
# Boletín 3.2
# Ejercicios sobre tuberías

July 14, 2016

1. Indica **de manera justificada** si las siguientes afirmaciones sobre la utilización de tuberías como mecanismo de sincronización entre dos procesos son verdaderas o falsas:

   a) Si la tubería está vacía, el lector siempre se queda bloqueado.

   b) Si en la tubería hay $p$ datos y el lector quiere leer $n$ datos, el lector se queda bloqueado.

   c) Las operaciones de lectura sobre una tubería pueden leer datos de tamaños distintos a los de las operaciones de escritura.

   d) El escritor puede escribir en la tubería aunque el lector no haya ejecutado ninguna operación de lectura sobre la misma.

   e) Dos procesos que quieren comunicarse a través de una tubería deben ejecutar ambos la llamada `pipe(tub)`.

   f) Una operación `write` sobre un fichero, si no genera error, siempre modifica el valor del puntero de posición de dicho fichero.

   g) El puntero de posición de un fichero abierto por un proceso es una característica asociada a dicho proceso y, consecuentemente, se guarda en el descritor de ese proceso.

h) Si un proceso deja un dato en una tubería, dos o más procesos pueden leer ese dato directamente de dicha tubería.

2. Indicar de manera justificada cuántas entradas de la tabla de ficheros abiertos por un proceso están ocupadas si, una vez creado dicho proceso, se realizan de manera sucesiva las siguientes operaciones:

   (a) Un `open` sobre un fichero que existe y sobre el que el usuario tiene permisos de lectura.

   (b) Un `read` sobre el fichero anterior.

   (c) Un `dup` sobre el fichero anterior.

   (d) Un `close` sobre el fichero abierto en la primera operación.

   (e) Un `pipe`.

   (f) Un `fork`.

3. ¿Qué significa realizar las siguientes operaciones:

   (a) `fd=open(``fich'',O_WRONLY|O_CREAT|O_TRUNC,0700);`

   (b) `fd=open(``fich'',O_WRONLY|O_CREAT|O_EXCL,0700);`

4. Realiza el ejercicio 7 de las transparencias.

5. Realiza el ejercicio 5 de las transparencias.

6. ¿Por qué no finaliza la ejecución del programa `pie2.c` de la página 150 del libro de Badía et al.?

7. Sea el programa que se especifica a continuación (y en el que se han numerado las líneas). A partir de él responde de forma justificada a las siguientes preguntas:

   (a) Si el valor del argumento que se pasa al programa es 6, ¿cuál es la salida por pantalla de la ejecución del programa?

   (b) ¿Qué varía con respecto al apartado anterior si se elimina la función `close` de la línea 9?

   (c) Si se intercambia el orden de las líneas 6 y 7, ¿cuál sería entonces la salida por pantalla del proceso hijo si el valor del argumento que se pasa al programa fuese 6?

   (d) ¿Qué ocurre si en el programa original se intercambian las líneas 22 y 23?

```
1. #include <stdio.h>
2. #include <unistd.h>
3. #include <stdlib.h>
4. int main(int argc, char *argv[])
5. { int tubo[2], i, dato, suma, pid;
6.   pipe(tubo);
7.   pid=fork();
8.   if (pid == 0)
```

```
 9.    { close(tubo[1]);
10.      close(0);
11.      dup(tubo[0]);
12.      close(tubo[0]);
13.      suma = 0;
14.      while (read(0,&dato,sizeof(int))>0)
15.        if (dato%2 == 0) suma = suma + dato;
16.      printf("Hijo finaliza. Resultado: %d \n", suma);
17.      exit(0);
18.    } else {
19.      close(tubo[0]);
20.      dato=atoi(argv[1]);
21.      for (i=1;i<=dato;i++) write(tubo[1],&i,sizeof(i));
22.      close(tubo[1]);
23.      wait(NULL);
24.      printf ("Padre finaliza\n");
25.      exit(0);
26.    }
27. }
```

8. Reescribe el código del proceso hijo del programa anterior para que el resultado de su ejecución se almacene en el fichero que se pase como segundo argumento al programa.

