

## Prácticas Concurrency y Distribución (20/21)

Arno Formella, Anália García Lourenço, Noelia García Hervella

semana 5 abril – 9 abril

### Práctica 5: Sincronización y exclusión

**Objetivos:** Objetos compartidos entre procesos con acceso sincronizado. Este ejercicio pretende reiterar el concepto de compartir objetos entre hilos y utilizar una referencia del objeto compartido para controlar el acceso de los procesos a las secciones críticas, es decir, a código que en un momento de tiempo, solamente un hilo puede ejecutar.

1. Implementa una clase `ClassA` que contenga un solo método `EnterAndWait()`. Este método debe hacer lo siguiente y en este orden:
  - a) imprimir un mensaje indicando cual es el hilo que está comenzando a ejecutarlo;
  - b) esperar unos segundos; y
  - c) imprimir otro mensaje indicando el hilo que está acabando de ejecutar el método.
2. Implementa una clase `ClassB` que implemente `Runnable` y que se construya recibiendo como parámetro un objeto de la clase `ClassA`. Haz que en su método `run()` simplemente llame al método `EnterAndWait()` del objeto con cual ha sido construido.
3. Implementa una clase principal con un método `Main` en el que se crea un único objeto de la clase `ClassA` y varios objetos de la clase `ClassB` a los que se les pasa a todos como parámetro el objeto de la clase `ClassA`. Después se crearán y ejecutarán el mismo número de hilos (objetos de la clase `Thread`) que de objetos de tipo `ClassB` tengamos pasándoles como parámetros los objetos de la clase `ClassB`, de tal forma que cada método `run()` de cada objeto de la clase `ClassB` se ejecute en un hilo diferente.
4. Análiza: ¿Cuál es el resultado? ¿Cuántos hilos pueden estar simultáneamente ejecutando el método `EnterAndWait()`?

5. Limitamos ahora el acceso a la sección crítica (el método `EnterAndWait()` en nuestro caso) utilizando sincronización (el mecanismo empleando `synchronized` o `lock` que vimos la semana pasada).
6. Modifica el código para que solo un hilo pueda estar ejecutando el método `EnterAndWait()` en cualquier instante. Realiza dos versiones: una con `synchronized` y otra con un cerrojo (`lock`). Explica lo que observas.
7. Aumenta el código de las dos clases `ClassA` y `ClassB` que ambas clases contengan un contador con el objetivo que el contador de la clase compartida se decremente y el contador de la class no compartida se incrementa.
8. Modifica el método `EnterAndWait()` para que sea un `EnterAndDecrement()`, es decir, un método sincronizado que decremente el contador y devuelve `false` si ha llegado a 0 (`true` si no).
9. Usa ahora un bucle en el método `run()` de la clase `ClassB` que entre tantas veces en el método `EnterAndDecrement()` que sea posible y que cuente las veces conseguidos (con su contador), por ejemplo con un código como:
 

```
while(A.EnterAndDecrement()) { cnt.Increment(); }
```
10. Imprime al final de `run()` cuantas veces se ha conseguido decrementar el contador compartido realizando experimentos con diferentes números de hilos y inicializaciones del contador compartido.
11. Para la preparación de la siguiente semana: intenta averiguar el funcionamiento de los métodos `wait()` y `notify()` (respectivamente `notifyAll()`), sobre todo: para que sirven...
12. Para la preparación de la siguiente semana—¡reutilizaremos el código!—aumenta tu código para que el hilo principal tenga acceso a los valores finales de los contadores de acceso de cada hilo (por ejemplo cada hilo copia su valor al final de su `run()` en una posición determinada de un array compartido). El hilo principal imprime algunas estadísticas de este array, por ejemplo, valor mínimo, valor máximo, valor media.