LLamadas al Sistema Unix

Sistemas Operativos

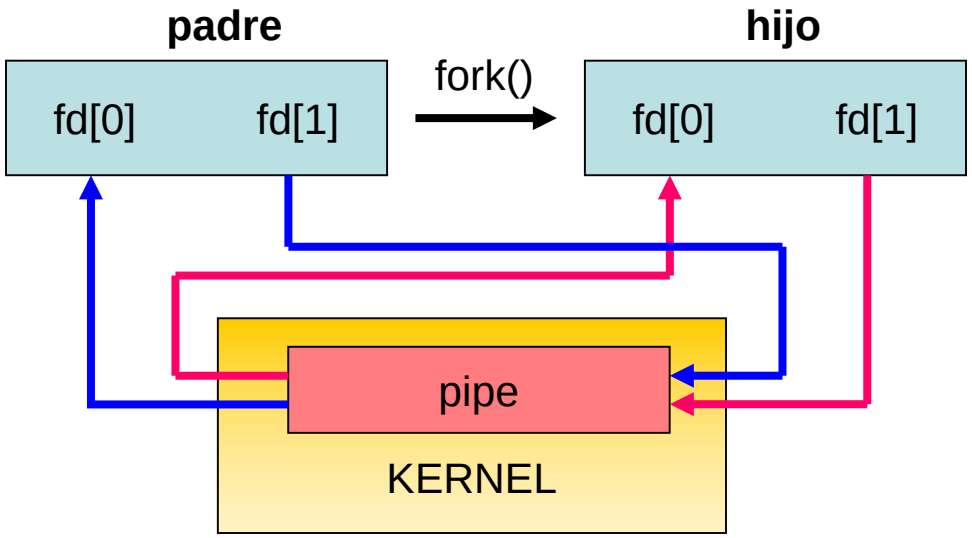
Escuela de Ingeniería Civil Informática

Comunicación Interproceso
pipes()

UNIVERSIDAD DEL BÍO-BÍO

Pipes sin nombre

- Es un canal de comunicación entre procesos emparentados, el cual es referenciado a través de un descriptor de archivo (file descriptor o fd).
- Son equivalentes al operador "|" o "<, >" de la *Shell* de Unix.
- Funciona como una estructura tipo FIFO
- Sólo existe mientras lo usen y desaparece al cerrarlo.



Pipes sin nombre – Crear y Abrir un pipe

- *pipe()* CREA y ABRE un canal de comunicación retornando un arreglo de dos descriptores, uno para lectura y otro para escritura.

- Prototipo:

Archivo cabecera	*#include <unistd.h>*		
Formato	*int pipe(int fd[2]);*		
Salida	Exito	Fallo	Valor en errno
	0	-1	Si

Pipes sin nombre – Escribir en el pipe

- Para ESCRIBIR en un pipe se utiliza la llamada al sistema *write()*, *la misma de los archivos*.

- Prototipo:

Archivo cabecera	`#include <unistd.h>`		
Formato	`ssize_t write(int fd, const void *buff, size_t nbyte);`		
Salida	Exito	Fallo	Valor en errno
	n° de bytes escritos exitosamente	-1	Si

- Si el pipe está lleno *write* se bloquea (espera) hasta que se retiren (se lean) datos suficientes como para poder completar la escritura.

- La capacidad de un PIPE varía según la implementación UNIX pero en ningún caso será menor de 4096 bytes (4KB).

Pipes sin nombre – Leer el pipe

- Para LEER un pipe se utiliza la llamada al sistema *read(), la misma de los archivos*.

- Prototipo:

Archivo cabecera	`#include <unistd.h>`		
Formato	`ssize_t read(int fd, void *buff, size_t nbyte);`		
Salida	Exito	Fallo	Valor en errno
	n° de bytes leídos exitosamente	-1	Si

- Los datos se leen y eliminan en el mismo orden en que se escribieron.
- Usar *read()* sobre un PIPE vacío, bloquea el proceso hasta que otro proceso introduzca datos en el PIPE.
- Si todos los descriptores de escritura del PIPE están cerrados, *read* devuelve cero (fin de archivo).

Pipes sin nombre – Cerrar el pipe

- Para CERRAR los descriptores asociados con un pipe se utiliza la llamada al sistema *close(), la misma de los archivos*.

