Prácticas Concurrencia y Distribución (17/18)

Arno Formella, Anália García Lourenço, Lorena Otero Cerdeira, David Olivieri

semana 12 – 18 marzo

5. Semana 5 (12/03–18/03): Sincronización y exclusión

1. **(P4: para entregar en grupo de práctica)**:
   Objetos compartidos y acceso sincronizado.

   Objetivo: Este ejercicio pretende reiterar el concepto de compartir objetos entre hilos y utilizar la referencia del objeto compartido para controlar el acceso de subprocesos a las secciones críticas.

   Configuración del problema:

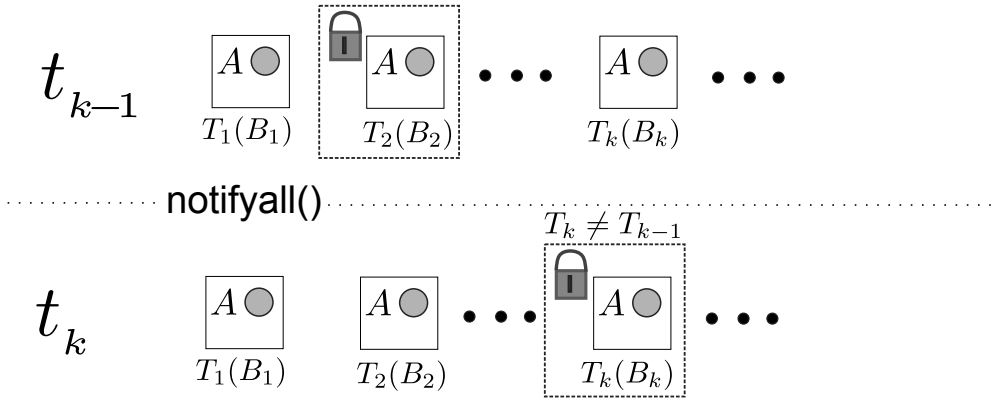
   a) **Class A:** Implementa una clase A que contenga un solo método `EnterAndWait()`. Este método debe hacer lo siguiente, en este orden:
      1) imprimir un mensaje indicando cual es el hilo que está comenzando a ejecutarlo;
      2) luego se detendrá durante unos segundos; y
      3) vuelve a imprimir otro mensaje indicando el hilo que está acabando de ejecutar el método.

   b) **Class B (MyThread):** Realiza una clase B que implemente `Runnable`, que reciba como parámetro un objeto de la clase A, y en el método `run()` simplemente llame al método `EnterAndWait()` de ese objeto.

   c) **Main Class:** Una clase con un método `main()`, en el que se crea un objeto de la clase A, y varios objetos de la clase B a los que se les pasa a todos como parámetro el mismo objeto de la clase A. Es decir,
      1) todos los objetos de la clase B compartirán el mismo objeto de la clase A.
      2) Después se crearán y ejecutarán el mismo número de hilos que de objetos de tipo B tengamos, pasándoles como parámetros los objetos de la clase B, de forma que cada método `run()` de cada objeto de clase B se ejecute en un hilo diferente.

   d) **Análisis:** ¿Cuál es el resultado? ¿Cuántos hilos pueden estar simultáneamente ejecutando el método `EnterAndWait()`?

   e) **Acceso limitado a la sección crítica:** Utilizando sincronización, modifica el código para que solo un hilo pueda estar ejecutando el método `EnterAndWait()` en cualquier instante. Explica lo que observas.



2. **(P3: para entregar dentro de una semana)**: Exclusión condicional en el acceso a secciones críticas.

