Práctica 1.2: Sincronización entre hilos

Índice

Objetivos

El principal objetivo de esta práctica es familiarizarse con los mecanismos de sincronización para hilos disponibles en Linux. Más concretamente, en los distintos ejercicios propuestos se hará uso de cerrojos, variables condicionales y semáforos sin nombre.

Ejercicios

Ejercicio 1: Problema de los filósofos comensales

Implementar dos soluciones al problema de los filósofos comensales, la primera usando mutexes y la segunda semáforos. La implementación con cada mecanismo debe incluirse en dos programas C independientes –*philosophers_mtx.c* y *philosophers_sem.c*–, cada uno con su propia función `main()`. Nótese que se proporciona un fichero *philosophers.c* como material suplementario. Este archivo debe utilizarse como punto de partida para construir ambas soluciones.

En este problema, cada filósofo se emula mediante un hilo independiente. Por lo tanto, el programa contará con 5 hilos, creados con `pthread_create()`. Para simplificar la implementación, se asignará un número (0..4) a cada tenedor y a cada filósofo.

Cada filósofo piensa y come de forma repetitiva. En concreto, la acción de *pensar* se simula mediante la función `sleep()` o `usleep()` (consulta la página de manual asociada para más detalles) con un periodo de tiempo aleatorio. El filósofo debe coger su tenedor izquierdo y derecho (no necesariamente en ese orden para evitar bloqueos...). Después de comer, el filósofo volverá a dejar los tenedores sobre la mesa, y luego dormirá durante algún tiempo antes de volver a pensar. Al implementar el programa, coger un tenedor se llevará a cabo adquiriendo un mutex o invocando la operación `wait()` en un semáforo, dependiendo del tipo de solución empleada. Del mismo modo, devolver un tenedor a la mesa se reduce a liberar un mutex o invocar la operación `signal()` de un semáforo.

Ejercicio 2: Inserción alternada en un vector con múltiples hilos

En este ejercicio se desarrollarán dos variantes de un programa multihilo con las siguientes características comunes:

- El programa principal creará un vector de enteros de tamaño especificado por el usuario (con la opción `-n <tamaño>`) reservando adecuadamente la memoria con `malloc()`. Este vector se declarará como variable global del programa.
- A continuación el programa creará una serie de hilos con `pthread_create()` y esperá a su terminación. Cuando los hilos hayan terminado el programa principal imprimirá por pantalla el contenido final del vector, con los valores separados por comas.
- Cada hilo creado realizará la inserción alternada de un valor concreto en el vector. La sincronización de la inserción alternada se implementará con semáforos POSIX sin nombre.

Parte A. Desarrollar un programa con las características arriba mencionadas y que cree 2 hilos. Estos dos hilos insertarán los numeros 1..N de forma alternada en el vector compartido. En particular, el hilo 1 insertará los números impares, y el hilo 2 los números pares. De esta forma, la impresión por pantalla del vector será como se indica en el ejemplo de ejecución:

```
$ ./alternateA -n 10
1,2,3,4,5,6,7,8,9,10
```

Parte A. Desarrollar un programa con las características arriba mencionadas y que cree 3 hilos. Estos tres hilos insertarán los numeros 1..N de forma alternada en el vector compartido. El orden de inserción será el siguiente: hilo 1, hilo 2, hilo 3, hilo 1, hilo 2, hilo 3, y así sucesivamente. La salida por pantalla deberá ser la misma que en el ejemplo anterior, alterándose solamente el mecanismo de inserción de los números en el vector en el código de los distintos hilos.

Ejercicio 3: Productor-consumidor con acceso a ficheros

Desarrollar un programa `pc-files` con el siguiente modo de uso:

```
$ ./pc-files [ -i fichero_entrada ] [ -o fichero_salida ]
```

Nótese que ninguna de las opciones es obligatoria. En el caso de que se no se especifique la opción `-i`, el fichero de entrada por defecto considerado será la entrada estándar (ya abierto y con descriptor ya predefinido `stdin` en la biblioteca estándar). En caso de no pasar la opción `-o` en la línea de comando, el fichero de salida predefinido será *out.txt*. Si el fichero de salida especificado ya existe su contenido se truncará; en caso contrario, se creará dicho fichero.

El programa `pc-files` creará dos hilos POSIX, que se comportarán respectivamente como productor y consumidor. Ambos hilos se comunicarán usando un buffer compartido (global) de cadenas de caracteres con capacidad para 4 cadenas:

```
#define MAX_SBUFFER_SIZE 4
char* shared_buffer[MAX_SBUFFER_SIZE];
```

El productor leerá cadenas de caracteres terminadas en `'\n'` del fichero de entrada, y enviará cada cadena al consumidor usando el buffer compartido, así hasta detectar fin de fichero. La inserción se realizará añadiendo al buffer el puntero al comienzo de la cadena, cuya memoria se reservará con `malloc()` desde el productor. Para indicar al consumidor que no hay más cadenas que procesar, el productor insertará NULL en el buffer compartido.

El consumidor extraerá cadenas de este buffer compartido, las imprimirá en un fichero de salida en orden de recepción y liberará la memoria con `free()`. El hilo consumidor terminará al extraer la cadena NULL del buffer. El programa principal terminará cuando ambos hilos hayan acabado. Para la lectura por líneas del fichero se recomienda el uso de la función `fgets()` de la biblioteca estándar (consultar su página de manual).

Para implementar las restricciones de sincronización típicas del productor consumidor ha de usarse un mutex, variables condicionales, y otras variables compartidas necesarias para el correcto acceso al buffer compartido.

Ejemplo de ejecución:

```
## Creación de fichero de entrada manualmente
osuser@debian:~/ejercicio3$ echo London > in.txt
osuser@debian:~/ejercicio3$ echo Barcelona >> in.txt
osuser@debian:~/ejercicio3$ echo Madrid >> in.txt
osuser@debian:~/ejercicio3$ echo Berlin >> in.txt
osuser@debian:~/ejercicio3$ echo Lisbon >> in.txt
osuser@debian:~/ejercicio3$ echo Paris >> in.txt
osuser@debian:~/ejercicio3$ cat in.txt
London
Barcelona
Madrid
Berlin
Lisbon
Paris

## Ejecución
osuser@debian:~/ejercicio3$ ./pc-files -i in.txt
osuser@debian:~/ejercicio3$ cat out.txt
London
Barcelona
Madrid
Berlin
Lisbon
Paris
osuser@debian:~/ejercicio3$
```