- Prototipo:

Archivo cabecera	`#include <unistd.h>`		
Formato	`int close(int fd);`		
Salida	Exito	Fallo	Valor en errno
	0	-1	Si

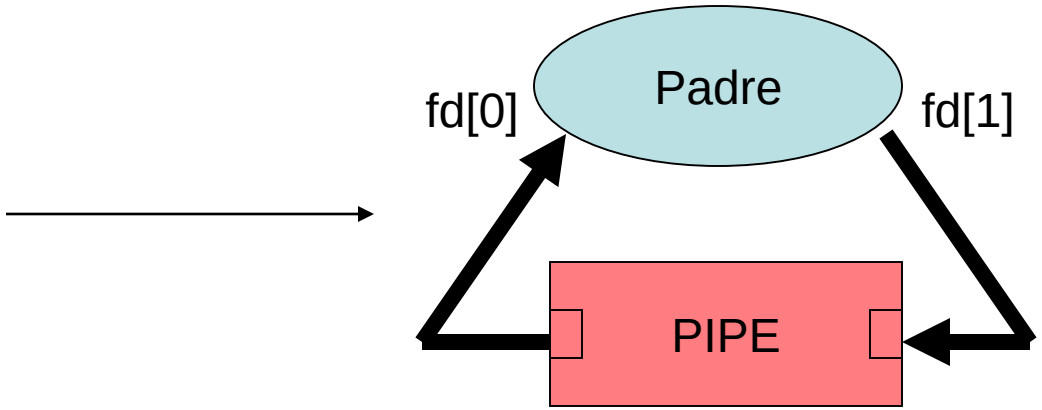
- Cerrando el extremo de escritura se indica la condición de fin de archivo para los procesos lectores del pipe.

- Si se cierra el extremo de lectura, cualquier intento de escritura producirá un error. Se genera una señal SIGPIPE y *errno* tendrá el valor EPIPE al proceso que escribe.

Pipes sin nombre – Ejemplo 1

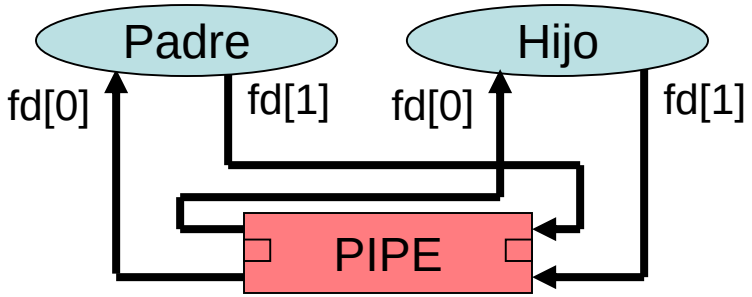
```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

#define BUFSIZE 10

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

  if (argc!=2) {
    printf("Usar: %s mensaje\n", *argv);
    exit(1);
  }
```

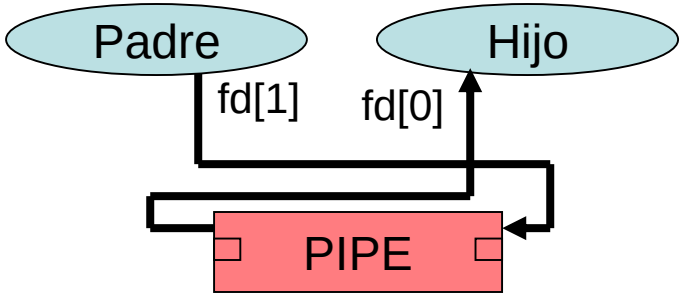
```
if (pipe(fd)==-1) {
  perror("pipe");
  exit(2);
}
```



Pipes sin nombre – Ejemplo 1…

```
switch ( fork() ) {
    case -1:
      perror("fork");
      exit(3);
    case 0: /*hijo*/
      close(fd[1]);
      if (read(fd[0], message, BUFSIZE)!=-1)
        printf("Mensaje recibido por el hijo: [%s]\n", message);
      else
        perror("read");
        exit(4);
      break;
    default: /*padre*/
      close(fd[0]);
      if (write(fd[1], argv[1], strlen(argv[1]))!=-1)
        printf("Mensaje enviado por el padre: [%s]\n", argv[1]);
      else
        perror("write");
        exit(5);
  }
  exit(0);
}
```





Pipes sin nombre

¿Cómo usa los pipes el Shell?

```
$ cat > texto.txt
```

- *cat* toma la entrada desde ***stdin*** y la muestra en ***stdout***.
- Al usar el redireccionamiento “>” la salida se envía a *texto.txt*.
- Para lograr este redireccionamiento en un programa en C, se debe:
  - abrir el archivo para crear un descriptor asociado a el.
  - cambiar el descriptor de salida para *cat* haciendo que ahora apunte a *texto.txt*.

