Prácticas Concurrencia y Distribución (18/19)

Arno Formella, Anália García Lourenço, Hugo López Fernández, David Olivieri

semana 28 febrero – 12 marzo

2. Semana 6 (28/02–12/03): Sincronización y exclusión II

La semana pasada, se probó como usar el mecanismo de `wait()` y `notify()` / `notifyAll()` para controlar la planificación de hilos dada alguna condición. Ahora, en este laboratorio, queremos llevar esta idea más allá y ordenar los hilos. Esto se hará de dos maneras: 1) utilizando `notifyAll()`, donde cada hilo se despierta y necesita verificar la condición de acceso a la sección crítica (es decir, si es su turno), o 2) directamente usando una referencia al siguiente hilo y despertando solo a dicho hilo mediante `notify()` para que sea el siguiente en acceder.

1. **(P4: para entregar en grupo de práctica)**:
   Objetivo: el objetivo de este ejercicio es que haya dos hilos alternándose en el acceso al método `enterAndWait()` de un objeto de clase `ClassA`.

   Configuración del problema:

   a) **Reutilización de código:** recupera el código de `ClassA` y `ClassB` del apartado 1 de la práctica de la semana pasada (5ª semana), en que se sincronizaba el acceso de varios hilos a la sección crítica (método `enterAndWait()`).

   b) **Modificación de ClassB:** Ahora, la sincronización (llamadas a `wait()` y `notify()` debe hacerse empleando las referencias a los objetos `Thread` establecidas. Modifica la clase `ClassB` de manera que: 1) tenga un método
   `setThread(Thread t)` que permita establecerle una referencia a otro hilo, 2) hacer que ejecuten el código del método `run()` de manera infinita (hasta que hayan sido interrumpidos). Después, dentro del bucle de ejecución indefinida, los hilos harán lo siguiente: 1) esperar a ser notificados para ejecutar el método `enterAndWait()`, 2) ejecutar dicho método y 3) notificar al otro hilo.

   c) **Main:** Implementa una clase Main con un método principal en el que se crea un único objeto de la clase `ClassA` y dos objetos de la clase `ClassB` a los que se les pasa como parámetro el objeto de la clase `ClassA` creado. Después se crearán dos hilos (objetos de la clase `Thread`) para que ejecuten los objetos de tipo `ClassB` creados. A cada objeto de la clase `ClassB` se le debe de establecer una referencia al otro hilo (al que no lo ejecuta).

   Tras lanzar los dos hilos, el método principal debe esperar a que ambos se encuentren en estado de espera para despertar a uno de ellos y que comience el intercambio de ejecuciones del método. Pasado un tiempo, el método principal debe interrumpir los hilos.

2. **(P3: para entregar dentro de una semana)**: Objetivo: en el código `enterAndWait()` de la semana pasada, cualquier hilo podía entrar en la sección crítica (con un poco más de restricción asociada a los hilos rojos/negros). Ahora, queremos que los hilos entren dado un orden específico. Haremos esto de

dos formas diferentes: 1) utilizando `notifyAll()`, que es un método menos eficiente ya que despierta a todos que están esperando, y 2) notificando solo al hilo al que toca su turno. Consulta la figura a continuación para la notificación de llamada explícita en el orden de hilos:

Sigue estos pasos:

a) Configuración del problema: Usa tu código del problema `enterAndWait()` de la semana pasada.

b) Orden secuencial (método de fuerza bruta): utilizando directamente el código de la semana pasada, añade el código necesario en los hilos para ejecutar la sección crítica en orden con llamadas usando `notifyAll()`. Pista: tendrás que asignar a los hilos un número que indique su orden, de manera que este orden pueda ser comprobado antes de ejecutar el método.

c) Orden secuencial (método explícito/eficiente): utilizando la idea de la Figura 2, escribe el código apropiado para que el orden de los hilos que ejecutan la sección crítica se realice explícitamente mediante llamadas al siguiente hilo en la lista.

d) Análisis: Compara el tiempo de ejecución del método de fuerza bruta (parte b) con el método explícito (parte c). Haz un plot y describe tus observaciones en palabras.