Notas acerca de la sincronización con candados en Java.

Armando Ballinas Nangüelú.
armballinas@gmail.com

26 de abril de 2015

1. Introducción.

La sincronización de los procesos en el problema del productor-consumidor se puede resolver usando los monitores nativos de Java mediante los métodos y bloques `synchronized` así como haciendo que los consumidores esperen por una notificación mediante el método `Object.wait()`.

En las siguientes secciones se describe una manera de lograr una sincronización más eficiente mediante el uso de candados (*locks*) y condiciones (*conditions*).

2. Candados en Java.

Los candados son una primitiva de sincronización que se basa en la idea de obtener un permiso para leer o escribir un recurso compartido. En este caso el permiso lo obtiene el hilo que *adquiera* el candado. El hilo con el candado puede acceder al objeto compartido sin restricciones hasta que *libere* el candado.

Cuando algún hilo tiene el candado ningún otro hilo puede modificar el objeto compartido pues para lograr esto se necesita primero adquirir el candado. Una vez que el candado ha sido liberado los demás hilos pueden tratar de obtener el candado de nuevo.

En java la noción de candado está definida mediante la interfaz `Lock` que se encuentra en el paquete `java.util.concurrent.locks`. La implementación más simple y utilizada es `ReentrantLock`.

2.1. La clase ReentrantLock.

Esta clase implementa la interfaz `Lock` y define métodos para adquirir y liberar el candado. También permite definir condiciones de espera para los hilos mediante el objeto `Condition`.

El método `lock()` permite adquirir el candado. Si el candado no está siendo usado por ningún hilo, entonces este método termina inmediatamente y el hilo que lo invocó *obtiene* el candado. Por otro lado, como la clase implementa los candados permitiendo la propiedad *reentrante* entonces si este hilo ya tiene el candado y lo vuelve a solicitar no ocurrirá nada. Finalmente, si otro hilo ya tiene el candado, entonces este se bloqueará hasta que lo intente obtener.

Otro método para intentar obtener el candado es el método `tryLock()`. Este método también sirve para adquirir el candado pero solo si está disponible. Si el candado está tomado por otro hilo, entonces este método devuelve `False` inmediatamente. Este método es útil cuando el hilo puede hacer otras acciones, en particular si no puede obtener el candado puede esperar en una *condición* para ser despertado después.

El método `unlock()` sirve para liberar el candado. Este método solo lo debe ejecutar el hilo que posea el candado pues de lo contrario se lanzará una excepción.

2.2. Condiciones de espera.

En Java se pueden definir distintas condiciones de espera que funcionan de manera similar a los métodos `wait()` y `notify()`. Los objetos `Condition` se pueden asociar a un candado en particular mediante el método `Lock.getCondition()`. Este método devuelve un objeto `Condition` asociado a este objeto.

Un hilo puede esperar en una condición invocando el método `await` del objeto `Condition` de manera similar a cuando espera ejecutando el método `wait()`. Un hilo puede despertar a hilos esperando en una condición ejecutando los métodos `signal()` y `signalAll()` de la clase `Condition`. Estos métodos son similares a los métodos de `Object` `notify()` y `notifyAll()`.

3. Sincronización fina en el problema productor-consumidor.

Para permitir que los hilos puedan encolar y decolar simultáneamente se pueden usar dos candados, uno para encolar y otro para decolar. Para evitar *deadlock* se debe dar prioridad a la operación de encolar.

Si la cantidad de elementos en la cola es al menos dos, entonces el productor solo necesita obtener el candado de encolar y el consumidor el candado de decolar. Si la longitud es uno o cero, entonces ambos tipos de hilos necesitan los dos candados, sin embargo, si el consumidor no puede adquirir el segundo (considerar el método `tryLock()`) entonces debe liberar su candado para que los productores puedan obtenerlo.

Además se debe constuir una condición donde esperarán los consumidores cuando la cola esté vacía. Cada que un productor introduzca un objeto puede invocar el método `signalAll()` de la condición para despertar a lo consumidores dormidos mediante el método `await()`.

Esta sincronización es más fina pero requiere de más cuidado para ser implementada por lo que a los que lo hagan se les otorgarán **3 puntos extra**.