Práctica 2.4: Tuberías

Objetivos

Las tuberías ofrecen un mecanismo sencillo y efectivo para la comunicación entre procesos en un mismo sistema. En esta práctica veremos los comandos e interfaz para la gestión de tuberías, y los patrones de comunicación típicos.

Contenidos

Preparación del entorno para la práctica

Esta práctica únicamente requiere las herramientas y entorno de desarrollo de usuario.

Tuberías sin nombre

Las tuberías sin nombre son entidades gestionadas directamente por el núcleo del sistema y son un mecanismo de comunicación unidireccional eficiente para procesos relacionados (padre-hijo). La forma de compartir los identificadores de la tubería es por herencia (en la llamada `fork(2)`).

Ejercicio 1. Escribir un programa que emule el comportamiento de la shell en la ejecución de una sentencia en la forma: `comando1 argumento1 | comando2 argumento2`. El programa creará una tubería sin nombre y creará un hijo:

- El proceso padre redireccionará la salida estándar al extremo de escritura de la tubería y ejecutará `comando1 argumento1`.
- El proceso hijo redireccionará la entrada estándar al extremo de lectura de la tubería y ejecutará `comando2 argumento2`.

Probar el funcionamiento con una sentencia similar a: `./ejercicio1 echo 12345 wc -c`

***Nota*:** Antes de ejecutar el comando correspondiente, deben cerrarse todos los descriptores no necesarios.

FICHERO:

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

int main(int argc, char ** argv){
   if(argc<5){
      printf("Error. Introducir comandos y argumentos\n");
      return -1;
   }
   int tuberia[2];
```

```
  pipe(tuberia);

  int pid = fork();
  switch(pid){
    case -1:
      perror("Error");
      return 1;
      break;
    case 0: //Hijo
      dup2(tuberia[0],0); //redireccionar entrada estandard a lectura de tuberia.
      close(tuberia[1]); //Se cierra escritura para leer
      close(tuberia[0]); //Se cierra lo que has utilizado
      execlp(argv[3],argv[3],argv[4],NULLL);//ejecutar segundo comando con su argumento
      return 0;
      break;
    default: //Padre
      dup2(tuberia[1],1);
      close(tuberia[0]); //Se cierra escritura para leer
      close(tuberia[1]); //Se cierra lo que has utilizado
      execlp(argv[1],argv[1],argv[2],NULLL);
      return 0;
      break;
  }
  return 0;

}
```

TERMINAL:

```
[cursoredes@localhost ~]$ g++ practica4.cpp -o ej1
[cursoredes@localhost ~]$ ./ej1 echo 12345 wc -c
6
```

Ejercicio 2. Para la comunicación bi-direccional, es necesario crear dos tuberías, una para cada sentido: `p_h` y `h_p`. Escribir un programa que implemente el mecanismo de sincronización de parada y espera:

- El padre leerá de la entrada estándar (terminal) y enviará el mensaje al proceso hijo, escribiéndolo en la tubería `p_h`. Entonces permanecerá bloqueado esperando la confirmación por parte del hijo en la otra tubería, `h_p`.
- El hijo leerá de la tubería `p_h`, escribirá el mensaje por la salida estándar y esperará 1 segundo. Entonces, enviará el carácter `'1'` al proceso padre, escribiéndolo en la tubería `h_p`, para indicar que está listo. Después de 10 mensajes enviará el carácter `'q'` para indicar al padre que finalice.

FICHERO:

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


int main(int argc, char ** argv){
```

```
   int tuberia_ph[2];
   int tuberia_hp[2];
   int rc = 0;
   int m = 0;
   int fin = 0;
   char buffer[256];
   pipe(tuberia_ph);//Crear la tuberia padre->hijo
   pipe(tuberia_hp);//Crear la tuberia hijo->padre

   int pid = fork();
   switch(pid){
      case -1:
         perror("Error");
         return 1;
         break;
      case 0: //Hijo
         close(tuberia_ph[1]); //Se cierra escritura para leer
         close(tuberia_hp[0]); // Se cierra lectura para escribir
         while(!fin){
            rc = read(tuberia_ph[0], buffer, 256);//Leer de ese extremo de tuberia. El hijo se
queda bloqueado aqui hasta que llega string.
            buffer[rc] = '\0'; //Añadir manualmente final de cadena
            printf("Mensaje recibido: %s\n", buffer);
            sleep(1);
            if(++m == 10){
               write(tuberia_hp[1], "Q", 1);
               fin = 1;
            }
            else{
               write(tuberia_hp[1], "L", 1);
            }
         }
         close(tuberia_ph[0]); //Se cierra lo que has utilizado
         close(tuberia_hp[1]);
         return 0;
         break;
      default: //Padre
         close(tuberia_ph[0]); //Se cierra escritura para leer
         close(tuberia_hp[1]); // Se cierra lectura para escribir
         while(!fin){
            printf("? ");
            scanf("%s", buffer);
            write(tuberia_ph[1], buffer, strlen(buffer) + 1);
            read(tuberia_hp[0], buffer, 1);
            if(buffer[0] == 'Q'){
               fin = 1;
            }
         }
         close(tuberia_ph[1]); //Se cierra lo que has utilizado
         close(tuberia_hp[0]);
         return 0;
         break;
   }
   break;

}
```