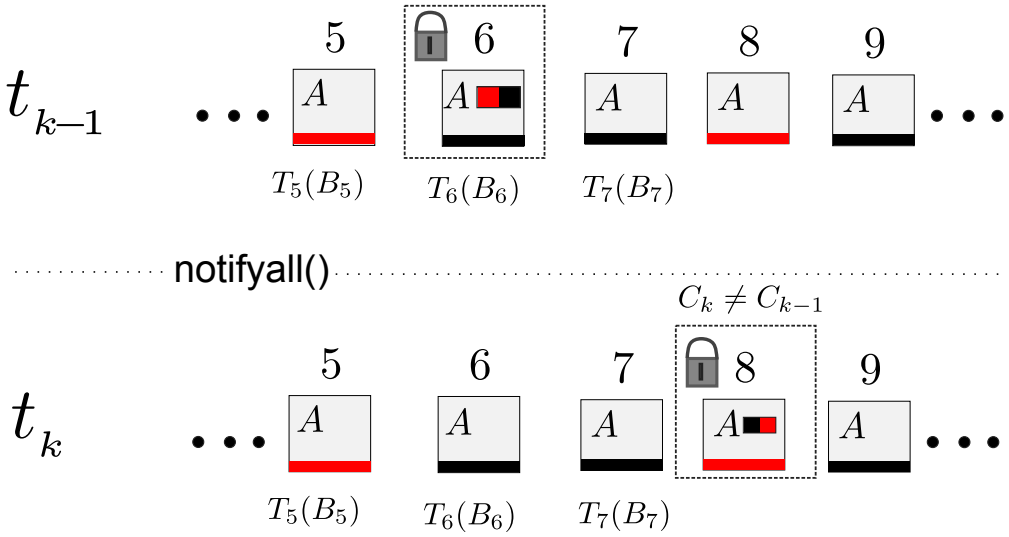
   Objetivo: El propósito de este ejercicio es comprender cómo controlar el acceso de hilos a *secciones críticas*. En el problema 1, el método sincronizado limitó el acceso a una sección crítica independientemente de cualquier otra condición. Mientras que todos los hilos compiten continuamente para entrar, es el sistema operativo el que finalmente decide cual es el siguiente hilo. Aquí queremos controlar ese acceso de forma explícita dadas ciertas condiciones mediante el mecanismo `wait`/`notifyAll` de Java.

   a) **Exclusión condicional:** La condición para este ejercicio es que solo un hilo puede ejecutar la sección crítica y una vez terminado, el siguiente hilo no puede ser el mismo hilo. Reutiliza el código del problema anterior con las modificaciones:

      1) **Modifica Class A:** agrega dos atributos enteros: `Nactual`, un contador que indica el número de veces que se ha entrado en `EnterAndWait()`, y `nhUltimo`, que indica el último hilo que ha accedido a la sección crítica (utilizado para garantizar que el siguiente hilo sea cualquier menos `nhUltimo`, y que luego debe actualizarse con el hilo actual que ingresa). `Nactual` debe comenzar en un valor inicial y disminuir a cero, lo que indica el final del programa.
      2) **Modifica Class B:** Modifica la sección de ejecución del subproceso para implementar la condición de exclusión condicional con `wait`/`notifyAll`. Los hilos que no están en la sección crítica, están en un estado de espera. Agrega cualquier otra variable o código necesario.
      3) **Modifica el programa principal:** Modifica el código del main de manera apropiada. Ejecuta el código con `M` hilos y `N` accesos totales a la sección crítica de `EnterAndWait()`; por ejemplo, `N=20` veces, mientras que `M=6` hilos. Comprueba que funciona como se espera.

   b) **Exclusión condicional con rojo/negro alternando:** En este problema, tu programa debería controlar los hilos basándose en una condición adicional: su color.

      1) Asigna un atributo de `Color` (C) (tipo `String`) a cada hilo (la clase B), que puede tomar el valor de 'rojo' o 'negro'.
      2) Modifica el código del main para asignar a cada hilo un color, elegido al azar.
      3) Modifica Class A y Class B de manera apropiada. Para seleccionar el siguiente hilo (después de una llamada a `notifyAll()`), la regla es que el color del hilo debe alternar entre las iteraciones (por ejemplo, si el hilo anterior fue rojo, el siguiente hilo con acceso a la sección crítica debe ser negro). Esto se ilustra en la siguiente figura, donde una posible solución es almacenar el atributo de color *anterior* y *actual* en la variable compartida A, que se utiliza para comparar quién puede ingresar a la sección crítica.


t_{k-1}
5
6
7
8
9
A
$T_5(B_5)$
$T_6(B_6)$
$T_7(B_7)$
notifyall()
$C_k \neq C_{k-1}$
t_k
5
6
7
8
9
A
$T_5(B_5)$
$T_6(B_6)$
$T_7(B_7)$