Boletín 4

Ejercicios sobre gestión del sistema de ficheros

July 14, 2016

1. El tamaño de un fichero, ¿es un atributo estático o dinámico? ¿Y la fecha de modificación del fichero? ¿Y el puntero de posición (de lectura/escritura)? Justifica en todos los casos las respuestas.

2. Explica qué diferencia hay entre nombrar en un sistema tipo UNIX a un fichero por su nombre absoluto y por su nombre relativo.

3. ¿Cuándo se crea la tabla de ficheros abiertos de un proceso? ¿Dónde se almacena? ¿Está vacía cuando se crea? ¿Cuándo se le añaden elementos?

4. ¿Cuándo se crean nuevas entradas en la tabla de ficheros abiertos en el sistema? ¿Y en la de inodos?

5. ¿Pueden apuntar distintas entradas de la tabla de ficheros abiertos de un proceso a la misma entrada de la tabla de ficheros abiertos en el sistema? Justifica la respuesta.

6. ¿Puede apuntar una entrada de la tabla de ficheros abiertos de un proceso a la misma entrada de la tabla de ficheros abiertos en el sistema que otra entrada de la tabla de ficheros abiertos por otro proceso distinto? Justifica la respuesta.

7. ¿Dónde se alamacena el puntero de posición (de lectura/escritura) de un fichero? ¿Por qué se alamacena ahí y no en otros lugares?

8. Sea el siguiente programa:

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

int main(int argc, char *argv[])
{
          int n, fd;
          char buffer[10];

           if (argc < 1) {
             printf("Faltan parametros ...\n");
             exit(-1);
           }

          fd=open(argv[1],O_RDONLY);
          if (fd <0) {
             printf("Error apertura de fichero %s\n",argv[1]);
             exit(-1);
          }
          if (fork() != 0){
             wait(NULL);
             printf("Soy el padre\n");
             while ((n=read(fd,buffer,10))>0)
             {   write(1,buffer,n);
                 sleep(1);
             }
          } else {
               printf("Soy el hijo\n");
               while ((n=read(fd,buffer,10))>0)
               {   write(1,buffer,n);
                   sleep(1);
               }
           }
          close(fd);
          exit(0);
}
```

A partir de este programa responde a las preguntas que aparecen más abajo asumiendo que este se ejecuta pasándole como argumento un fichero que contiene la siguiente información:

```
abcdefghijklmnopqrstuvxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
```

(a) ¿Qué salida proporciona la ejecución del programa? Justifica la respuesta.

(b) ¿La salida sería diferente si el proceso padre no realizase la llamada `wait`?

9. Sea el siguiente programa:

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

int main(int argc, char *argv[])
{
    int n, fd;
    char buffer[10];

    if (fork() != 0){
      fd=open(argv[1],O_RDONLY);
      if (fd <0) {
        printf("Error apertura de fichero %s\n",argv[1]);
        exit(-1);
      }
      wait(NULL);
      printf("Soy el padre\n");
      while ((n=read(fd,buffer,10))>0)
      {  write(1,buffer,n);
         sleep(1);
      }
    } else {
       fd=open(argv[1],O_RDONLY);
       if (fd <0) {
         printf("Error apertura de fichero %s\n",argv[1]);
         exit(-1);
       }
       printf("Soy el hijo\n");
       while ((n=read(fd,buffer,10))>0)
       {  write(1,buffer,n);
          sleep(1);
       }
     }
     close(fd);
     exit(0);
}
```

A partir de este programa responde a las preguntas que aparecen más abajo asumiendo que este se ejecuta pasándole como argumento un fichero que contiene la siguiente información:

```
abcdefghijklmnopqrstuvxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
```

(a) ¿Qué salida proporciona la ejecución del programa? Justifica la respuesta.

(b) ¿La salida sería diferente si el proceso padre no realizase la llamada `wait`?

