Programación Concurrente y de Tiempo Real*
Grado en Ingeniería Informática
Asignación de Prácticas Número 4

Las primeras estrategias para el control de la exclusión mutua se basaron en algoritmos que compartían información utilizando varibles comunes. Ello permitía que si un hilo deseaba ejecutar su sección crítica, pudiera saber qué hacían los demás antes de acceder a ella, y esperar en caso negativo mediante un bucle de espera activa, en el cual se está comprobando continuamente el estado de esas variables hasta encontrar la situación adecuada para el ingreso en la sección crítica. En esta práctica se le pide, al objeto de que se familiarice con esta clase de algoritmos de control de e.m. y la estregia que emplean, que realice la implementación de varios de ellos. 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

La literatura sobre programación concurrente siempre estudia el Algoritmo de *Dekker* y similares partiendo de un enfoque de refinamiento sucesivo de cuatro intentos incorrectos que utilicen alguna o ambas de las siguientes técnicas de sincronización, junto con el mecanismo de espera ocupada:

- ejecución de la sección crítica por turnos
- ejecución de la sección crítica mediante flags de aviso

En las referencias bibiográficas propuestas para el Tema 4 del curso es posible encontrar descripciones en pseudocódigo de estos intentos incorrectos y de los algoritmos que funcionan, así como análisis de su comportamientos y peculiaridades de cierta profundidad. Recomendamos por tanto la lectura, con carácter previo al desarrollo de la práctica, de al menos los capítulos dedicados a algoritmos de exclusión mutua con variables compartidas descritos en lassiguientes obras:

- Ben-Ari, M. Principles of Concurrent and Distributed Programming, 2ª edición.

*©Antonio Tomeu

- Palma et al., Programación Concurrente, 2ª edición.

Durante la clase de prácticas se expondrán las dos primeras etapas del refinamiento sucesivo mediante código en Java que los soportan y que utilizan el primero la técnica de los turnos, y el segundo la técnica de los flags de aviso. **En todos los casos, la sección crítica será un incremento de una variable compartidas por las tareas**.

1. Implemente, a partir de las referencias bibliográficas indicadas, las etapas tercera y cuarta del refinamiento sucesivo, y guárdelas en `tryThree.java` y `tryFour.java`; escriba un corto documento `analisis.pdf` utilizando LaTeX con *OverLeaf* que recoja el comportamiento obtenido y su interpretación del mismo. Utilice herencia de la clase `Thread`.

2. Nuevamente, a partir de las referencias bibliográficas indicadas, implemente el algoritmo de *Dekker* para dos procesos. Guarde su trabajo en `algDekker.java` y añada al documento `analisis.pdf` su intepretación de los resultados de su ejecución. Utilice herencia de la clase `Thread`.

3. Escriba un programa que implemente el Algoritmo de *Eisenberg-McGuire* para dos procesos. Cree dos hebras (que deberán compartir las variables de control comunes por implementación de la interfaz `Runnable`, y guarde su código en `algEisenbergMcGuire.java`. Utilice un ejecutor de tamaño fijo. Dedique un apartado del documento `analisis.pdf` a este algoritmo que recoja el comportamiento obtenido y su interpretacióndel mismo.

4. Una solución válida para el problema de la exclusión mutua con n procesos es el algoritmo de *Lamport*. Escriba un programa llamado `algoLamport.java` que lo implemente para n procesos utilizando objetos que implementen la interfaz `Runnable` y un ejecutor de tamaño fijo para procesarlos. Dedique un apartado del documento `analisis.pdf` a este algoritmo que recoja el comportamiento obtenido y su interpretacióndel mismo.

5. Una solución incorrecta con variables comunes para el problema de la sección crítica fue el algoritmo de *Hyman*. Impleméntelo utilizando tareas soportadas por expresiones λ y analice su comportamiento en tiempo de ejecución. Guarde su código en `algHyman.java`. Dedique un apartado del documento `analisis.pdf` a este algoritmo que recoja el comportamiento obtenido y su interpretacióndel mismo.

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.