

Ayuda para Práctica 4

Monitores

Descarga este documento en: <http://goo.gl/4gflo>

Actualizado 26/04/2013

Nota: estas notas reflejan una solución propuesta por el profesor del grupo de los lunes. Está sujeta a cambio en cualquier momento, explicándolo en dicho grupo de prácticas. Haz caso a tu profesor antes que a estas notas.

Más ayuda: en tutorías o
Pablo.Bermejo@uclm.es

1. Semántica de notify()/notifyAll()

- Recordad que cuando un hilo llama a *wait()* queda suspendido a la espera de recibir una señal de activación.
- Cuando hay varios hilos esperando...¿qué hilo se despierta primero al recibir 1 o varias señales?
 - Javadoc dice que en teoría el orden es arbitrario...
 - ...**VAMOS A COMPROBAR SI ESTO ES CIERTO EN LA PRÁCTICA**
- También hay que comprobar que el hilo despertador no libera el método sincronizado hasta que no termina su código: protocolo Resume and Continue.

1. Semántica de notify()/notifyAll()

- Creamos un objeto sobre el que se llamará a wait, notify y notifyAll:

```
public class Semantica()  
public synchronized void espera(int id)  
public synchronized void despiertaUno()  
public synchronized void despiertaTodos()
```

- Y 3 clases de hilos, cada uno encargado de llamar al método correspondiente de Semántica.

```
public class HiloEspera extends Thread()  
public HiloEspera(Semantica S, int ID)  
public void run()
```

```
public class HiloDespierta1a1 extends Thread()  
public HiloDespierta(Semantica S, int cuantos)  
public void run()
```

```
public class HiloDespiertaTodos extends Thread()  
public HiloDespiertaTodos(Semantica S)  
public void run()
```

1. Semántica de notify()/notifyAll()

- Y una clase Main, donde se invoca al main, y se dispone de 2 parámetros globales que controlan la ejecución

```
public class Main()  
static final int HILOS_A_DORMIR = //los que queráis  
static final boolean DESPIERTA_SOLO_A_UNO = //lo que queráis
```

- ¿En qué orden se despiertan los hilos al despertarlos uno a uno? ¿Y al hacerlo a la vez? ¿Arbitrario, FIFO, LIFO? ¿El javadoc tiene razón?

1. Semántica de notify()/notifyAll()

- Escribe un mensaje + sleep después de un notify()/notifyAll(), para comprobar que al despertar a un hilo, éste no hace nada hasta que no acaba el despertador.
Ej:
 - [HiloDespiertaTodos]: desencolo a todos los hilos PERO SIGO AQUÍ (Resumen and Continue)
 - [Hilo 20] salgo de la cola y termino el bloque sincronizado!

2. Puente SEGURO

- Es suficiente con preguntar antes de entrar al puente si está ocupado; y al salir hay que decir que ya no lo está → variable global ***boolean ocupado***.
- Es seguro, pero puede producir inanición o starvation.

2a. Puente JUSTO no fluido

- Se complica un poco. Necesitamos las siguientes variables globales para controlar el tráfico:
 - final int **MAX_EN_UN_SENTIDO**; //coches que pueden cruzar antes de dejar pasar a los del sentido contrario
 - boolean **ocupado**; //true cuando hay un coche en el puente
 - boolean **direccionDcha**; //true cuando el sentido permitido es de izda a dcha.
 - int **han_cruzado**; //cuenta los que han cruzado en el sentido permitido actualmente
- Cuando lo de arriba te funcione, ten en cuenta que, además, es posible que por un lado no haya coches por lo que hay que controlar cuándo no se debe cambiar el turno aunque toque:
 - int **dcha_esperando**; //coches esperando en el lado derecho
 - int **zqda_esperando**; //coches esperando en el lado izquierdo
- ¡Piensa cómo utilizarlas para que el puente sea seguro y justo!

2a. Puente JUSTO fluido

- final int **MAX_EN_UN_SENTIDO**; //coches que pueden cruzar antes de dejar pasar a los del sentido contrario
- boolean **direccionDcha**; //true cuando el sentido permitido es de izda a dcha.
- int **han_cruzado**; //cuenta los que han cruzado en el sentido permitido actualmente
- Para conseguir fluidez, cambiamos la booleana de *ocupado* por 2 enteros, que nos digan si el puente está ocupado y en qué dirección:
 - int **cruzando_a_dcha**; // coches cruzando hacia la derecha
 - int **cruzando_a_izqda**; //coches cruzando hacia la izqda
- Cuando lo de arriba te funcione, ten en cuenta que, además, es posible que por un lado no haya coches por lo que hay que controlar cuándo no dormir a los del otro lado aunque toque:
 - int **dcha_esperando**; //coches esperando en el lado derecho
 - int **izqda_esperando**; // coches esperando en el lado izquierdo

2b. Puente con Lock no fluido

- Instanciando un lock a `locks.ReentrantLock`, se consigue un puente seguro, y justo hasta conseguir una gestión de colas FIFO.

2b. Puente con Lock **fluido seguro**

- Ahora podemos tener 1 lock pero con 2 colas de hilos asociadas, en vez de una sola en los que se encolan los hilos llamando a wait.
- Necesitarás estas variables:
 - ReentrantLock **lock** = new ReentrantLock(true);
 - Condition **esperaIzquierda** = lock.newCondition(); // cola para los hilos de los coches rojos
 - Condition **esperaDerecha** = lock.newCondition(); // cola para los hilos de los coches azules
 - int **cruzando_a_dcha, cruzando_a_izqda**; //para conseguir fluidez
- El código dentro de cada método deberá ejecutarse en exclusión mutua. Como ya no tenemos synchronized, hay que capturar y liberar el lock.
- El fairness se conseguiría de forma similar al punto 2a