Programación servicios y procesos

Comunicación entre procesos

Comunicación entre procesos

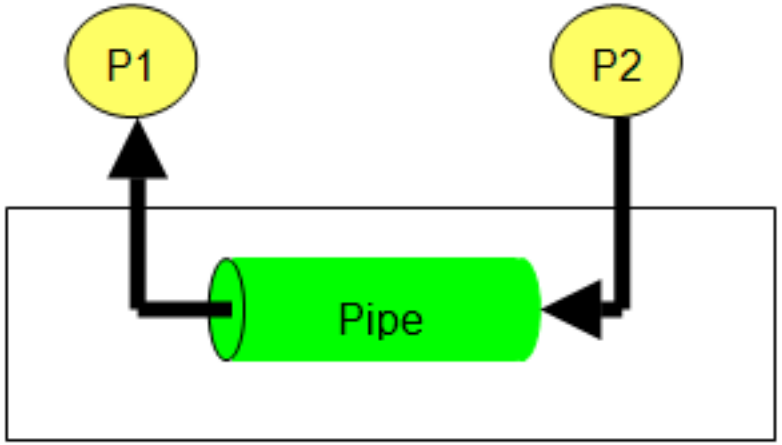
Existen varias formas para comunicar dos procesos en el sistema

- PIPES
- COLAS
- SEMÁFOROS
- Memória compartida…

PIPES

Falso fichero.

Proceso escribe … otro lee



Utilizar un PIPE

Creación

#include <unisted.h>

int pipe (int fd[2])

La función recibe un array de dos elementos (uno para lectura otro para escritura)

De forma normal:

- fd[0] Contiene descriptor lectura.
- fd[1] Contiene descriptor escritura.

Utilizar un PIPE

Escritura y lectura

*int read (int fd, void *buf,int count);*

*int write(int fd, void *buf,int count);*

Veremos mejor su utilización mediante un ejemplo...

Ejemplo PIPE entre padre e hijo

Vemos primera parte programa.

```c
// programa padre e hijo se comunican por pipe

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

int main()
{
int fd[2]; //creo vector entero
char buffer[30];
pid_t pid;

pipe(fd); //se crea el pipe

pid=fork(); //creo proceso hijo
```

Ejemplo PIPE entre padre e hijo

```
switch(pid)
    {
    case -1: //error
        printf("no se ha podido crear el hijo\n");
        exit(-1);
        break;

    case 0: //hijo escribe descriptor 1
        printf("el hijo escribe en el pipe...\n");
        write (fd[1], "Hola papi" ,10);
        break;

    default: //padre lee de descriptor 0
        wait(NULL); //padre espera hijo termine escribir
        printf("el padre lee del pipe...\n");
        read (fd[0], buffer ,10);
        printf ("\tMensaje leido:%s\n",buffer);
        break;

    } //fin switch


}
```

Problema: Comunicación bidireccional

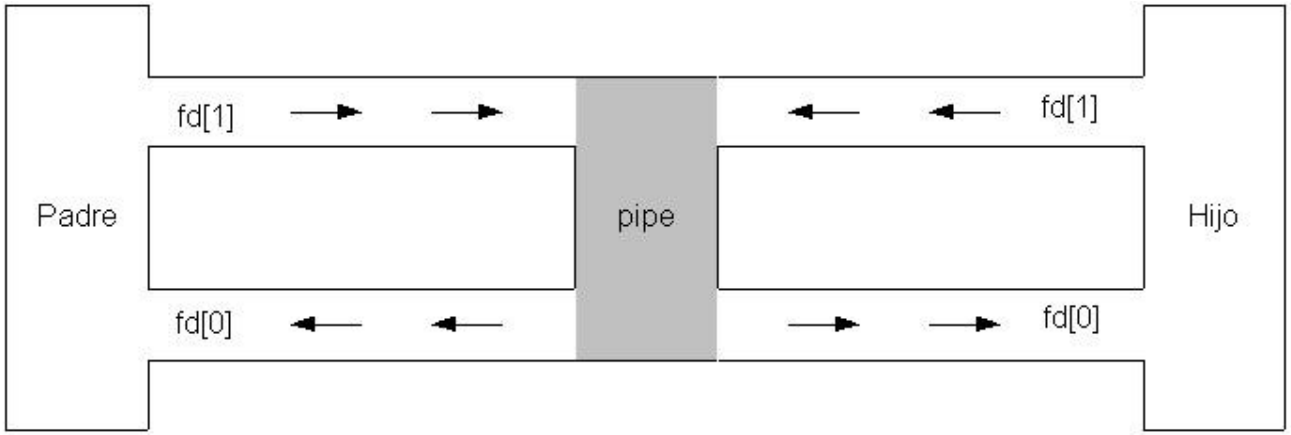
Al ser la comunicación en una única dirección debemos decidir en que dirección (de padre a hijo o de hijo a padre).

Descriptores se comparten entre procesos padres e hijos→ Debemos cerrar extremo no nos interesa.