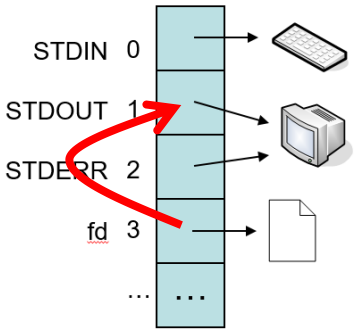
Pipes sin nombre – dup2()

- La llamada al sistema *dup2* permite realizar esta redirección.
- La función *dup2* posee dos argumentos, **fd1** y **fd2** dónde **fd1** es el descriptor que será copiado en **fd2**, antes de esto **fd2** es cerrado.
- Prototipo:

Archivo cabecera	`#include <unistd.h>`		
Formato	`int dup2(int fd1, int fd2);`		
Salida	Exito	Fallo	Valor en errno
	Nuevo descriptor	-1	Si

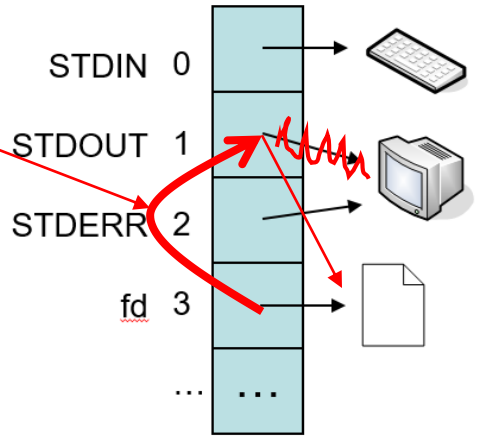
- Ejemplo:

```
dup2(fd, STDOUT);
```



Pipes sin nombre – Ejemplo 2

Se copia el puntero ubicado en la celda 3 a la celda 1




```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h> /*O_WRONLY y O_CREAT*/
#include <unistd.h>

main() {
  int fd;
  mode_t mode= S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
  if ((fd= open("texto.txt", O_WRONLY | O_CREAT, mode)==-1)
    perror("No se puede abrir texto.txt");
  else
    if (dup2(fd, STDOUT_FILENO)==-1)
      perror("No se puede redirigir a la salida estandar");
  /*de aqui en adelante todo se escribe en el archivo*/
  printf("Esta es una prueba de dup2.\n");
  close(fd);
}
```

Pipes sin nombre – Ejemplo 3

¿Cómo implementar el siguiente comando?

```
$ ls - l  |  sort -n +4
```

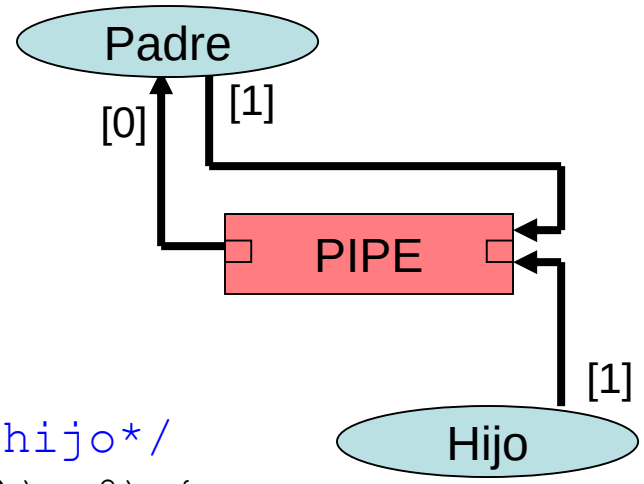
- Este comando permite ordenar de manera numérica (*sort –n*) la salida de *ls –l*, tomando como campo de ordenamiento el cuarto desde el primero (*+4*).