10. Sea el siguiente programa:

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

main(int argc, char *argv[])
{
   int n,fd;
   char buffer[10];

   if (argc < 2)
     {
       printf("Faltan parametros....\n");
       exit(-1);
     }
   fd=open(argv[1],O_RDWR);
   if (fd == -1){
      perror("No existe el fichero \n");
      exit(-1);
   }
   if (fork()!= 0){
     wait(NULL);
     printf("Soy el padre y mi fd = %d\n",fd);
     while ((n=read(fd,buffer,10))>0)
     {  write(1,buffer,n);
        sleep(1);
     }
     printf("Acaba el padre\n");
   } else {
     printf("Soy el hijo\n");
     printf("Antes de ejecutar exec, fd = %d\n",fd);

     close (0);
     dup(fd);
     close(fd);

     execlp("./prog1","./prog1",NULL);
```

```
      printf("Problema con el exec...\n");
     }
   close(fd);
   exit(0);
}

/*prog1*/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

main()
{
   int n;
   char buffer[10];

     while ((n=read(0,buffer,10))>0)
     {   write(1,buffer,n);
         sleep(1);
     }
     printf("Acaba el hijo\n");
     exit(0);
}
```

A partir de este programa responde a las preguntas que aparecen más abajo asumiendo que este se ejecuta pasándole como argumento un fichero que contiene la siguiente información:

```
abcdefghijklmnopqrstuvxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
```

(a) ¿Qué salida proporciona la ejecución del programa? Justifica la respuesta.

(b) ¿La salida sería diferente si el proceso padre no realizase la llamada `wait`?

11. Sea el siguiente programa:

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

int main()
{
   int pid, i, n, fd;
   char buffer;
```

```
   fd = open ("f",O_RDONLY);
   printf("Proceso principal: %d con fd=%d\n", getpid(),fd);
   if ((n=read(fd,&buffer,1))>0) write (1,&buffer,1);
   write (1,"\n",1);

   for (i=0; i<3; i++)
   {  pid = fork();
      if (pid == 0)
      {  if ((i%2) == 0) fd = open ("f",O_RDONLY);
         printf("Proceso %d i=%d con fd=%d\n", getpid(), i, fd);
         while ((n=read(fd,&buffer,1))>0) write (1,&buffer,1);
         close (fd);
         exit (0);
      } else wait (NULL);
   }
   printf("Proceso %d i=%d con fd=%d\n", getpid(), i, fd);
   while ((n=read(fd,&buffer,1))>0) write (1,&buffer,1);
   close (fd);
   exit (0);
}
```

A partir de este programa responde a las preguntas que aparecen más abajo asumiendo que el fichero `f` contiene la siguiente información:

```
abcdefghijklmnopqrstuvxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
```

(a) ¿Qué salida proporciona la ejecución del programa? Justifica la respuesta.

(b) Muestra el valor de la variable `fd` y el contenido de la tabla de descriptores de ficheros para cada uno de los procesos una vez creados estos y antes de que ejecuten la función `close`. Muestra también en cada caso el contenido de la tabla de ficheros abiertos por el sistema y el de la tabla de inodos.

12. Realizar un programa que muestre por pantalla el nombre de los subdirectorios contenidos en un directorio y el correspondiente identificativo del usuario propietario de estos. El nombre del directorio se introducirá desde la línea de comandos.

    La ejecución del programa podría generar una salida por pantalla similar a la siguiente:

```
$ mostrar_subdirs ..
../. (UID 831)
../.. (UID 831)
../et_tuberias (UID 831)
../et_mutex (UID 831)
../et_procesos (UID 831)
$
```

13. Escribe un programa en C que, a partir del nombre de un directorio introducido desde la línea de comandos, comprima y añada al fichero `segur.zip` aquellos ficheros regulares que encuentre en el directorio pasado como argumento y que pertenezcan al usuario que ejecuta el programa.

    Para comprimir el fichero `directorio/fich` y añadirlo al fichero `segur.zip` podemos utilizar el programa `zip` de la siguiente forma:

    ```
    /bin/zip  -g  segur.zip  directorio/fich
    ```

    Esta orden se deber ejecutar con cada uno de los ficheros regulares contenidos en el directorio `directorio`. Tened en cuenta que no se podrán comprimir dos o más ficheros de forma simultánea.

14. Implementar un programa en C que imprima por pantalla los 8 primeros caracteres de las 4 primeras líneas de los ficheros regulares contenidos en el directorio que se introduce al programa como argumento.

    La salida que proporciona el programa es equivalente a ejecutar para cada uno de los ficheros regulares del directorio los siguientes comandos enlazados:

    ```
    cut -c1-8 directorio/fichero | head -4
    ```

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

    ```
    $ cut_head .
    Primeros 8 caracteres de las 4 primeras lineas del fichero ./ej3.c:
    #include
    #include
    #include
    #include

    Primeros 8 caracteres de las 4 primeras lineas del fichero ./ej1.c:
    #include
    #include
    #include
    #include

    $
    ```

15. (a) Write a program in C lenguaje 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.

