Programación Concurrente y de Tiempo Real*
Grado en Ingeniería Informática
Asignación de Prácticas Número 6

Se le plantean a continuación un conjunto de ejercicios sencillos de programación sobre paralelismo con control de la exclusión mutua y uso de ejecutores, que debe resolver de forma individual como complemento a la séptima sesión práctica. Para cada uno, debe desarrollar un programa independiente que lo resuelva. 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. Ejercicios

1. En la carpeta de la práctica se le proporciona el código necesario para implantar una solución cliente-servidor simple multihebrada. El inconveniente de esta solución es la necesidad de crear un *thread* para dar servicio a cada petición de un cliente recibida por el servidor. Una solución mucho más elegante es modelar la tarea de servicio mediante objetos `Runnable`, y delegar su ejecucion a un *pool* de *threads*. Se pide:

- Reescriba el código del servidor de forma que cada petición de servicio sea atendida por una hebra que procesará un objeto de clase `ThreadPoolExecutor`, y guárdelo en `ServidorHiloconPool.java`. Escriba otra versión equivalente sin usar ejecutores pero sí multihebrada y guárdela en otro fichero de nombre `ServidorHilosinPool.java`.
- Escriba ahora una versión del cliente que genere un número fijo de peticiones al servidor que irán creciendo, y guárdela en `clienteMultiple.java`.
- Tome los tiempos que ese cliente necesita para que sus peticiones sean procesadas en el servidor multihebrado con y sin utilizar el ejecutor.

*©Antonio Tomeu

Escriba un documento (recuerde que el uso de *OverLeaf* es obligatorio) llamado `analisis.pdf` donde muestre mediante una curva la evolución del tiempo como una función del número de peticiones.

2. Deseamos que varios hilos escriban en un fichero de disco externo de clase `RandomAccesFile` de forma paralela. Dicha escritura debe efectuarse bajo exclusión mutua sobre el fichero. Provea una solución llamada `ficheroSeguro.java` que cumpla con la especificación descrita, utilizando cerrojos `synchronized`.

3. Volvamos nuevamente al cálculo del valor de la integral (en este caso para la función seno en el intervalo $[0, 1]$) mediante un método de Monte-Carlo, que ya desarrolló en la Práctica Número 1 en una versión secuencial. Se le pide:

- Escribir una versión paralela de grano grueso utilizando la Ecuación de Subramanian, que lleve la cuenta del número de puntos en un único contador compartido, al que se accederá bajo exclusión mutua. Llámela `intParalelauniCont.java`. Tome tiempos de ejecución para diferente número de puntos.

- Repita el ejercicio anterior definiendo contadores locales a cada tarea para, de este forma, reducir la sobrecarga inherente a los bloqueos de exclusión mutua. Llámela `intParalelomultiCont.java`. Tome tiempos de ejecución para diferente número de puntos.

- Repita el ejercicio anterior definiendo cada tarea mediante una clase que implemente a la interfaz `Callable` y recogiendo los resultados utilizando la interfaz `Future<V>`. Llámela `intParaleloFutureCont.java`. Tome tiempos de ejecución.

- Trace ahora curvas utilizando el número de puntos de la aproximación al valor de la integral como variable independiente, y compruebe cómo la segunda versión es más eficiente que la primera. De la tercera no decimos nada. Usted nos los dirá a nosotros. Incluya sus curvas y su análisis en un documento (recuerde que el uso de *OverLeaf* es obligatorio) llamado `analisis2.pdf`.

4. Construya ahora una clase `cuentaCorrienteSegura.java` que sea segura frente a concurrencia utilizando métodos `synchronized`. Escriba utilizando concurrencia soportada mediante expresiones λ tareas que accedan de manera concurrenta a un objeto de la clase anterior para realizar ingresos y

reintegros y guarde el código que las procesa a través de un ejecutor de tamaña variable en `simulRedCajeros.java`. El programa mostrará los saldos inicial y final. Compruebe que son coherentes.

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**.

- 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.