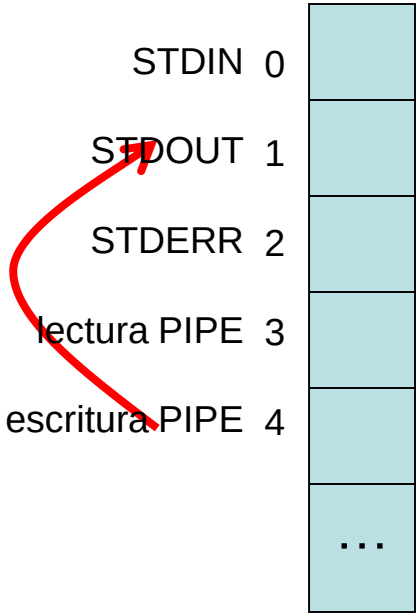
```
$ ls -l | sort -n +4
```

-rw-r--r--	1	Administ	Ninguno	28	Nov 30 19:26 texto.txt
-rwxrwxrwx	1	Administ	Ninguno	576	Nov 30 19:27 dup2_c.c
-rwxrwxrwx	1	Administ	Ninguno	836	Nov 30 17:27 pipe_c.c
-rwxr-xr-x	1	Administ	Ninguno	13630	Nov 30 19:25 dup2_c.o
-rwxr-xr-x	1	Administ	Ninguno	14423	Nov 30 17:27 pipe.o
total 32					
0	1	2	3	4	

Pipes sin nombre – Ejemplo 3 …

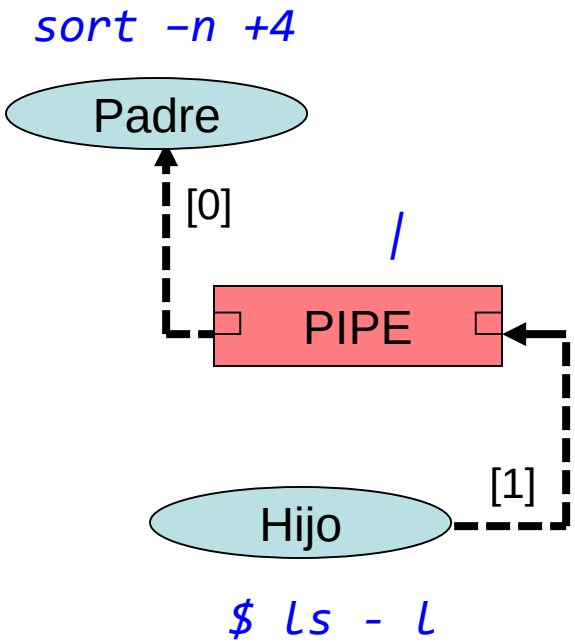
```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

main(void) {
  int fd[2];
  pid_t childpid;
  pipe(fd);                /*hijo*/
  if ((childpid= fork())==0) {
    dup2(fd[1], STDOUT_FILENO);
    close(fd[0]);
    close(fd[1]);
    execl("/usr/bin/ls", "ls", "-l", NULL);
    perror("Fallo en exec de ls");
  }
```

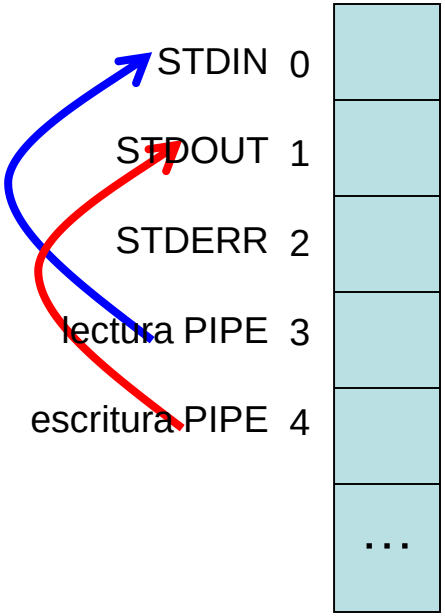




Pipes sin nombre – Ejemplo 3 …

```
else {/*padre*/
  dup2(fd[0], STDIN_FILENO);
  close(fd[0]);
  close(fd[1]);
  execl("/usr/bin/sort", "sort", "-n", "+4",NULL);
  perror("Fallo en exec de sort");
  }
exit(0);
}
```





Pipes sin nombre – Resumen de uso

1. Crear los PIPE'S que se necesiten
2. Generar los procesos hijos
3. Cerrar los descriptores que no se usen
4. Realizar la comunicación
5. Cerrar todos los descriptores que queden abiertos

LIMITACION: sólo se pueden usar entre procesos emparentados.

Pipes con nombre o FIFO´s

- Pueden ser utilizados por procesos no emparentados.
- Los FIFO'S son verdaderos archivos que si existen en el directorio donde sean creados, a través de ellos se realiza la comunicación.
- Para crear un FIFO se utiliza la llamada al sistema *mkfifo()*.

