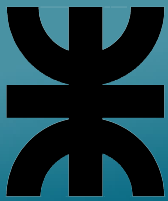




# Tuberías

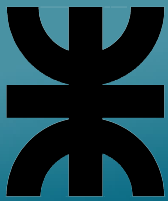


# Comunicación entre procesos

## Comunicación entre procesos o IPC (InterProcess Communication)

Frecuentemente los procesos necesitan comunicarse con otros procesos. Los tres puntos a solucionar:

- Como pasar información de un proceso a otro.
- Como asegurar que dos procesos no se interfieran mientras realizan tareas críticas.
- Secuenciamiento correcto cuando existen dependencias



# Comunicación entre procesos

**Alguno de los métodos de IPC disponibles en Linux:**

- 1- Pipes
- 2- FIFOs (pipes con nombre)
- 3- Colas de mensajes
- 4- Segmentos de memoria compartida
- 5- Comunicación TCP/IP (sockets)

Los pipes, FIFOs y cola de mensaje utilizan llamadas al Kernel.

Los pipes, FIFOs, cola de mensaje y memoria compartida se utilizan entre procesos de una misma PC

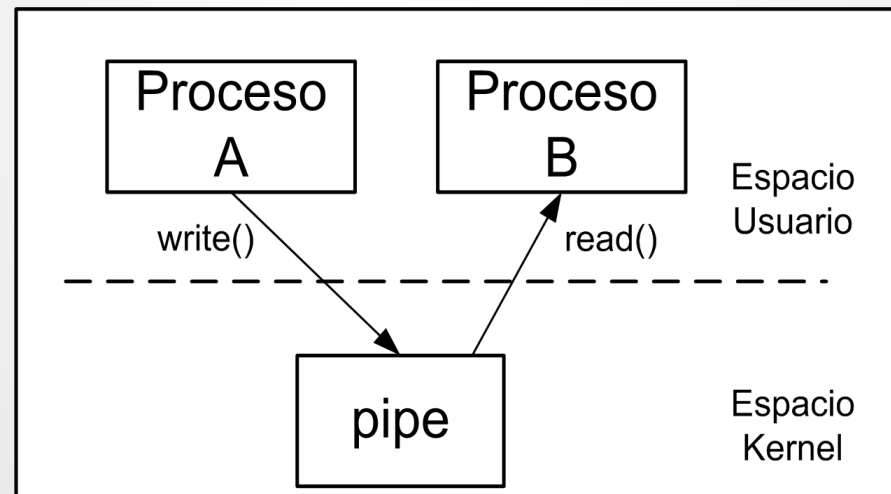


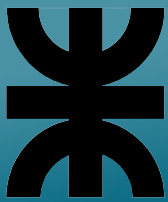
# Tuberías

## PIPE - Tuberías sin nombre

- Usualmente se usa para pasar datos entre procesos relacionados (padre, hijo).
- Buffer mantenido en memoria del Kernel. Capacidad limitada.
- Es tipo FIFO
- Es unidireccional

**Tubería (pipeline,  
pipe )**

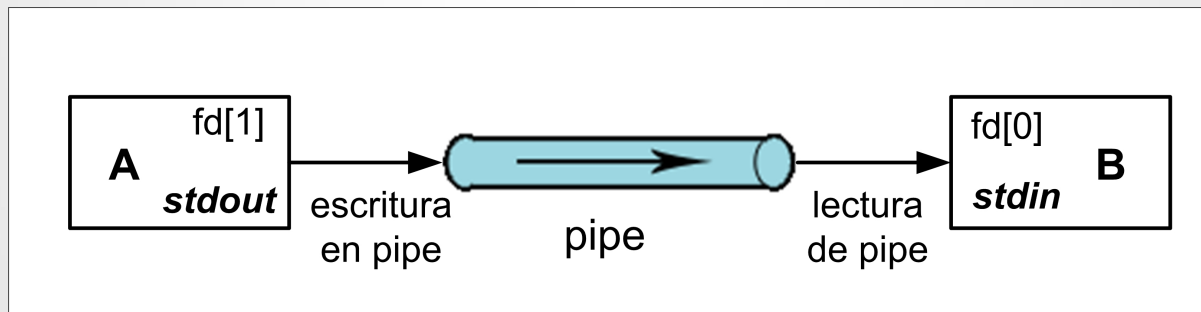


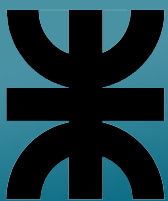


# Tuberías

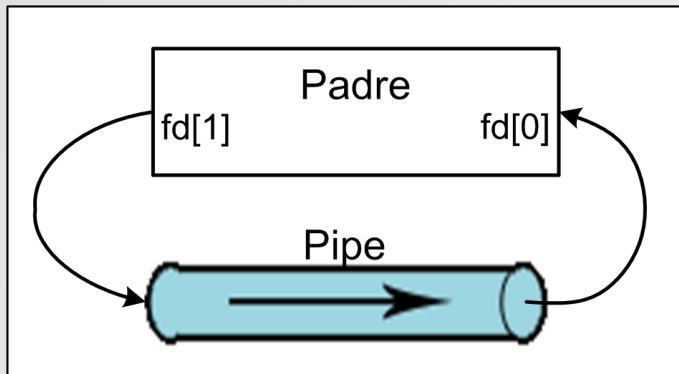
Es un método de conexión que une la salida estándar de un proceso a la entrada estándar de otro. Para esto se utilizan “descriptores de archivos” reservados, los cuales en forma general son:

- 0: entrada estándar (stdin).
- 1: salida estándar (stdout).
- 2: salida de error (stderr).

