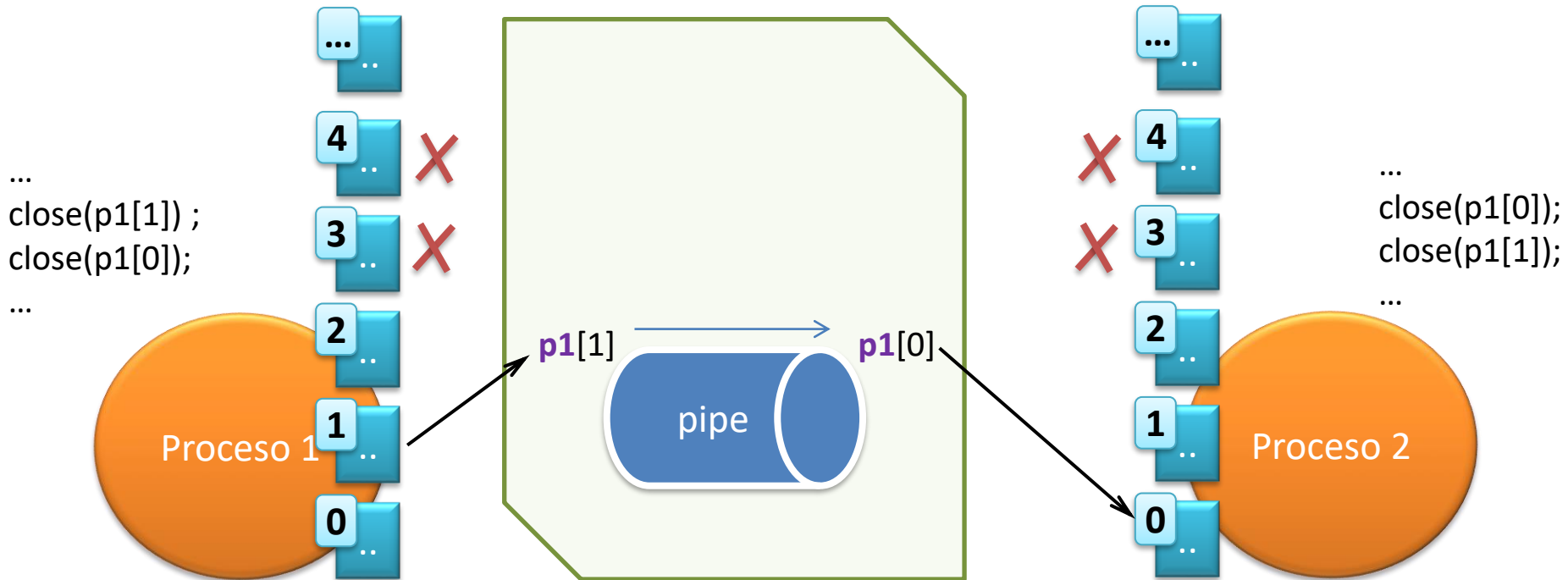
3 redirección



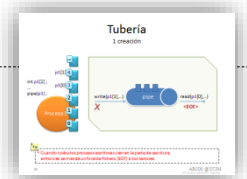
Redirección de la salida estándar en el padre...
Redirección de la entrada estándar en el hijo...

Tubería

4 limpieza



Cierre de los descriptors que no se usan en el padre...
Cierre de los descriptors que no se usan en el hijo...



