



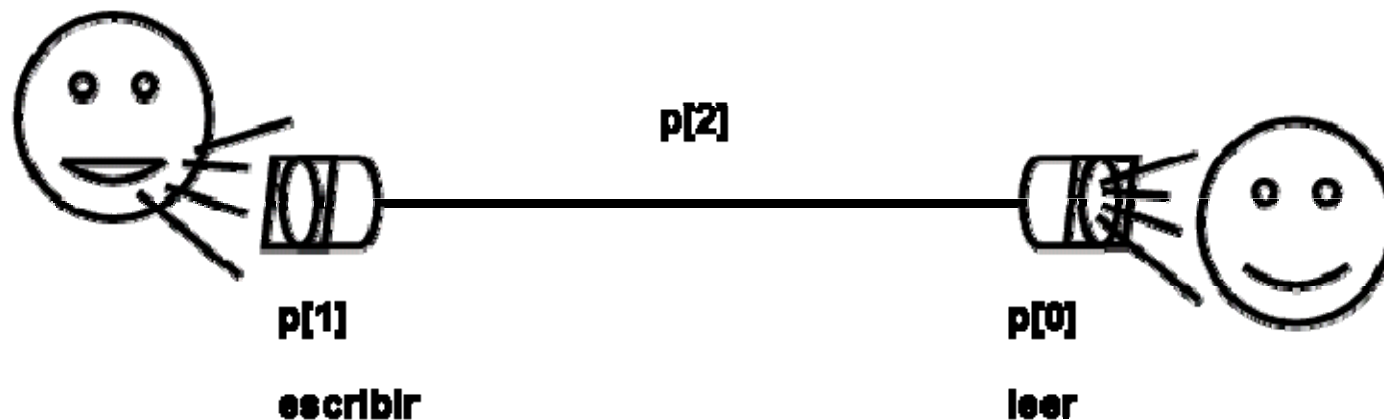
PIPES O TUBERÍAS

Sistemas Operativos 2010-1

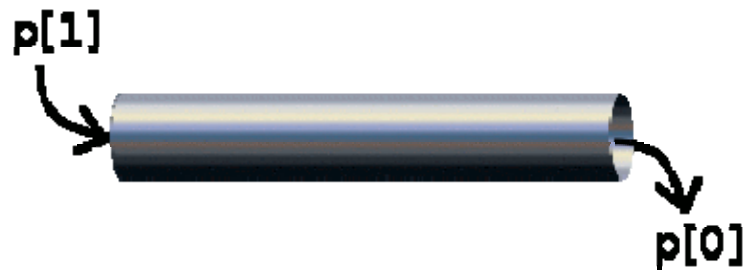
- 
- Las tuberías o “pipes” simplemente conectan la salida estándar de un proceso con la entrada estándar de otro. Normalmente las tuberías son de un solo sentido.
 - Las tuberías suelen ser “half-duplex”, es decir, de un único sentido, y se requieren dos tuberías “half-duplex” para hacer una comunicación en los dos sentidos, es decir “full-duplex”. Las tuberías son, por tanto, flujos unidireccionales de bytes que conectan la salida estándar de un proceso con la entrada estándar de otro proceso.

- 
- Cuando dos procesos están enlazados mediante una tubería, ninguno de ellos es consciente de esta redirección, y actúa como lo haría normalmente. Así pues, cuando el proceso escritor desea escribir en la tubería, utiliza las funciones normales para escribir en la salida estándar. Lo único especial que sucede es que el descriptor de archivo que está utilizando ya no corresponde a la terminal (ya no se escriben cosas a la pantalla).

- Se trata de un archivo especial que ha creado el núcleo. El proceso lector se comporta de forma muy similar: utiliza las llamadas normales para recoger valores de la entrada estándar, solo que ésta ya no se corresponde con el teclado, sino que será el extremo de la tubería.



