Programación Concurrente y de Tiempo Real*
Grado en Ingeniería Informática
Asignación de Prácticas Número 9

En esta asignación aplicará control de exclusión mutua y sincronización, utilizando para ello el API de alto nivel de Java, a diferentes situaciones que se le plantean. Documente todo su código con etiquetas (será sometido a análisis con `javadoc`). Si lo desea, puede también agrupar su código en un paquete de clases, aunque no es obligatorio.

1. Enunciados

1. Se desea modelar una carrera de triatlón utilizando un 100 tareas `Runnable` para simular a los cien participantes de la prueba. Cada participante modelado por una tarea realizará una secuencia de ejecución compuesta de las siguientes fases:

- `Posta de natación`
- `Posta de carrera ciclista`
- `Posta de carrera a pie`

La duración de cada una de las postas para cada concursante estará modelada por una espera de la tarea determinada por un número aleatorio como parámetro del método `sleep(int)`. Cada tarea toma tiempos cuando inicia y acaba cada posta, y acumula los tiempos totales para el triatlón completo en un array de tiempos de 100 ranuras; una por cada tarea. Cada posta tiene una salida neutralizada, de forma que todos los hilos deben llegar a la meta antes de que se inicie la posta siguiente. Modele esta condición mediante los objetos de clase CyclicBarrier que sean necesarios. Una vez terminado el triatlón, el programa principal lee el array de tiempos totales, determina el menor de ellos, e imprime cuál es el concursante (tarea) ganador. Guarde el código del programa en `triatlonBarreras.java`.

2. Utilizando cerrojos de clase `ReentrantLock` y colas `Condition`, adapte con estas herramientas el monitor de los lectores/escritores que utilizó prácticas anteriores. Renombre al monitor como `RWMonitorAN.java`. Utilícelo de nuevo para acceder de forma controlada a algún recurso compartido. Haga lo mismo con el monitor que escribió para controlar un sistema de impresoras (`ImpMonitor.java`), y con el monitor de simulación de un semáforo general (`semafMonitor.java`).

3. Utilice la solución de sockets con servidor multihebrado mediante pool de threads para escribir un servidor que actúe como contador de páginas web. Cada petición de un cliente remoto incrementa en uno el contador que el servidor mantiene. Controle la exclusión cuando sea necesario, y escriba dos versiones de todo el código, que hagan ese control mediante:

- Un objeto de clase `AtomicLong`. Fichero `atomicServer.java`.
- Un cerrojo de clase `ReentrantLock`. Fichero `reentrantServer.java`.

El cliente lo guardará en `clienteContador.java`.

4. Escriba una solución al problema de los filósofos utilizando un monitor que emplee los recursos del API de alto nivel. Guarde el monitor en `filoApiAN.java`. Escriba una tarea `Runnable` que modele a los cinco filósofos, y actívelos mediante un ejecutor de capacidad fija. Guarde todo esto en `usaFiloApiAn.java`.

5. Utilizando la clase `Semaphore` del API de alto nivel, implemente una solución al problema del productor-consumidor, y guárdela en `prodCons.java`.

6. Deseamos conocer el rendimiento en tiempo de ejecución de las siguientes técnicas de control de exclusión mutua en java: cerrojos `synchronized`, semáforos de clase `Semaphore`, cerrojos de clase `ReentrantLock`, y objetos `atomic`. Escriba un código (guárdelo en `comparativa.java`) similar al siguiente pseudocódigo:

```
public long criticalSection(long iter){
ini=activar-cronometro;
for(long i=0; i<iter; i++){
pre-protocolo;
  n++; //seccion critica
post-protocolo;
}
fin=parar-cronometro;
return(fin-ini);
}
```

Los protocolos de acceso y salida de la sección crítica los construirá con las técnicas de control de e.m ya citadas, y hará pruebas con un número creciente de iteraciones. Tome tiempos y construya una gráfica que incluya las curvas de tiempo para cada técnica de control como una función del número de iteraciones. Esa curva le dirá que técnica es más rápida. Guárdela en `curva.pdf`.

2. Procedimiento y Plazo de Entrega

Se ha habilitado una tarea de subida en *Moodle* que le permite subir cada fichero que forma parte de los productos de la práctica de forma individual en el formato original. Para ello, suba el primer fichero de la forma habitual, y luego siga la secuencia de etapas que el propio *Moodle* le irá marcando. Recuerde además que:

- Los documentos escritos que no sean ficheros de código deben generarse **obligatoriamente** utilizando Latex, a través del editor *OverLeaf*, disponible en la nube. Tiene a su disposición en el Campus Virtual un manual que

le permitirá desarrollar de forma sencilla y eficiente documentos científicos de alta calidad. Puede encontar el citado manual en la sección dedicada a Latex en el bloque principal del curso virtual. El url de *OverLeaf* es: `https://www.overleaf.com/`

- No debe hacer intentos de subida de borradores, versiones de prueba o esquemas de las soluciones. *Moodle* únicamente le permitirá la subida de los ficheros por **una sola vez**. utilizará paralelismo de grano grueso, particionando un problema y definiendo el número de hilos necesarios para su resolución en función de ecuaciones de balanceado de carga.

- La detección de plagio (copia) en los ficheros de las prácticas, o la subida de ficheros vacíos de contenido o cuyo contenido no responda a lo pedido con una extensión mínima razonable, invalidará plenamente la asignación, sin perjuicio de otras acciones disciplinarias que pudieran corresponder.

- El plazo de entrega de la práctica se encuentra fijado en la tarea de subida del Campus Virtual.

- Entregas fuera de este plazo adicional no serán admitidas, salvo causa de fuerza mayor debidamente justificadas mediante documento escrito.

- Se recuerda que la entrega de todas las asignaciones de prácticas es recomendable, tanto un para un correcto seguimiento de la asignatura, como para la evaluación final de prácticas, donde puede ayudar a superar esta según lo establecido en la ficha de la asignatura.