TERMINAL:
[cursoredes@localhost ~]$ ./ej2
? h
Mensaje recibido: h
? o
Mensaje recibido: o
lecho
? Mensaje recibido: l
a
? Mensaje recibido: a
? aaaa
Mensaje recibido: aaaa
? bbbbb
Mensaje recibido: bbbbb
? fsfedr
Mensaje recibido: fsfedr
? fsddssrf
Mensaje recibido: fsddssrf
? eferfer
Mensaje recibido: eferfer
? ertretr
Mensaje recibido: ertretr

Tuberías con nombre

Las tuberías con nombre son un mecanismo de comunicación unidireccional, con acceso de tipo FIFO, útil para procesos sin relación de parentesco. La gestión de las tuberías con nombre es igual a la de un archivo ordinario (open, `write`, `read`…). Revisar la información en `fifo(7)`.

Ejercicio 3. Usar la orden `mkfifo` para crear una tubería con nombre. Usar las herramientas del sistema de ficheros (`stat`, `ls`…) para determinar sus propiedades. Comprobar su funcionamiento usando utilidades para escribir y leer de ficheros (ej. echo, `cat`, `tee`...).

TERMINAL 1:
[cursoredes@localhost ~]$ mkfifo tuberia3
[cursoredes@localhost ~]$ ls -l
total 136
lrwxrwxrwx 1 cursoredes cursoredes 11 Nov 10 19:32 aaaa -> directorioP
drwxr-xr-x 2 cursoredes cursoredes 116 Sep 9 2018 Desktop
drwxrwxr-x 2 cursoredes cursoredes   6 Nov 10 19:32 directorioP
drwxr-xr-x 2 cursoredes cursoredes   6 Sep 9 2018 Documents
drwxr-xr-x 2 cursoredes cursoredes   6 Sep 9 2018 Downloads
-rwxrwxr-x 1 cursoredes cursoredes 8752 Nov 29 18:30 ej1
-rwxrwxr-x 1 cursoredes cursoredes 8784 Nov 23 17:24 ej11
-rwxrwxr-x 1 cursoredes cursoredes 8760 Nov 23 18:06 ej12
-rwxrwxr-x 1 cursoredes cursoredes 8976 Nov 10 20:43 ej15
-rwxrwxr-x 1 cursoredes cursoredes 13176 Nov 11 00:20 ej17
-rwxrwxr-x 1 cursoredes cursoredes 8904 Nov 29 12:55 ej2
-rwxrwxr-x 1 cursoredes cursoredes 8496 Nov 23 12:14 ej7
-rwxrwxr-x 1 cursoredes cursoredes 8856 Nov 23 12:33 ej8

```
-rwxrwxr-x 1 cursoredes cursoredes  8784 Nov 10 19:25 ej9
-rwxrwxr-x 1 cursoredes cursoredes     0 Nov 10 20:43 ficheroPrueba
drwxr-xr-x 2 cursoredes cursoredes     6 Sep  9  2018 Music
drwxr-xr-x 2 cursoredes cursoredes   147 Sep 22  2018 Pictures
-rw-rw-r-- 2 cursoredes cursoredes  2037 Nov 11 00:20 practica2.cpp
-rw-rw-r-- 2 cursoredes cursoredes  2037 Nov 11 00:20 practica2.cpp.hard
lrwxrwxrwx 1 cursoredes cursoredes    13 Nov 10 19:55 practica2.cpp.sym -> practica2.cpp
-rw-rw-r-- 1 cursoredes cursoredes  1036 Nov 23 18:15 practica3.cpp
-rw-rw-r-- 1 cursoredes cursoredes  1051 Nov 29 18:29 practica4.cpp
drwxr-xr-x 2 cursoredes cursoredes     6 Sep  9  2018 Public
-rwxrwxr-x 1 cursoredes cursoredes    52 Nov 10 20:16 salida2.txt
-rwxrwxr-x 1 cursoredes cursoredes    45 Nov 10 20:07 salida.txt
drwxr-xr-x 2 cursoredes cursoredes     6 Sep  9  2018 Templates
prw-rw-r-- 1 cursoredes cursoredes     0 Nov 29 18:32 tuberia3
drwxr-xr-x 2 cursoredes cursoredes     6 Sep  9  2018 Videos
[cursoredes@localhost ~]$ echo 12345 > tuberia3
[cursoredes@localhost ~]$
```