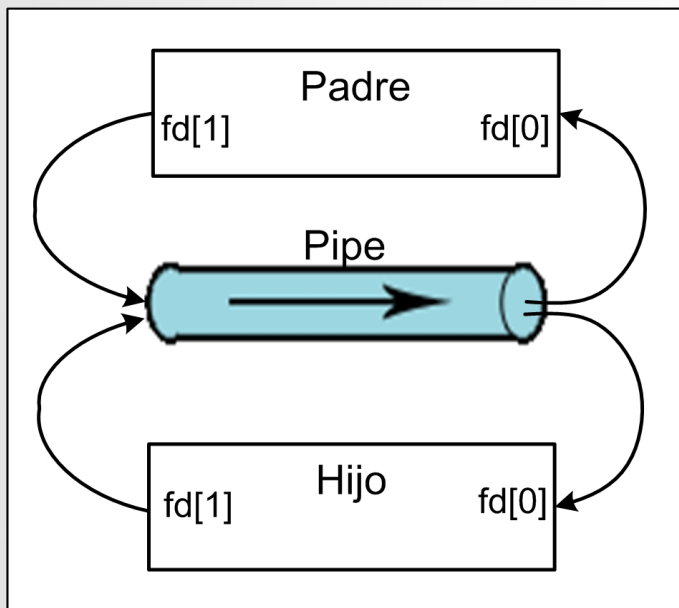




# Tuberías



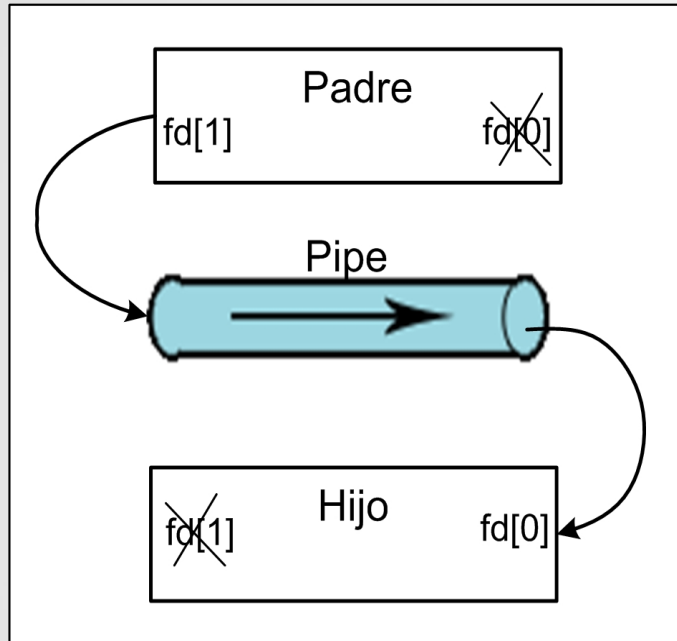
1) Al crear una tubería el proceso se comunica con sí mismo



2) Para que la tubería pueda comunicarse con otro proceso debemos hacer un fork y crear un hijo. Luego de hacer un fork, la tubería se hereda al proceso hijo.



# Tuberías



3) Se recomienda que el padre haga un `close()` de `p[0]` (el lado de lectura de la tubería), y el hijo haga un `close()` de `p[1]` (el lado de escritura de la tubería) o viceversa . Así cerramos el extremo del pipe que no utiliza.



# Tuberías

## Creación de tuberías en C

Para crear una tubería **simple** con C, se usa la llamada al sistema *pipe()*. Toma un argumento que es un arreglo de dos enteros, y si tiene éxito, la tabla contendrá dos nuevos descriptores de archivos para ser usados por la tubería.

```
#include <unistd.h>  
int pipe(int fd)
```

*pipe()* devuelve -1 en caso de error, o 0 si tuvo éxito

Donde *fd* es un arreglo de dos enteros , esos enteros son los descriptores:

*fd*[0] es para leer

*fd*[1] es para escribir





# Tuberías

## Funciones de lectura y escrituras de tuberías

Para **escribir a la entrada de una tubería** se usa la función **write()**

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

```
int write(int fd[1], void *buffer, size_t count);
```

Devuelve el número de bytes escritos o **-1** si hubo error.

Para **leer a la salida de una tubería** se usa la función **read()**

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

```
int read(int fd[0], void *buffer, size_t count);
```

Devuelve el número de bytes leídos, **0** si EOF o **-1** si hubo error.



# Tuberías

Para **cerrar los lados de una tubería** se usa la función **close()**

```
#include <unistd.h>  
int close(int fd);
```

Devuelve 0 si hubo éxito o -1 hubo error.

fd: es alguno de los descriptores (fd[0] es para leer, fd[1] es para escribir)

Si se cierran todos los descriptores de la tubería, esta se destruye.



# Tuberías

## Situaciones conflictivas:

- Un proceso que lee una tubería vacía se bloquea.
- Un proceso que escribe en una tubería llena se bloquea.
- Si dos procesos quieren leer desde una misma tubería no se puede determinar quien leyó primero.
- Un proceso trata de escribir en una tubería en la cual ningún proceso tiene un descriptor de lectura abierto. El kernel envía señal SIGPIPE al proceso, por defecto mata el proceso.



# Tuberías

## Tuberías en línea de comando

Una tubería es una combinación de varios comandos que se ejecutan simultáneamente, donde el resultado del primero se envía a la entrada del siguiente. Esta tarea se realiza por medio del carácter barra vertical pipe “|” .

Este mecanismo es ampliamente usado, en la línea de comandos (shell)

```
$ ls | sort
```

Es un ejemplo de “pipeline”, donde se toma la salida de un comando ls como entrada de un comando sort

la salida por stdout del primer comando (ls: listar) es reenviada por el stdin del segundo (sort: ordenar alfabéticamente)