Comunicación bidireccional

Flujo información de **padre a hijo**

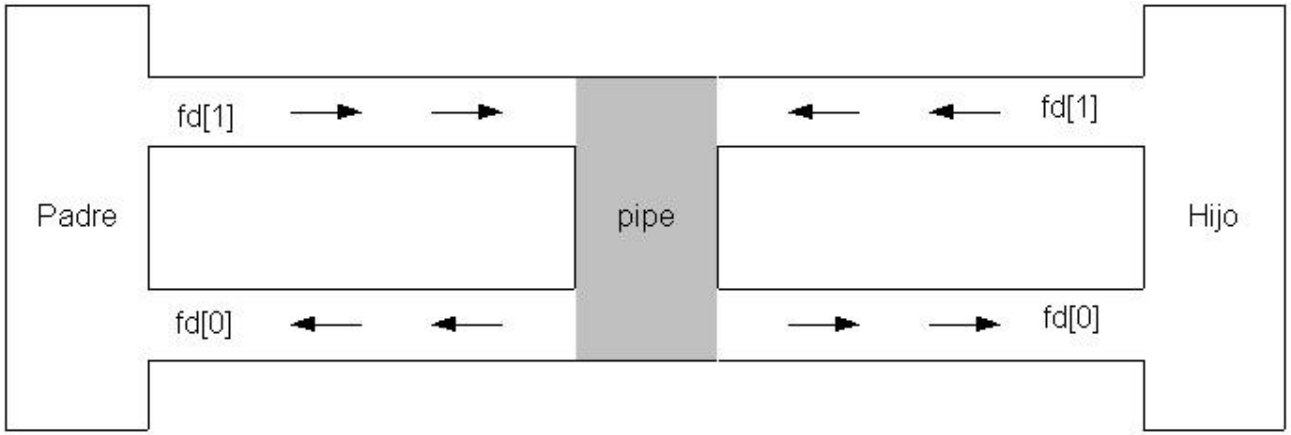
- El padre debe cerrar el descriptor de lectura fd[0] (no va a usar)
- El hijo debe cerrar el descriptor de escritura fd[1] (no va a usar)



Comunicación bidireccional

Flujo información de **hijo a padre**

- El padre debe cerrar el descriptor de lectura fd[1] (no va a usar)
- El hijo debe cerrar el descriptor de escritura fd[0] (no va a usar)



Código comunicación padre a hijo

Primera parte igual que la anterior...

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

int main()
{
int fd[2];
pid_t pid;
char saludoPadre[]="Buenos dias hijo.\0";
char buffer[80];

pipe(fd); //creo pipe
pid=fork(); //creo proceso
```

Código comunicación padre a hijo

Observa como respeta las normas anteriores

```
switch(pid) {
case -1:  //ERROR
    printf ("No se ha podido crear hijo...");
    exit(-1);
case 0: //hijo recibe
    close (fd[1]); //cierra el descriptor de entrada
    read (fd[0],buffer, sizeof(buffer)); //leo en el pipe
    printf ("\El hijo recibe algo del pipe.%s\n",buffer);
    break;
default: //padre envia
    close (fd[0]); //cierra el descriptor de entrada
    write (fd[1],saludoPadre, strlen(saludoPadre)); //escribo en el pipe
    printf ("\El hijo envia mensaje al hijo\n",buffer);
    wait(NULL); //espero al proceso hijo
    break;
}
return 0;
}
```

PIPES con nombres : FIFOS

Los FIFOS permiten comunicar procesos que no tienen que ser padre e hijo.

Un FIFO es como un fichero con nombre que existe en el sistema de ficheros y que pueden abrir, leer y escribir múltiples procesos.

Los datos escritos se leen como en una cola, y una vez leídos no pueden ser leídos de nuevo.

Características FIFO

- Una operación de escritura en un FIFO queda en espera hasta que el proceso pertinente abra el FIFO para iniciar la lectura.

- Solo se permite la escritura de información cuando un proceso vaya a recoger dicha información.

Usos FIFO: Función mknod

Se puede ejecutar desde terminal:

En terminal >mknod FIFO1 p //creo FIFO

> cat FIFO1 // en espera hasta entre algo..

En otro terminal

>ls>FIFO1

.. se ejecuta el cat .

```
david@david-OEM ~/pss $ mknod FIFO1 p
david@david-OEM ~/pss $ cat FIFO1
actividad2
actividad2.c
doshijos
doshijos.c
ejemplo_fork_pipe
ejemplo_fork_pipe.c
ejemplopipe
ejemplopipe1.c
ejemplopipe3
ejemplopipe3.c
ejemwriteread
ejemwriteread.c
entregable3_1
entregable3_1.c
execv
execv.c
FIFO1
fork
fork2.c
```

mknod en C

Función:

mknod("FIFO1", S_IFIFO|0644 , 0);

- "FIFO1" se crea como archivo FIFO.
- S_IFIFO: Modo de creación crea un FIFO (con S_IFIFO) y modo permisos 644.
- 0 (tercer argumento) Se ignora (al crear un FIFO)

Ejemplo. Crea y lee cadena de FIFO

Crea el FIFO, lo abre y lee cadena (si la contiene).

```
mknod("FIFO1",S_IFIFO|0660,0);

while(TRUE)
    {
      fp=open("FIFO1",O_RDONLY);
      nbytes=read(fp,buffer,TAM_BUF-1);
      buffer[nbytes]='\0';
      printf("Cadena recibida: %s \n",buffer);
      close(fp);
    }
```

Ejemplo. Escribe en FIFO

Abre archivo para escritura, escribe y cierra

```
int main()
{
  int fp;
  char saludo[]="Un saludo!!!";
  fp=open("FIFO1",1);

if (fp==-1)
    {
    printf ("ERROR AL ABRIR ARCHIVO\n");
    exit(1);
    }
printf ("mandando información al fifo...\n");
write (fp,saludo,strlen(saludo));
close(fp);
```