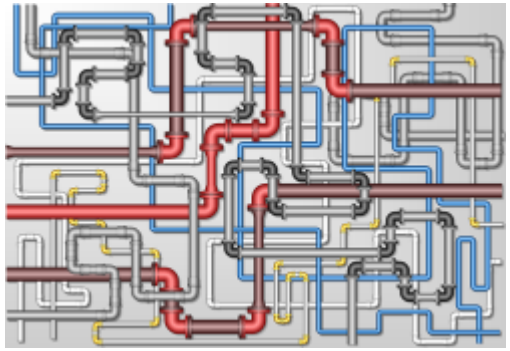
Pipes

C Prácticas



Un pipe o tubería se puede considerar como un canal de comunicación entre dos procesos. Las hay de dos tipos, sin nombre y con nombre también llamadas fifos. Cuando un proceso crea una tubería sin nombre sus hijos la heredan y se podrán comunicar. En esta entrada os explico y pongo un ejemplo de cómo se programa esta comunicación.

Las tuberías sin nombre, que son las que vamos a utilizar, se crean con la llamada al sistema pipe(). Sólo el proceso que hace la llamada a pipe() y sus descendientes podrán utilizarla. La llamada pipe() tiene la siguiente declaración:

```
int pipe(int fildes[2]);
```

Si la llamada funciona correctamente devolverá 0 y creará una tubería sin nombre; en caso contrario, devolverá -1 y en errno estará el código de error producido.

La tubería creada la vamos a poder manejar a través del array fildes. Los dos elementos de ese array se comportan como dos descriptores de fichero y los vamos a utilizar para escribir y para leer en la tubería.

- Para leer utilizaremos fildes[0], que se comporta como un fichero de sólo lectura.
- Para escribir utilizaremos files[1], que se comporta como un fichero de sólo escritura.

El kernel trata a la tubería como a un fichero del sistema, por lo que podremos utilizar las llamadas al sistema write() y read() para escribir y leer datos en la tubería.

Los descriptores de ficheros se heredan por los hijos que el proceso que haya creado la tubería genere con fork(). De esta forma, el padre escribirá en la tubería y el hijo, que habrá heredado la tubería podrá leer de ella, estableciendo así la comunicación entre padres e hijos.

Cuando un proceso no va a utilizar uno de los descriptores, es conveniente cerrarlo con la llamada a close() como cualquier otro fichero. Así por ejemplo, en una comunicación unidireccional entre un padre (escritor) y un hijo (lector), el padre cerrará el descriptor de lectura y el hijo cerrará el descriptor de escritura, puesto que no los van a usar.

La llamada a read() para sacar datos de la tubería bloquearán al proceso que la realice hasta que no haya datos que otro proceso haya escrito con write(). Por tanto el kernel se encarga de bloquear al proceso si no hay datos que leer. Los datos se escriben y leen en el mismo orden, el kernel proporciona el mecanismo FIFO por el que los datos se leen en el mismo orden en que se escriben.

Los datos de la tubería se gestionan en el buffer caché en memoria, con lo que la comunicación mediante tuberías es mucho más rápida que hacerlo con ficheros ordinarios. El bloque de datos más grande que puedo ubicar en la tubería depende del sistema pero suele ser 4096 bytes.

Cuando la tubería está llena las llamadas a write() quedan bloqueadas hasta que no se saquen suficientes datos para poder escribir el siguiente bloque de bytes.