Tubería

resumen

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

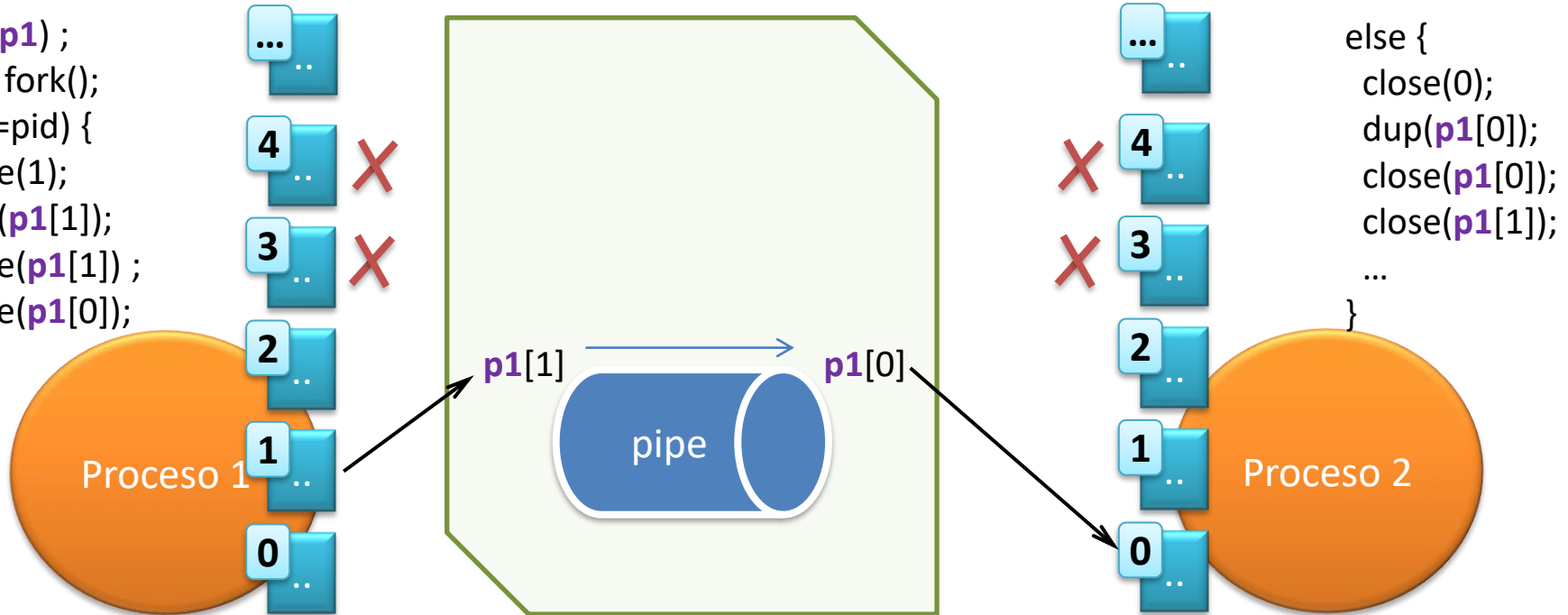
```
...
```

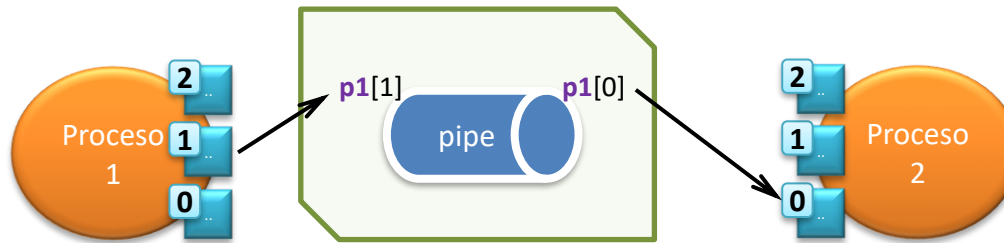
```
pipe(p1);  
pid = fork();
```

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

```
...
```

```
}
```





```

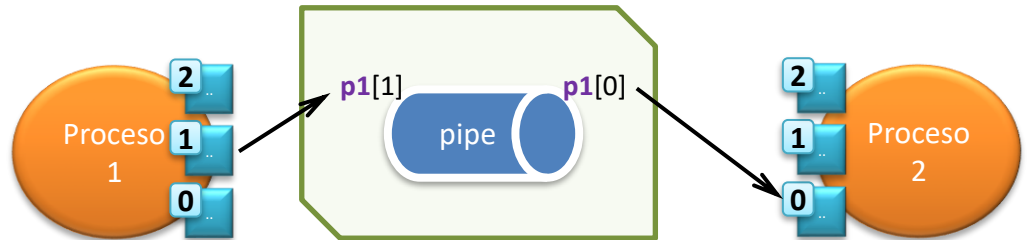
int p1[2] ;
...
pipe(p1) ;
pid = fork();
if (0!=pid) {
    close(1);
    dup(p1[1]);
    close(p1[1]) ;
    close(p1[0]);
    ...
}
else {
    close(0);
    dup(p1[0]);
    close(p1[0]);
    close(p1[1]);
    ...
}

