Test sesión 8

Dado el siguiente trozo de código:

```
int fifo;
while((fifo = open("mipipe", O_WRONLY | O_NONBLOCK)) == -1 && errno == ENXIO && i<30)
{
        sleep(2);
        i++;
}
```

Asumiendo que la pipe con nombre "mipipe" existe, indica de las opciones siguientes, la que describe qué realiza este código:
Trieu-ne una:

a. Intenta abrir una pipe con nombre "mipipe" para sólo escritura esperando a que aparezcan lectores durante al menos un minuto.
b. Espera por una señal SIGPIPE durante al menos 1 minuto.
c. Intenta sobreescribir una pipe con nombre "mipipe" durante al menos 1 minuto.
d. Intenta desbloquear una pipe con nombre "mipipe" durante al menos 1 minuto.

Dado el siguiente código incompleto:

```
int main() {
      char buf[256];
      int pd[2];
      pipe(pd);
      int pid = fork();
      if(pid == 0) {
            dup2(pd[1], 1);
            close(pd[0]);
            close(pd[1]);
            execlp("ls","ls",NULL);
      }
      else {
            close(pd[1]);
            read(<<x1>>, buf, sizeof(buf));
            write(<<x2>>, <<x3>>, <<x4>>);
            close(pd[0]);
            waitpid(-1,NULL,0);
      }
}
```

Indica cuál de las siguientes opciones son **CIERTAS** de forma que el resultado de la ejecución fuera la de la ejecución del comando *ls* (asumiendo que no se redirecciona la entrada ni la salida de la ejecución del programa).
Trieu-ne una:

a. <<x1>> debería ser pd[0], <<x2>> debería ser pd[1], <<x3>> debería ser *buf* y <<x4>> debería ser *strlen(buf)*
b. <<x1>> debería ser pd[0], <<x2>> debería ser 1, <<x3>> debería ser *buf* y <<x4>> debería ser *strlen(buf)*
c. <<x1>> debería ser pd[1], <<x2>> debería ser 0, <<x3>> debería ser *buf* y <<x4>> debería ser *sizeof(buf)*
d. <<x1>> debería ser pd[0], <<x2>> debería ser pd[1], <<x3>> debería ser *buf* y <<x4>> debería ser *strlen(buf)*

Dada la siguiente secuencia de comandos y salida asociada, ejecutada desde un terminal (**Terminal 1)**:

```
# mknod mipipe p
# ls
mipipe out.txt code.c
# ls > mipipe
```

En otra terminal (**Terminal 2**) en la misma máquina, y desde la misma carpeta que en la terminal anterior y ejecutamos el siguiente comando:

```
# cat < mipipe
```

De las siguientes opciones, indique cuál es la que muestra el resultado de la ejecución del comando en **Terminal 2:**
Trieu-ne una:

a. En Terminal 2, se obtendrá la siguiente salida:
code.c
out.txt
b. En Terminal 2, se obtendrá la siguiente salida:
code.c
mipipe
out.txt
c. En Terminal 2 el comando se queda en espera de que en Terminal 1 se escriban caracteres desde línea de comandos.
d. En Terminal 2, el comando no muestra ningún resultado y se queda bloqueado ya que se no se indica fichero para mostrar.

Una comunicación entre dos procesos que sea bidireccional, es decir que ambos procesos puedan tanto escribir como leer mensajes, se podría implementar de varias formas. Indica cuál de las siguientes afirmaciones es **CIERTA**:
Trieu-ne una o més:
a. Con una única pipe con nombre sería suficiente ya que tiene dos extremos.
b. Necesitamos dos pipes con nombre ya que ambos procesos leyendo/escribiendo una sola pipe se podría perder información o alguno de los procesos podría bloquearse.
c. Con una única pipe sin nombre sería suficiente ya que tiene dos extremos.
d. Necesitamos dos pipes sin nombre ya que ambos procesos leyendo/escribiendo una sola pipe se podría perder información o alguno de los procesos podría bloquearse.
e. Es imposible porque no comparten memoria.

Dado el siguiente código:

```
int main() {
     int fd[2], ret;
     char buf[32];
     pipe (fd);
     if (fork() == 0) {
             close (fd[1]);
             ret = read(fd[0], buf, sizeof(buf));
             close(fd[0]);
             write (1, buf, ret);
     } else {
             close (fd[0]);
             sprintf(buf, "Toc 1\n");
             write (fd[1], buf, strlen(buf));
             sprintf(buf, "Toc 2\n");
             write (fd[1], buf, strlen(buf));
             sprintf(buf, "Toc 3\n");
             write (fd[1], buf, strlen(buf));
             close (fd[1]);
     }
```

Si ejecutamos el programa sin redireccionar la entrada ni la salida, indica de las opciones siguientes cuáles son **CIERTAS**:
Trieu-ne una:
a. Por salida estándar se mostrará el mensaje "Toc 1". El proceso hijo acabará su ejecución pero el proceso padre quedará bloqueado esperando lectores.
b. Por salida estándar no mostrará ningún mensaje porque se cierran los canales antes de poder leer ningún mensaje.
c. Por salida estándar mostrará como mínimo los mensajes: "Toc 1", "Toc 2". Los procesos acabaran su ejecución.
d. Por salida estándar mostrará como mínimo el mensaje "Toc 1", el resto de los mensajes podrían aparecen (en orden) pero no necesariamente. Los dos procesos acabaran su ejecución.