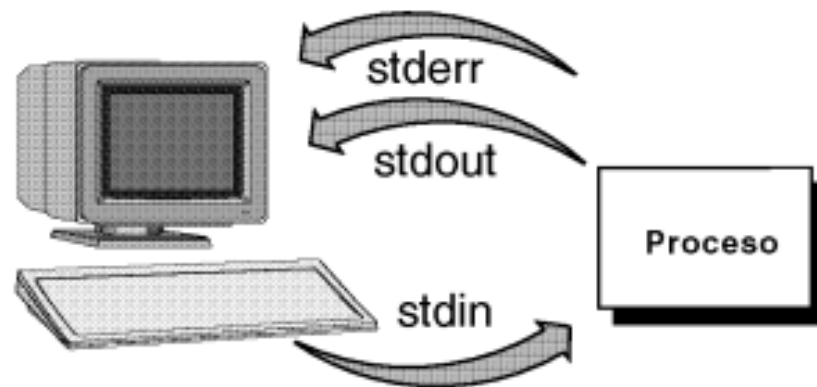
- Los procesos están autorizados a realizar lecturas no bloqueantes de la tubería, es decir, si no hay datos para ser leídos o si la tubería está bloqueada, se devolverá un error. Cuando ambos procesos han terminado con la tubería, el inodo de la tubería es desechado junto con la página de datos compartidos.



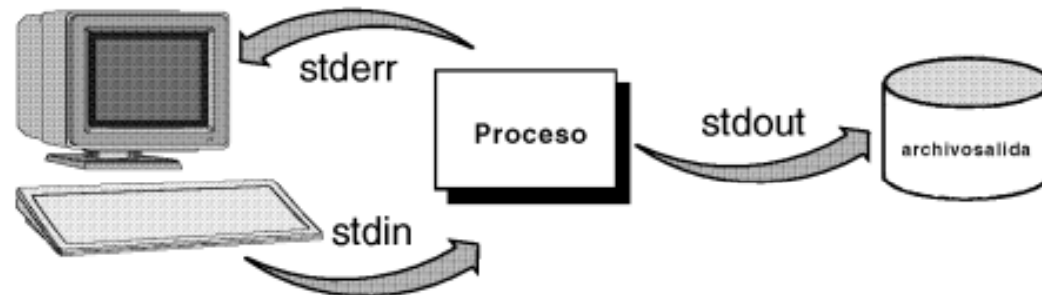
Stdin, stdout, stderr

- Cada proceso abre tres “archivos” estándar: entrada estándar (**stdin**), salida estándar (**stdout**) y error estándar (**stderr**). Los programas utilizan estos archivos del modo siguiente:
- La **entrada estándar** es el lugar desde el cual el programa prevé leer su entrada. Por defecto, los procesos leen el archivo **stdin** desde el teclado.
- La **salida estándar** es el lugar donde el programa graba su salida. Por defecto, los procesos graban el archivo **stdout** en la pantalla del terminal.

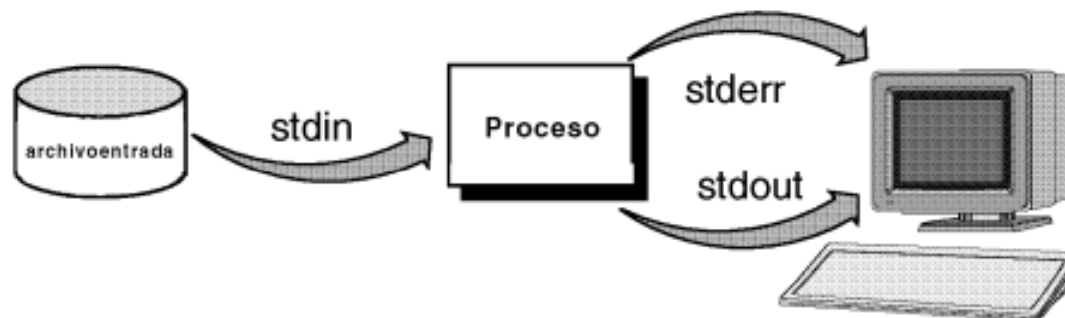
El ***error estándar*** es el lugar donde el programa escribe sus mensajes de error. Por defecto, los procesos graban el archivo **stderr** en la pantalla del terminal.