```

- 1) Creación
- 2) fork()
- 3) Redirección (padre)
- 4) Limpieza (padre)
- 3) Redirección (hijo)
- 4) Limpieza (hijo)

Tubería

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: “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]);
        execlp("grep", "grep", "a", NULL);
    } else { /* código del hijo */
        close(STDOUT_FILENO);
        dup(fd[STDOUT_FILENO]);
        close(fd[STDOUT_FILENO]);
        close(fd[STDIN_FILENO]);
        execlp("ls", "ls", NULL);
    }
    return 0;
}
```


Agenda



Linux



Comunicación
con tuberías



Ejercicios

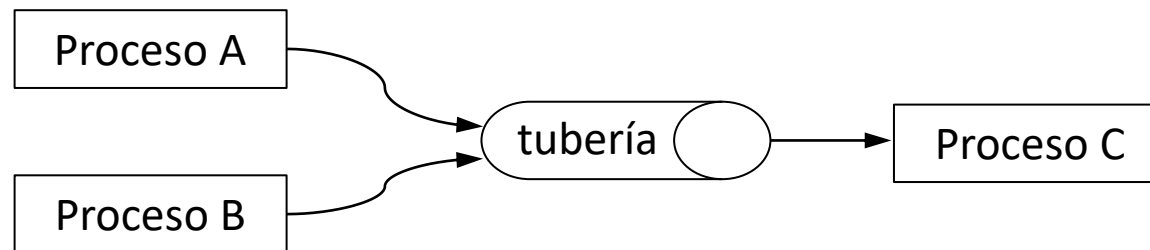




Final del curso 2008-2009

Ejercicio 5 y 6 (3.5 puntos)

- Escribir una función en C sobre UNIX que cree tres procesos comunicados mediante una tubería, de manera que dos de ellos tengan la salida estándar asociada a la tubería y el otro la entrada estándar. *Argumentos: nombres de programa que deberán ejecutar los tres procesos hijos.*





Final del curso 2008-2009

Ejercicio 5 y 6 (3.5 puntos)

```
#include <stdio.h>

int main( void )
{
    int tuberia[2];
    int pid1, pid2;

    /* el proceso padre, que crea el pipe, será el proceso p1 */
    if (pipe(tuberia) < 0) {
        perror("No se puede crear la tubería") ;
        exit(0);
    }

    /* se crea el proceso p2 */
    switch ((pid1=fork()) {
        case -1: perror("Error al crear el proceso") ;
                /* se cierra el pipe */
                close(tuberia[0]) ;
                close(tuberia[1]) ;
                exit(0) ;
                break ;
    })
```



Final del curso 2008-2009

Ejercicio 5 y 6 (3.5 puntos)

```
case 0: /* proceso hijo, proceso B */
    /* se cierra el descriptor de lectura del pipe */
    close(tuberia[0]) ;
    /* aquí iría el código del proceso B */
    /* escribiría usando el descriptor tuberia[1] */
    break ;

default: /* el proceso padre crea ahora el proceso C */
    switch ((pid2 = fork()) {
        case -1: perror("Error al crear el proceso") ;
                close(tuberia[0]) ;
                close(tuberia[1]) ;
                /* se mata al proceso anterior */
                kill(pid1, SIGKILL) ;
                exit(0) ;
        case 0: /* proceso hijo (C) lee de la tubería */
                close(tuberia[1]) ;
                /* código del proceso C que lee de la tubería */
                break ;
        default: /* el proceso padre (B) escribe en la tubería */
                close(tuberia[0]) ;
                /* código del proceso B que escribe en la tubería */
                break ;
    }
}
```

Sistemas Operativos

sesión 12: tuberías

Grado en Ingeniería Informática

Universidad Carlos III de Madrid