Veamos un ejemplo de cómo un padre y un hijo comparten un string mediante una tubería.

```
#include &lt;sys/types.h&gt;
#include &lt;string.h&gt;
#include &lt;stdlib.h&gt;
#include &lt;stdio.h&gt;

#define SIZE 512

int main( int argc, char **argv )
{
  pid_t pid;
  int p[2], readbytes;
  char buffer[SIZE];

  pipe( p ); //Creamos la tubería en código padre.

  if ( (pid=fork()) == 0 )
  { // hijo
    close( p[1] ); /* cerramos el lado de escritura del pipe */

    //Leemos del pipe en bloques de SIZE bytes que almacenamos en la variable buffer.
    //mientras readbytes sea &gt; 0 escribimos los volcamos en la salida estandar que es
    //el fileid 1(primer argumento de la llamada a write()
    while( (readbytes=read( p[0], buffer, SIZE )) &gt; 0)
      write( 1, buffer, readbytes );

    close( p[0] );
  }
  else
  { // padre
    close( p[0] ); /* cerramos el lado de lectura del pipe */

    strcpy( buffer, "Mensaje del padre que se transmite por la tubería\n" );
    write( p[1], buffer, strlen( buffer ) );

    close( p[1] );
  }
  waitpid( pid, NULL, 0 ); //Esperamos a la terminación del hijo para no dejarlo huérfano
  exit(0);
}
```

Veamos otro ejemplo, esta vez comparten un array de enteros.

```
#include &lt;sys/types.h&gt;
#include &lt;string.h&gt;
#include &lt;stdlib.h&gt;
#include &lt;stdio.h&gt;
```

```
#define COUNT(x) ((int)(sizeof(x)/sizeof(int)))

int main( int argc, char **argv )
{
  pid_t pid;
  int p[2], i;
  int intSize = sizeof(int);
  int arrayLength;
  int arrayLeido[10];
  int array[]={1,2,3,4,5,6,7,8,9,10};

  arrayLength=COUNT(array);

  pipe( p ); //Creamos la tubería en código padre.

  if ( (pid=fork()) == 0 )
  { // hijo
    close( p[1] ); /* cerramos el lado de escritura del pipe */

    read(p[0], arrayLeido, arrayLength*intSize);

    for(i=0; i&lt;arrayLength; i++)
      printf("arrayLeido[%d]=%d\n",i,arrayLeido[i]);

    close( p[0] );
  }
  else
  { // padre
    close( p[0] ); /* cerramos el lado de lectura del pipe */

    write( p[1], array, arrayLength*intSize);

    close( p[1] );
  }
  waitpid( pid, NULL, 0 ); //Esperamos a la terminación del hijo para no dejarlo huérfano
  exit(0);
}
```

Para hacer una comunicación bidireccional habría que usar otra tubería para volcar datos del hijo al padre. Podríamos estar tentados a usar una única tubería, pero esto plantearía problemas de sincronización en el acceso a la misma y habría que solucionarlos con señales o semáforos.

Veamos un ejemplo con mensajes string:

```
#include &lt;sys/types.h&gt;
#include &lt;string.h&gt;
```

```c
#include &lt;stdlib.h&gt;
#include &lt;stdio.h&gt;

#define SIZE 512

int main( int argc, char **argv )
{
  pid_t pid;
  int a[2], b[2], readbytes;
  char buffer[SIZE];

  pipe( a );
  pipe( b );

  if ( (pid=fork()) == 0 )
  { // hijo
    close( a[1] ); /* cerramos el lado de escritura del pipe */
    close( b[0] ); /* cerramos el lado de lectura del pipe */

    while( (readbytes=read( a[0], buffer, SIZE ) ) &gt; 0)
      write( 1, buffer, readbytes );
    close( a[0] );

    strcpy( buffer, "Mensaje del hijo al padre\n" );
    write( b[1], buffer, strlen( buffer ) );
    close( b[1] );
  }
  else
  { // padre
    close( a[0] ); /* cerramos el lado de lectura del pipe */
    close( b[1] ); /* cerramos el lado de escritura del pipe */

    strcpy( buffer, "Mensaje del padre al hijo\n" );
    write( a[1], buffer, strlen( buffer ) );
    close( a[1]);

    while( (readbytes=read( b[0], buffer, SIZE )) &gt; 0)
      write( 1, buffer, readbytes );
    close( b[0]);
  }
  waitpid( pid, NULL, 0 );
  exit( 0 );
}
```

← Entrada anterior

Entrada siguiente →

Escuela Politécnica Superior de Elche

Universidad Miguel Hernández

Miguel Onofre Martínez Rach