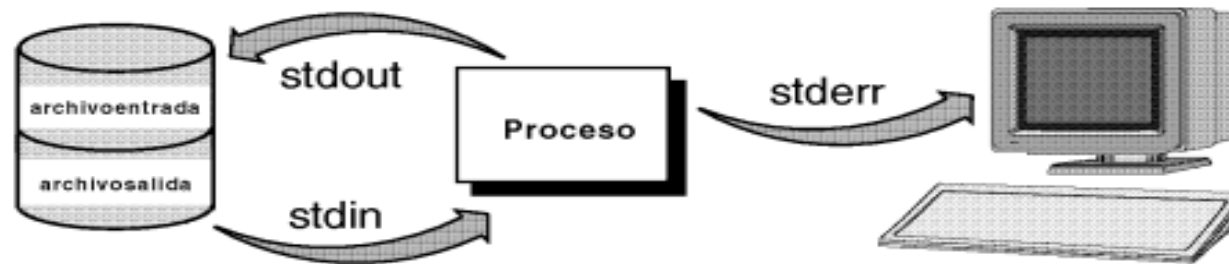
- El shell permite desviar la salida estándar de un proceso desde la pantalla (el valor por defecto) a un archivo. Desviar la salida permite almacenar el texto generado por un comando en un archivo; además, es una manera práctica de seleccionar qué archivos o dispositivos (por ejemplo, impresoras) utiliza un programa.




- El shell permite desviar la entrada estándar de un proceso para que la entrada se lea desde un archivo en lugar de desde el teclado: ***comando < archivoentrada***
- Tendremos un comando cuya entrada se desvía y *archivoentrada* es el nombre del archivo a partir del cual el proceso lee la entrada estándar. El archivo debe existir para que el desvío se lleve a cabo con éxito.





- Puede desviar tanto la entrada estándar como la salida estándar de un comando




- `sort < calcetines > ordenarcalcetines`

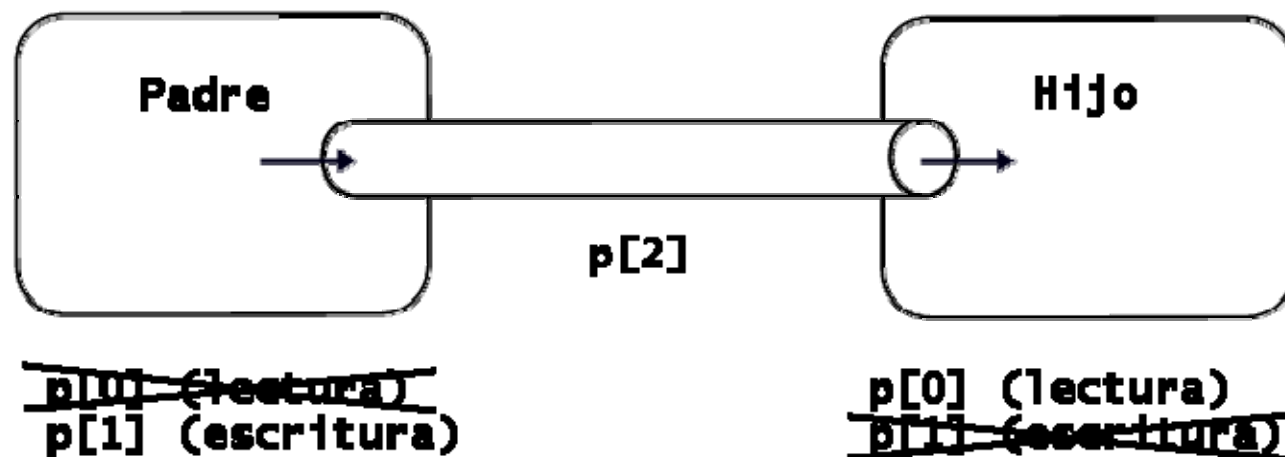
- 
- La utilización de tuberías es muy habitual en cualquier entorno UNIX, por ejemplo:
 - `cat /etc/passwd | grep bash | wc -lines`
 - Los comandos “cat”, “grep” y “wc” se lanzan en paralelo y el primero va “alimentando” al segundo, que posteriormente “alimenta” al tercero. Al final tenemos una salida filtrada por esas dos tuberías. Las tuberías empleadas son destruidas al terminar los procesos que las estaban utilizando.


- 
- Un pipe o tubería se inicia con un arreglo de dos enteros: `int tuberia[2];`
 - Para crear la tubería se emplea la función `pipe()`, que abre dos descriptores de archivo y almacena su valor en los dos enteros que contiene el arreglo de descriptores. El primer descriptor de archivo es abierto como `O_RDONLY`, es decir, sólo puede ser empleado para lecturas.