(b) Modify the previous program so that it executes the previous shell commands for each of the regular files that a directory contains. The directory is given as an argument to the program and, therefore, it should be navigated by using the corresponding system calls.

This could be a possible output dumped on the screen by the program:

```
$ find_dir dir pattern out
$ cat out
file1: 3
file2: 4
file3: 5
file4: 7
$
```

where the input arguments, `dir`, `pattern` and `out` are the directory name, the pattern to be found and the name of the output file, respectively. And `file1`, `file2`, `file3` and `file4` are the names of the files contained in the `dir` directory; and `3`, `4`, `5` and `7` are the total number lines that have the pattern in those files.

16. Realiza un programa en lenguaje C que, utilizando llamadas al sistema, muestre por pantalla el nombre de los enlaces duros y blandos contenidos en un directorio que se pasa como argumento al programa. Puede que tengas que utilizar la variante adecuada de la función `stat()`.

17. Modifica el ejercicio 19 del boletín de ejercicios de tuberías para que la operación `grep` se realice sobre cada uno de los ficheros regulares que contiene en un primer nivel el directorio que se pasa al programa como segundo argumento. El primero, tercer y cuarto argumento continúan siendo los mismos que en el ejercicio anterior.

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

```
$ ls dir
f1  f2
$ cat dir/f1
2
14
1
$ cat dir/f2
10
5
$ ./recorrer_dir 1 dir ferr fout
Apariciones de 1 en dir/f2
10
```

```
Apariciones de 1 en dir/f1
1
14
$ cat fout
10
14
1
$
```

18. 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
$
```

19. Sean los siguientes programas:

```
$ cat puntero_fich.c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

main(int argc, char *argv[])
{
   int n,fd;
   char buffer[3];

   fd=open(argv[1],O_RDWR);
   printf("Soy el padre y mi fd = %d\n",fd);
   if ((n=read(fd,buffer,3))>0)  write(1,buffer,n); printf("\n");

   if (fork()!= 0){
     wait(NULL);
     printf("Soy el padre y mi fd = %d\n",fd);
     if ((n=read(fd,buffer,3))>0)  write(1,buffer,n); printf("\n");
     printf("Soy el padre antes de ejecutar prog\n");
     close (0);
     dup(fd);
     close(fd);

     execlp("./prog","./prog",NULL);
     printf("Acaba el padre\n");
   } else {
     printf("Soy el hijo y mi fd = %d\n",fd);
     if ((n=read(fd,buffer,3))>0)  write(1,buffer,n); printf("\n");

     fd=open(argv[1],O_RDWR);
     printf("Soy el hijo y mi fd = %d\n",fd);
     while ((n=read(fd,buffer,3))>0)
     {  write(1,buffer,n);
        sleep(1);
     }
     close(fd);
     exit(0);
```

```
   }
}
$ cat prog.c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

main()
{
   int n;
   char buffer[10];

     printf("Dentro de prog\n");
     while ((n=read(0,buffer,3))>0)
     {   write(1,buffer,n);
         sleep(1);
     }
     printf("Acaba prog\n\n");
     exit(0);
}
```

A partir de estos programas indica de manera justificada qué salida proporciona la ejecución del primer programa asumiendo que se le pasa como argumento un fichero que contiene la siguiente información:

```
abcdefghijklmnopqrstuvxyz
```

20. 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
$
```

21. Realiza un programa en lenguaje C que, utilizando llamadas al sistema, ejecute el comando

    ```
    wc -l <fichero>
    ```

    para cada uno de los ficheros regulares que estén contenidos en el directorio que se pasa al programa como argumento y que tengan más de un enlace duro. No es necesario buscar de forma recursiva en ese directorio.

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

    ```
    $ ln f f.h
    $ ls -l
    -rw------- 2 castano uji  5 Jan  1 22:32 f
    -rw------- 2 castano uji  5 Jan  1 22:32 f.h
    -rw------- 1 castano uji 27 Dec 31 19:00 fich
    $ cat f
    hola
    $ recorrer_dir .
    wc -l  ./f
    1 ./f
    wc -l  ./f.h
    1 ./f.h
    $
    ```