SISTEMAS OPERATIVOS - LABORATORIO
16 de enero de 2024, Turno 2

Nombre _____ DNI _____

Apellidos _____ Grupo _____

INSTRUCCIONES

- Se debe entregar una carpeta individual con el código de cada apartado (1A, 1B, etc.), para así garantizar la evaluación independiente de cada uno.
- Si el código no compila o su ejecución produce un error grave la puntuación de ese apartado será 0.

Cuestión 1. (5,5 puntos) Desarrollar un programa **`lanzamiento`** que ejecute comandos de bash especificados por el usuario, y espere a su terminación. Se proporciona un esqueleto de código con un *Makefile*, así como dos ficheros de entrada para comprobar el correcto funcionamiento de los dos últimos apartados del ejercicio.

a. **(1,5 puntos)** Implementar la opción -x del programa **`lanzamiento`**, que permitirá al usuario indicar como argumento de esta opción un comando de bash (simple o compuesto) que el programa **`lanzamiento`** ejecutará. Para ello, el programa incluirá una función `pid_t run_command(const char* command)` que:
   - Creará un proceso hijo
   - Hará que el proceso hijo ejecute el programa bash, pasándole como opciones de línea de comandos "-c" y el comando que el usuario quiere ejecutar (argumento de la función). Así por ejemplo, si se desea ejecutar "echo hello; sleep 1; echo goodbye" vía bash, se debería ejecutar el siguiente comando desde el proceso hijo:
     `/bin/bash -c "echo hello; sleep 1; echo goodbye".`
   - La función devolverá el pid del proceso hijo (sin esperar a que éste termine su ejecución)

   Cuando el programa reciba la opción -x invocará la función run_command pasando como comando el argumento de la opción -x, quedándose después esperando hasta que el proceso hijo termine de ejecutar el comando bash.

b. **(2 puntos)** Implementar la opción -s del programa **`lanzamiento`**, que permitirá al usuario indicar como argumento de la opción la ruta de un fichero con comandos bash a ejecutar. Este fichero será interpretado por líneas, tomando cada línea como un comando a ejecutar con la función `run_command()` del apartado anterior. Los comandos se ejecutarán de forma secuencial, esperando a que un comando termine antes de ejecutar el siguiente. **Sugerencia:** usar fgets para leer del fichero por líneas.

c. **(2 puntos)** Implementar la opción -b del programa, que tendrá efecto solamente si se pasa junto con la opción -s. En este caso, si se pasan las opciones -x y -s, los comandos del fichero indicado por la opción -x se ejecutarán uno tras otro sin esperar a que el comando anterior termine. Sólo cuando se hayan lanzado a ejecución todos los comandos indicados en el fichero (cada uno por parte de un proceso "bash" hijo) se esperará a que terminen todos. Asimismo, cada vez que uno de los comandos lanzados termine, el programa imprimirá por la salida estándar el número de comando que ha terminado, su PID y código de terminación –-p.ej., "@@ Command #3 terminated (pid: 11576, status: 0)", como se muestra en el ejemplo de ejecución–. Para ello, el programa deberá mantener una estructura de datos –por simplicidad, array de longitud máxima prefijada– que permita asociar los PIDs de los hijos, con el número de cada comando en el orden de lanzamiento.

Ejemplo de ejecución

```
# Prueba apartado A
$ ./lanzamiento -x "echo hello; sleep 1; echo goodbye"
hello
goodbye

# Prueba apartado B
$ ./lanzamiento -s commands_b
@@ Running command #0: echo -n hello; sleep 1; echo " goodbye"
hello goodbye
@@ Command #0 terminated (pid: 11396, status: 0)
@@ Running command #1: ls -l
total 48
```

```
-rw-r--r-- 1 usuarioso usuarioso    66 ene  8 21:28 commands_b
-rw-r--r-- 1 usuarioso usuarioso    42 ene  8 21:25 commands_c
-rwxr-xr-x 1 usuarioso usuarioso 17144 ene  8 21:27 lanzamiento
-rw-r--r-- 1 usuarioso usuarioso  2401 ene  8 21:23 lanzamiento.c
-rw-r--r-- 1 usuarioso usuarioso  9472 ene  8 21:27 lanzamiento.o
-rw-r--r-- 1 usuarioso usuarioso   263 ene  8 21:27 Makefile
@@ Command #1 terminated (pid: 11403, status: 0)
@@ Running command #2: echo "done" ; false
done
@@ Command #2 terminated (pid: 11407, status: 256)

# Prueba apartado C
$ ./lanzamiento -s commands_c -b
@@ Running command #0: echo one
@@ Running command #1: sleep 6
@@ Running command #2: sleep 3
@@ Running command #3: echo two
@@ Running command #4: sleep 1
one
@@ Command #0 terminated (pid: 11576, status: 0)
two
@@ Command #3 terminated (pid: 11579, status: 0)
@@ Command #4 terminated (pid: 11580, status: 0)
@@ Command #2 terminated (pid: 11578, status: 0)
@@ Command #1 terminated (pid: 11577, status: 0)
```

Cuestión 2. (4,5 puntos) Implementar una variante del ejercicio 4 (cuestión A) de la práctica 4, que haga uso de diez hilos del mismo proceso –en lugar de diez procesos distintos– para crear de forma concurrente un fichero *output.txt* con el siguiente contenido:

```
00000111112222233333444445555566666777778888899999
```

El programa ha de crear 10 hilos desde el main(). Cada hilo recibirá un parámetro con su número de hilo correspondiente (del 0 al 9) que coincidirá con el orden de creación del hilo desde el programa principal. Cada hilo se encargará de escribir en el fichero su número de hilo 5 veces, en la posición del fichero indicada arriba. De este modo, el hilo 0 escribirá los 5 primeros bytes (cinco ceros), el hilo 1 los 5 siguientes (cinco unos), y así sucesivamente. Tras crear los hilos, el hilo main del programa se quedará bloqueado hasta que los demás hilos terminen, y a continuación imprimirá por pantalla el contenido final del fichero.

Para garantizar la evaluación individualizada de los distintos aspectos de la implementación, el programa se desarrollará de forma incremental, siguiendo las instrucciones de cada uno de los siguientes apartados:

a. **(0.5 puntos)** Crear una versión inicial del programa donde los hilos creados simplemente escriban por pantalla su número de hilo (pasado como argumento), y a continuación terminen. Es decir, en el código de este apartado no se realizará ningún procesamiento sobre el fichero.
b. **(2.5 puntos)** Modificar el apartado anterior para que los hilos generen el contenido del fichero escribiendo en paralelo en el mismo fichero, y sin usar ningún tipo de mecanismo de sincronización entre hilos. Es decir, cada hilo debe ocuparse de escribir los bytes que le corresponde ubicando el puntero de posición donde proceda antes de escribir en el fichero.
c. **(1.5 puntos)** Realizar una modificación sobre el código del apartado anterior, para que ahora se genere el contenido del fichero de forma cooperativa entre hilos usando un único descriptor de fichero compartido por todos los hilos. El hilo main del programa será el encargado de obtener este descriptor de fichero, compartido entre hilos. Nótese que para que cada hilo escriba su parte correspondiente en el fichero es necesario en este caso que los hilos se sincronicen. Los hilos se lanzarán en paralelo, aunque se debe garantizar que el contenido final del fichero sea el indicado más arriba. Para ello no es preciso establecer un orden específico en las escrituras (p.ej. la implementación podría permitir al hilo 1 escribir “su parte” del fichero antes que el hilo 0). Se deja al estudiante escoger los mecanismos de sincronización entre hilos para imponer el comportamiento deseado.