TERMINAL 2:

```
[cursoredes@localhost ~]$ cat tuberia3
12345
```

Ejercicio 4. Escribir un programa que abra la tubería con el nombre anterior en modo sólo escritura, y escriba en ella el primer argumento del programa. En otro terminal, leer de la tubería usando un comando adecuado.

FICHERO:

```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char ** argv){
   if(argc<2){
      printf("Error. Introducir argumento\n");
      return -1;
   }
            int fd = open("tuberia3", O_WRONLY);
            write(fd, argv[1], strlen(argv[1]));
            close(fd);
   return 0;

}
```

TERMINAL 1:

```
[cursoredes@localhost ~]$ g++ practica4.cpp -o ej4
[cursoredes@localhost ~]$ ./ej4 blabla
```

TERMINAL 2:
[cursoredes@localhost ~]$ cat tuberia3
blabla

Multiplexación síncrona de entrada/salida

Es habitual que un proceso lea o escriba de diferentes flujos. La llamada `select(2)` permite multiplexar las diferentes operaciones de E/S sobre múltiples flujos.

Ejercicio 5. Crear otra tubería con nombre. Escribir un programa que espere hasta que haya datos listos para leer en alguna de ellas. El programa debe mostrar la tubería desde la que leyó y los datos leídos. Consideraciones:

- Para optimizar las operaciones de lectura usar un *buffer* (ej. de 256 bytes).
- Usar `read(2)` para leer de la tubería y gestionar adecuadamente la longitud de los datos leídos.
- Normalmente, la apertura de la tubería para lectura se bloqueará hasta que se abra para escritura (ej. con `echo 1 > tuberia`). Para evitarlo, usar la opción `O_NONBLOCK` en `open(2)`.
- Cuando el escritor termina y cierra la tubería, `read(2)` devolverá 0, indicando el fin de fichero, por lo que hay que cerrar la tubería y volver a abrirla. Si no, `select(2)` considerará el descriptor siempre listo para lectura y no se bloqueará.

FICHERO:

```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/select.h>


int main(int argc, char ** argv){
   mkfifo("tuberia5", 0777);
   char buffer[256];
           int fd1 = open("tuberia3", O_RDONLY|O_NONBLOCK);
   int fd2 = open("tuberia5", O_RDONLY|O_NONBLOCK);

   int cambios  = 0;
           while(cambios != -1){
      fd_set rfds;
      FD_ZERO(&rfds);
      FD_SET(fd1, &rfds);
      FD_SET(fd2, &rfds);
      int n = (fd1 < fd2) ? fd2 + 1: fd1 + 1;
      cambios = select(n, &rfds, NULL, NULL, NULL);
      if(cambios > 0){
```

```
        if(FD_ISSET(fd1, &rfds)){//para ver que pipe esta listo
           read(fd1, buffer,256);
           printf("Hemos leido de tuberia: %s.\nHemos leido : %s\n","tuberia3",buffer);
           close(fd1);
           int fd1 = open("tuberia3", O_RDONLY|O_NONBLOCK);
        }
        else{
           read(fd2, buffer,256);
           printf("Hemos leido de tuberia: %s.\nHemos leido : %s\n","tuberia5",buffer);
           close(fd2);
           int fd2 = open("tuberia5", O_RDONLY|O_NONBLOCK);
        }
     }
  }
          close(fd1);
  close(fd2);
  return 0;

}
```

TERMINAL 1:

```
[cursoredes@localhost ~]$ g++ practica4.cpp -o ej5
[cursoredes@localhost ~]$ ./ej5
Hemos leido de tuberia: tuberia5.
Hemos leido : 123456

Hemos leido de tuberia: tuberia3.
Hemos leido : blablabla
```

TERMINAL 2:

```
[cursoredes@localhost ~]$ echo 123456 > tuberia5
[cursoredes@localhost ~]$ echo blablabla > tuberia3
```