- 
- El segundo se abre como `O_WRONLY`, limitando su uso a la escritura. De esta manera se asegura que el pipe sea de un solo sentido: por un extremo se escribe y por el otro se lee, pero nunca al revés.
 - `int tuberia[2];`
 - `pipe(tuberia);`

- 
- Una vez creado un pipe, se podrán hacer lecturas y escrituras de manera normal, como si se tratase de cualquier archivo. Sin embargo, no tiene demasiado sentido usar un pipe para uso propio, sino que se suelen utilizar para intercambiar datos con otros procesos.
 - Como ya sabemos, un proceso hijo hereda todos los descriptores de archivos abiertos de su padre, por lo que la comunicación entre el proceso padre y el proceso hijo es bastante cómoda mediante una tubería.

- Para asegurar la unidireccionalidad de la tubería, es necesario que tanto padre como hijo cierren los respectivos descriptores de archivo. El proceso padre deberá cerrar el extremo de lectura de la tubería, mientras que el proceso hijo cierra el extremo de escritura de la misma:



- 
- La tubería “p” se hereda al hacer el `fork()` que da lugar al proceso hijo, pero es necesario 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).
 - Una vez hecho esto, los dos procesos pueden emplear la tubería para comunicarse (siempre unidireccionalmente), haciendo `write()` en `p[1]` y `read()` en `p[0]`, respectivamente.



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

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

```
#include <string.h>
```

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

```
#define SIZE 512
```

```
int main(void)
```

```
{
```

```
    pid_t pid;
```

```
    int p[2], readbytes;
```

```
    char buffer[SIZE];
```

```
    pipe( p );
```

```
    if ( (pid=fork()) == 0 )
```

```
    {
```

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

```
        while( (readbytes=read( p[0], buffer, SIZE )) > 0)
```

```
            write( 1, buffer, readbytes );
```

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

```
    }
```



else

{

close(p[0]);

strcpy(buffer, "Esto llega a través de la
tubería\n");

write(p[1], buffer, strlen(buffer));

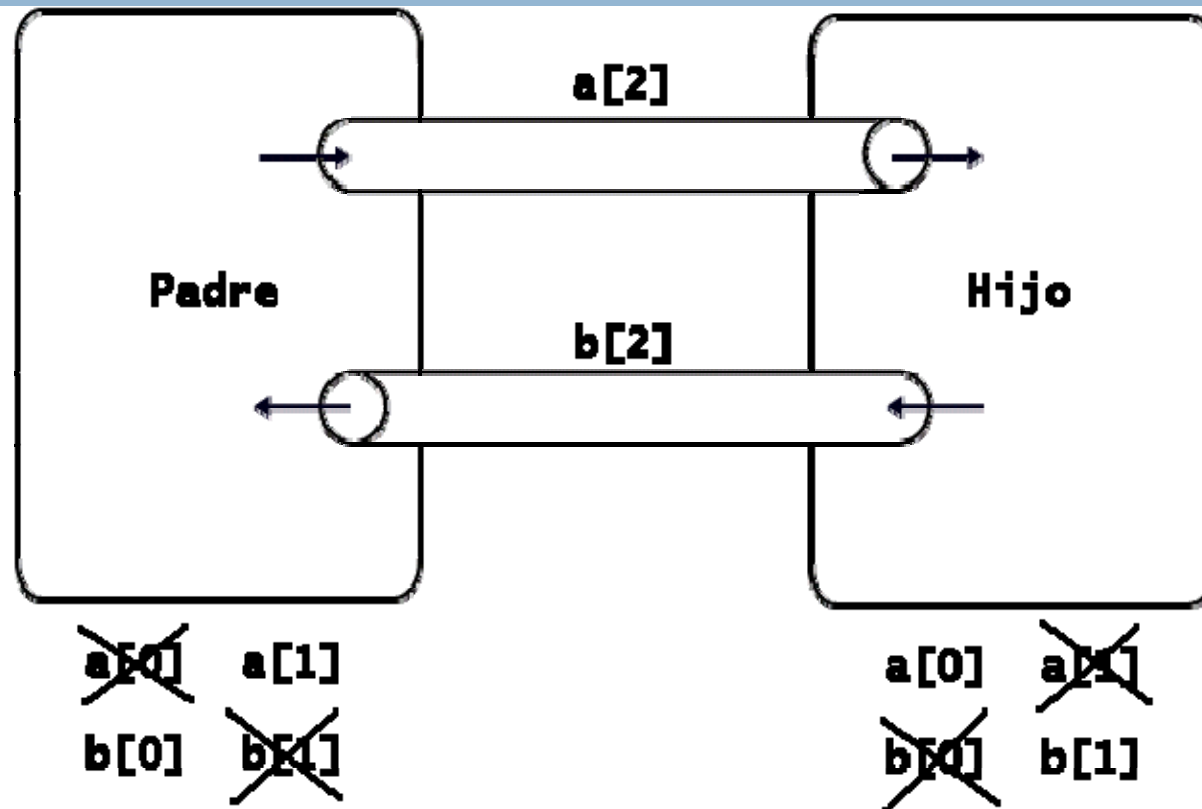
close(p[1]);