9. Se pretende utilizar una tubería para controlar que dos procesos accedan a un recurso de forma excluyente (de uno en uno). Indica de manera justificada si el siguiente código sincroniza ambos procesos para conseguirlo.

```
int testigo[2]={0,1};
int tub[2];
...
pipe(tub);
write(tub[1],testigo,sizeof(testigo));

if (fork() != 0)
{   if (fork() != 0)
    {
      .....
    } else {
       ....
       read(tub[0],testigo,sizeof(int));
       /*   ACCESO A RECURSO */
       write(tub[1],testigo,sizeof(int));
       .....
    }
}else {
       ....
       read(tub[0],testigo,sizeof(int));
       /*   ACCESO A RECURSO */
       write(tub[1],testigo,sizeof(int));
       ....
  }
```

10. En el siguiente código se muestra las instrucciones que ejecutan dos procesos, A y B, para sincronizar su ejecución. Suponer que las tuberías `tub1` y `tub2` las ha

creado el proceso padre antes de crear a los procesos A y B y que, cuando se ejecutan estos fragmentos de código, no hay ningún dato almacenado en ellas. Indica el orden en que se ejecutan los fragmentos de código etiquetados como *partes* en el código de ambos procesos.

```
...
/* Proceso A */
...
read(tub1[0],&testigo,sizeof(int));
/* parte A1 */
write(tub2[1],&testigo,sizeof(int));
read(tub1[0],&testigo,sizeof(int));
/* parte A2*/
write(tub2[1],&testigo,sizeof(int));
read(tub1[0],&testigo,sizeof(int));
....
```

```
/* Proceso B */
...
write(tub1[1],&testigo,sizeof(int));
read(tub2[0],&testigo,sizeof(int));
/* parte B1 */
write(tub1[1],&testigo,sizeof(int));
read(tub2[0],&testigo,sizeof(int));
/* parte B2 */
write(tub1[1],&testigo,sizeof(int));
....
```

11. Realiza el ejercicio 9 de las transparencias. Como allí se indica, el proceso padre ha de imprimir los datos pares e impares que lee de la tubería de manera alternada, de forma que se muestre una secuencia de números ordenada: 0, 1, 2, 3, ... El ejemplo 4 de las transparencias puede servirte de referencia.

12. Escribe un programa en C que cree dos procesos hijos, H1 y H2. Todos ellos se estarán ejecutando concurrentemente en el sistema. El proceso padre creará 100 datos enteros de forma aleatoria y se los enviará a uno de los procesos hijos: los que son múltipos de 5 a H1 y el resto a H2. Cada uno de los procesos hijos sumará los datos recibidos y el valor acumulado junto con el total de datos recibidos se enviarán al proceso padre, que los mostrará en pantalla. Primero debe recibir e imprimir los datos enviados por H1 y después los de H2. En las transparencias 9 y 10 del tema 3 tienes un programa que genera números aleatorios en C.

13. Write a program in lenguaje C that intercommunicates two processes so that they execute the following shell commands by using pipes:

    ```
    grep "pattern" file | wc -l >> out
    ```

    That is, the shell commands `grep` and `wc` shown above should be run by invoking the `execlp` system call. Assume that `pattern`, `file` and `out` are a pattern and two file names respectively, given as three arguments to the program.

14. Realiza un programa en el que, mediante llamadas al SO Unix, se genere un fichero con el resultado de ejecutar la orden

    ```
    ls -l | cut -c1-10 > fichero
    ```

    El nombre del fichero se pasará como argumento al programa. Los ejercicios 1 y 3 del libro de Badía et al. pueden servirte de referencia. Y el vídeo *"Ejercicio sobre comandos enlazados por tuberías"*, donde se resuelve el ejercicio 19 de este boletín, también.

15. Modifica el programa anterior para que, además de volcar el resultado al fichero que se pasa como argumento, se muestre por la salida estándar.

    **Nota:** El comando `tee` permite volcar la información que se muestra por pantalla simultáneamente a un fichero que se pasa como argumento a dicho comando. En este ejercicio no puedes ejecutar explícitamente este comando con la función `execlp`. Has de realizar el volcado al fichero y a la pantalla de forma manual (usando las llamadas `read` y `write`) de manera similar a como se indica en al transparencia 116 del tema. El vídeo *"Ejercicio sobre comandos enlazados por tuberías"*, donde se resuelve el ejercicio 19 de este boletín, puede servirte de referencia.

16. Modifica el programa anterior para que ejecute explícitamente el comando `tee` mediante la función `execlp`. Es decir, el programa ha de ejecutar lo siguiente:

    ```
    ls -l | cut -c1-10 | tee fichero
    ```

17. Realiza un programa en lenguaje C que, utilizando llamadas al sistema, muestre por pantalla el resultado de ejecutar la orden

    ```
    grep a argv[1]
    ```

    y, a continuación, el resultado de ejecutar

    ```
    grep a argv[1] | wc -l 2>> argv[2]
    ```

    La orden `grep` debe ejecutarse solo una vez.

