# Prácticas Concurrencia y Distribución (17/18)

Arno Formella, Anália García Lourenço, Lorena Otero Cerdeira, David Olivieri

semana 5 – 11 marzo

## 4. Semana 4 (5/03–11/03): Sincronización de hilos

**Objetivos:** Sincronización de hilos

1. **(P4: para entregar en grupo de práctica)**: Contador sincronizado.

   El objetivo de este problema es escribir un contador concurrente que sincroniza el acceso de un grupo de hilos a un acumulador. Sigue estos pasos:

   *a*) **Configuración del problema:** crea una clase de hilo vacío (por el momento) (`MiThread`) que *extends* `Thread` (o *implements* `Runnable`). También crea una clase *principal* (por ejemplo, `MiProblema`) donde dentro de su método *main* crea una *array* (o `ArrayList`) del objeto `MiThread`.

   *b*) **Clase Contador:** Elabora una clase `Contador` que tenga un método `Incrementar(n)` que ejecute un bucle de $n$ iteraciones que incrementa una variable interna (un acumulador total) cada cierto tiempo, y al acabar devuelva el valor actual de dicha variable.

   *c*) **Threads:** Desde el método principal, lanza varios hilos que comparten un objeto de tipo `Contador`. Después de esperar un intervalo aleatorio de tiempo (para proporcionar una variación de salida), cada hilo debe llamar al método `Incrementar(n)` con `n` (también elegida de manera aleatoria), y obtener/imprimir el valor del acumulador total devuelto por ese método. Explica el comportamiento de la salida.

   *d*) **Bloques sincronizados:** Ahora construye un bloque sincronizado para la operación de `Incrementar` (el bloque con la palabra reservada `synchronized` debe actuar sobre el objeto contador compartido para ejecutar un conjunto/bloque de código). ¿Cómo cambia esto el comportamiento del contador?

   *e*) **Utilizando Java `AtomicInteger/LongAdder`:** Reemplaza el código anterior para usar un `AtomicInteger` y/o `LongAdder`. Para realizar la operación del incremento, usa un método adecuado del gran conjunto disponible para estos objetos (consulta la documentación de la API de Java). Explica lo que observas.

2. **(P3: para entregar dentro de una semana)**: Más sobre sincronización.

   El propósito de este problema es estudiar más a fondo la sincronización de hilos utilizando el código del problema 1.

   *a*) **Java Locks:** En lugar de bloques sincronizados, la API de Java tiene un conjunto de objetos optimizados de alto nivel para concurrencia eficiente. En particular, aquí vas a utilizar el paquete `java.util.concurrent.locks`, que proporciona un mecanismo para proporcionar exclusión mutua similar al método sincronizado. Reemplaza el bloque sincronizado con un `lock`.

*b*) **Análisis/Medidas:** Ejecuta el código del contador para todos los métodos de sincronización empleados variando la cantidad de hilos utilizados. ¿Puedes notar una diferencia en el rendimiento? Explica tus resultados.

*c*) **Between Atomic Operations:** El código anterior ilustra que las operaciones atómicas están protegidas. Sin embargo, ahora considera la adición de dos contadores, $p$ y $q$, donde el acumulador es: $q \longleftarrow q + p$. Este problema demuestra que las operaciones atómicas están protegidas, mientras que las operaciones entre sí no están. Modifica tu código para resolver este problema de calcular correctamente $q$, usando la sincronización según corresponda.