}

waitpid(pid, NULL, 0);

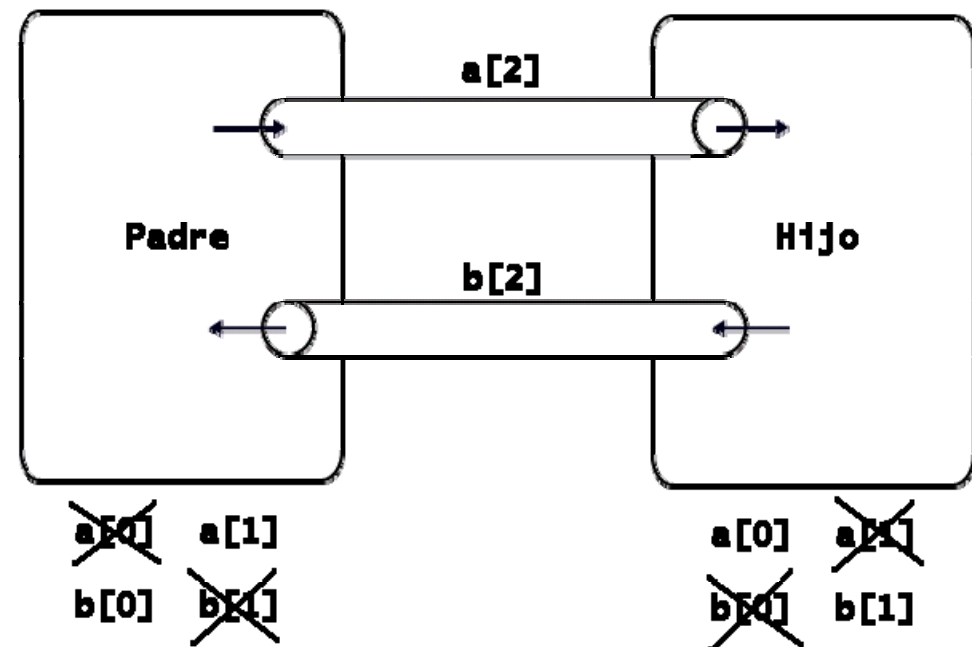
exit(0);

Pipe full duplex



- Para una comunicación full duplex es necesario tener dos tuberías.

- En el padre:
 - el lado de lectura de $a[2]$.
 - el lado de escritura de $b[2]$.
- En el hijo:
 - el lado de escritura de $a[2]$.
 - el lado de lectura de $b[2]$.



```
#include <sys/types.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#define SIZE 512
int main( void )
{
    pid_t pid;
    int a[2], b[2], readbytes;
    char buffer[SIZE];
    pipe( a );
    pipe( b );
    if ( (pid=fork()) == 0 )
    {
        close( a[1] );
        close( b[0] );
        while( (readbytes=read( a[0], buffer, SIZE ) ) > 0)
            write( 1, buffer, readbytes );
    }
}
```

```
close( a[0] );
    strcpy( buffer, "Soy tu hijo hablandote por la otra tuberia.\n" );
    write( b[1], buffer, strlen( buffer ) );
    close( b[1] );
}
else
{
    close( a[0] );
    close( b[1] );
    strcpy( buffer, "Soy tu padre hablandote por una tuberia.\n" );
    write( a[1], buffer, strlen( buffer ) );
    close( a[1]);
    while( (readbytes=read( b[0], buffer, SIZE )) > 0)
        write( 1, buffer, readbytes );
    close( b[0]);
}
waitpid( pid, NULL, 0 );
exit( 0 );
}
```

Ejercicio

```
$ who | tee guardarquién | wc -l
```

```
4
```

```
$ more guardarquién
```

luis	consola	Sep 7 08:50
teo	tty01	Sep 7 11:57
quique	tty02	Sep 7 08:13