    El vídeo *"Ejercicio sobre comandos enlazados por tuberías"*, donde se resuelve el ejercicio 19 de este boletín, puede servirte de referencia.

18. Realiza un programa en lenguaje C que, utilizando llamadas al sistema, ejecute el comando `grep patron fich1` y, a continuación, el comando `grep patron fich2`. Sobre la salida generada por estos dos comandos se ejecutará un único comando `sort -n 2> fich3`. El programa recibirá como argumento el patrón a buscar con los comandos `grep`, los ficheros `fich1` y `fich2` sobre los que realizar la búsqueda y el fichero `fich3` requerido por el comando `sort`. Incluye el esquema de los procesos y tuberías usados.

    Una posible salida de la ejecución del programa anterior podría ser la siguiente:

    ```
    $ cat f1
    3
    10
    6
    100
    21
    $ cat f2
    120
    ```

```
46
14
$ doble_grep_sort 1 f1 f2 f3
10
14
21
100
120
$
```

19. Realiza un programa en lenguaje C que, utilizando llamadas al sistema, muestre por pantalla el resultado de ejecutar la orden

    ```
    grep argv[1] argv[2]  2>> argv[3] | sort -n
    ```

    La salida del comando `grep` deberá guardarse también en un fichero que se pasa al programa como cuarto argumento. Esto es,

    ```
    grep argv[1] argv[2]  2>> argv[3] >> argv[4]
    ```

    La orden `grep` debe ejecutarse solo una vez.

20. Escribe un programa en C que cree dos procesos hijos, H1 y H2. Todos ellos se estarán ejecutando concurrentemente en el sistema. El proceso H1 generará tantos números aleatorios entre 0 y 9 como se le pasa al programa como primer argumento. El proceso H2 generará tantos números aleatorios entre 10 y 19 como se le pasa al programa como segundo argumento. El proceso padre imprimirá por pantalla los datos que le envían los hijos de manera alternada: un número de H1, otro de H2, uno de H1, otro de H2, y así sucesivamente hasta finalizar los datos que recibe de estos.

21. Realiza un programa en lenguaje C que, utilizando llamadas al sistema, muestre por pantalla el resultado de ejecutar el comando `grep patron fich1` y, a continuación, muestre por pantalla el resultado de ejecutar la siguiente secuencia de comandos: `grep patron fich1 | sort -n`. La orden `grep` debe ejecutarse solo una vez y la salida de ambas secuencias de comandos debe ser almacenada también en un fichero. Si este fichero existe inicialmente, se sobreescribirá su contenido.

    El programa recibirá como argumento el patrón a buscar con el comando `grep`, que ha de ser un número entero, el fichero `fich1` sobre el que realizar la búsqueda y el fichero `fich2` en el que guardar el resultado de las ejecuciones.

    Incluye un esquema con los procesos, tuberías y flujo de información usados.

    Una posible salida de la ejecución del programa anterior, al que llamaremos `tuberias.c`, podría ser la siguiente:

```
$ cat fich1
12
4
25
8
2
$ gcc tuberias.c -o tuberias
$ tuberias 2 fich1 fich2
Resultado del grep:
12
25
2
Resultado del sort:
2
12
25
$cat fich2
Resultado del grep:
12
25
2
Resultado del sort:
2
12
25
$
```

22. Realiza un programa en lenguaje C que, utilizando llamadas al sistema, ejecute la siguiente secuencia de comandos:

    ```
    ls -l | grep tex 2>> f2 | wc -l > f1
    ```

    Después guardará al final del contenido del fichero `f1` el resultado de ejecutar la siguiente secuencia de comandos:

    ```
    ls -l | grep tex 2>> f2 | wc -w
    ```

    Las órdenes `ls` y `grep` deben ejecutarse solo una vez.

    Incluye un esquema con los procesos, tuberías y flujo de información usados.

    Una posible salida de la ejecución del programa anterior, al que llamaremos `tuberias.c`, podría ser la siguiente:

    ```
    $ ls -l *tex*
    -rw------- 1 castano uji  5 Jan  1 21:44 f1.tex
    -rw------- 1 castano uji 12 Jan  1 21:44 f2.tex
    $ gcc tuberias.c  -o tuberias
    ```

```
$ tuberias
$ cat f1
Resultado de wc -l:
2
Resultado de wc -w:
18
$
```