Archivo cabecera	`#include <sys/types.h>` `#include <sys/stat.h>`		
Formato	`int mkfifo(const char *pathname, mode_t mode);`		
Salida	Exito	Fallo	Valor en errno
	0	-1	Si

mode corresponde al modo de apertura: lectura, escritura, etc. Igual que un archivo normal.

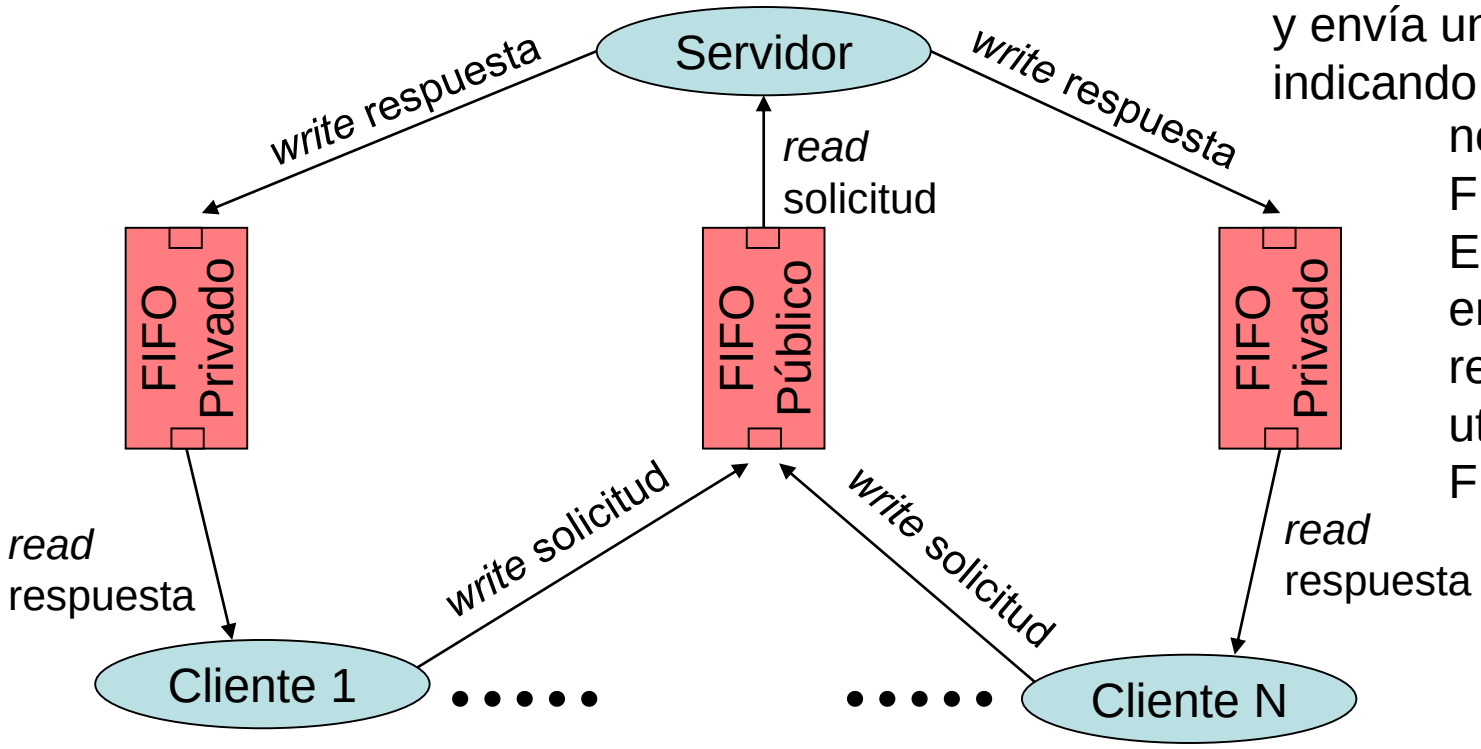
Pipes con nombre o FIFO´s

- Una vez creado el FIFO se debe abrir con *open* y realizar las operaciones de E/S mediante *read()* y *write()*.

- Ejemplo: cliente-servidor usando FIFO'S

Pipes con nombre o FIFO´s

- Esquema para varios clientes y un servidor.



El cliente crea un FIFO y envía una petición, indicando su PID y el nombre del FIFO creado. El servidor envía la respuesta utilizando el FIFO privado.