Un socket y una pipe se diferencian en:
Trieu-ne una o més:

a. Un socket permite comunicar procesos en máquinas diferentes pero conectadas a través de una red.
b. Una pipe utiliza dos canales para comunicar procesos, mientras que un socket utiliza un único canal de lectura y escritura.
c. Para poder utilizar sockets el programa debe tener permisos de root.
d. El socket utiliza dos canales para comunicar procesos, mientras que la pipe utiliza un único canal de lectura y escritura.

Para el caso de una operación de escritura sobre una pipe sin lectores, indica cuál de las afirmaciones es **CIERTA**:
Trieu-ne una:

a. El proceso que solicita la operación de escritura quedará bloqueado a la espera de lectores.
b. El programa no compila.
c. Cuando el proceso solicite la operación de escritura sobre una pipe sin lectores, éste recibirá una señal SIGPIPE.
d. La operación de escritura retornará un 0 indicando que ha podido escribir 0 bytes.

La operación de E/S *write*:
n = write (fd, buffer, count);
Solicita la escritura de *count* bytes a un dispositivo identificado por el número de canal *fd*. Si hubiera espacio para *count* bytes se escriben *count* bytes y retorna el valor de *count*.
Trieu-ne una:

Respostes

Vertader
Fals

La operación de E/S *read*:
n = read (fd, buffer, count);
Solicita la lectura de *count* bytes de un dispositivo identificado por el número de canal *fd*. Si hubiera disponible para leer un número *X* de bytes tal que 0 < X < *count*, el proceso que la ejecuta se bloqueará a la espera de completar la lectura de *count* bytes.
Trieu-ne una:

Respostes

Vertader
Fals

Cuando un proceso realiza una operación de escritura sobre una pipe llena:
Trieu-ne una:

a. La operación de escritura retorna un código de error.
b. Se produce una señal de SIGFULL.
c. Se sobreescribe la pipe perdiendo la información que había antes en la pipe y que aún no había sido leída.
d. El mensaje queda pendiente de enviar y el proceso continúa su ejecución.
e. El proceso se bloquea en la operación de escritura.

Dado el siguiente trozo de código con un esquema de creación de procesos y pipes sin nombre:

```
int pipe0[2], pipe1[2], pipe2[2];

pipe (pipe0);
// PROCESO A
if  (fork() == 0) {
      // PROCESO B
      pipe (pipe1);
      if (fork() == 0) {
            // PROCESO C
            pipe (pipe2);
      }
}
```

Indica cuál de las siguientes afirmaciones son **CIERTAS**:
Trieu-ne una o més:

a. pipe2 no puede utilizarse para comunicar: B y C
b. pipe1 puede utilizarse para comunicar: A y C, B y C
c. pipe0 puede utilizarse para comunicar: A y B, A y C, B y C
d. pipe1 no puede utilizarse para comunicar: A y B

Dado el siguiente trozo de código:

```
char buf[SIZE];
while ((n = read(0, buf, SIZE)) > 0)
      write(1, buf, n);
```

Indica cuáles de las siguientes afirmaciones son **CIERTAS**:
Trieu-ne una:

a. El proceso que ejecuta el código finalizará el bucle cuando lea el carácter ^D
b. El proceso que ejecuta el código quedará bloqueado en la operación write hasta que no logre escribir SIZE bytes
c. El proceso que ejecuta el código queda bloqueado en la operación read hasta que no logre leer SIZE bytes.
d. El proceso escribirá n bytes por salida estándar y continuará la ejecución del bucle mientras n>0

Dado el siguiente código:

```
main() {
      char buf[64];
      int pd[2];
      pipe(pd);
      if (fork()==0) {
            dup2(pd[0], 0);
            close(pd[0]);
            close(pd[1]);
            execlp("cat","cat",NULL);
      } else {
            close(pd[0]);
            sprintf(buf,"Un saludo\n");
            write(pd[1],buf,strlen(buf));
            close(pd[1]);
            sprintf(buf, "Fin del saludo\n");
            write(1, buf, strlen(buf));
      }
}
```

Suponiendo que lo ejecutamos sin redireccionar ningún canal, cual de las siguientes afirmaciones es **CIERTA**:
Trieu-ne una:

a. Por la salida estándar no mostrará ningún mensaje
b. Por la salida estándar mostrará únicamente el mensaje "Un saludo"
c. Por la salida estándar mostrará únicamente el mensaje "Fin del saludo"
d. Por la salida estándar mostrará los mensajes "Un saludo" y "Fin del saludo"

El siguiente trozo de código:

int v[2];
pipe (v);

Crea una pipe sin nombre, donde el parámetro v corresponde al buffer donde se almacenarán los datos leídos o escritos.
Trieu-ne una:

Respostes

Vertader
Fals

El comando para crear una pipe con nombre "mipipe" en la se realizarán escrituras y lecturas de un solo caracter por mensaje, es el siguiente:

mknod "mipipe" -c

Trieu-ne una:
Respostes

Vertader
Fals
