Universidad Rey Juan Carlos

Programación Concurrente
3º Grado en Ingeniería de Computadores

Examen Parte 2
11 de Enero de 2014

Apellidos:	**Nombre:**
DNI:	

El examen tendrá una duración de 2:30h

Ejercicio 1) Pregunta corta sobre concurrencia en lenguajes (2p)

¿Qué son OpenMP, Intel Cilk Plus, Win32, POSIX, Boost e Intel Threading Building Blocks? ¿En qué se parecen y en qué se diferencian? (2p)

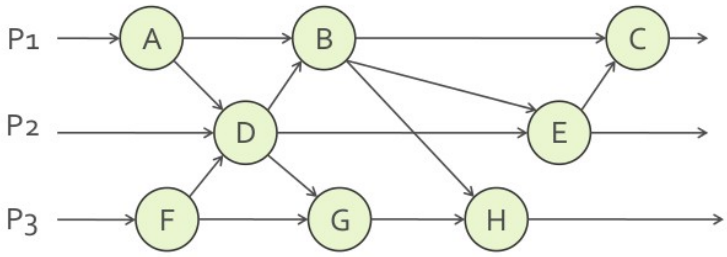
Ejercicio 2) Preguntas cortas sobre concurrencia en Java (2p)

A) Describe el funcionamiento de la herramienta de sincronización Intercambiador de Java (Exchanger). ¿Qué métodos tiene? ¿Para qué se usa? (1p)

B) ¿Qué son las activaciones inesperadas (*spurious wakeup*)? ¿En qué casos se pueden producir en Java? ¿Qué técnica se usa para evitar que esas activaciones afecten al correcto funcionamiento del programa? (1p)

Ejercicio 3) Programa (2p)

Implementa un programa Java completo que ejecute el siguiente diagrama de precedencia. Utiliza para ello la herramienta de sincronización CountDownLatch. Simula la ejecución de cada tarea como la impresión por pantalla de la letra indicada en el círculo. Implementa el código completamente (Creación de hilos, método main, etc.). No se puede utilizar SimpleConcurrent.



Solución

Ejercicio 4) Programa (4p)

Se dispone del programa *DataAccess,* un programa que tiene un único hilo de ejecución y accede a varias fuentes de datos. Para mejorar el rendimiento del programa en sistemas con varios núcleos de ejecución, se ha decidido convertir *DataAccess* en un programa concurrente.

El programa dispone de dos librerías de acceso a datos: *DataSource1* y *DataSource2*. Estas librerías no permiten su acceso concurrente, es decir, sólo un hilo de ejecución puede usar dichas librerías al mismo tiempo.

En el programa *DataAccess*, a veces se necesita acceder a *DataSource1* y otras veces a *DataSource2*. No obstante, en algunos casos, se puede utilizar indistintamente cualquiera de las dos fuentes de datos, *DataSource1* o *DataSource2*. Esto es especialmente importante cuando queremos convertir un programa secuencial (*single threaded*) en un programa concurrente. Si un hilo quiere acceder a un *DataSource* pero no le importa qué *DataSource* concreto es, y resulta que uno de los *DataSource* está siendo utilizado por otro hilo de ejecución, entonces podrá utilizar el *DataSource* libre.

Se pide:

Implementar únicamente la clase *DataSourceManager* para gestionar el acceso a los *DataSource,* de forma que sólo un hilo de ejecución pueda acceder a cada uno de ellos a la vez.

La clase debe tener la siguiente estructura:

```
public class DataSourceManager {

   //Atributos necesarios

   public DataSource(){
      //Inicializaciones necesarias
   }

   public void accessDataSource(int dataSource){
      //Este método será invocado cuando se quiera usar un DataSource concreto.
      //Se bloqueará hasta que ese dataSource esté libre. Una vez desbloqueado,
      //se podrá usar el dataSource indicado como parámetro. Cuando se termine de
      //usar, se deberá invocar el método "freeDataSource"
   }

   public int accessAnyDataSource(){
      //Este método será invocado cuando se quiera usar cualquier DataSource.
      //Se bloqueará hasta que cualquiera de los DataSource esté libre. El dataSource
      //libre se tendrá que devolver para que sea usado una vez desbloqueado el método.
      //Cuando se termine de usar, se deberá invocar el método "freeDataSource"
   }

   public void freeDataSource(int dataSource)
      //Este método se utilizará para liberar el DataSource que se esté usando para
      //que pueda ser usado por otros hilos.
   }

}
```

La clase DataSourceManager debe implementarse con los siguientes requisitos:

- Hay que tener en cuenta que los métodos de la clase *DataSourceManager* serán invocados por diferentes hilos de ejecución.
- Debe tener el comportamiento indicado en los comentarios del fragmento de código. Es decir, un hilo que intente acceder a un *DataSource* deberá invocar el método *accessDataSource(...)* o *accessAnyDataSource()* y quedará bloqueado hasta que lo pueda hacer. Cuando haya terminado de utilizar ese DataSource, deberá llamar a *freeDataSource(...)*.
- La asignación de los *DataSource* a los hilos debe realizarse en orden de llegada (FIFO).
- Cuando un *DataSource* queda libre, primero se intentará asignar a un proceso que sólo pueda acceder a ese *DataSource*. Si no hay ningún proceso de ese tipo esperando, se asignará a un proceso que pueda utilizar cualquier *DataSource*. En otro caso, el *DataSource* quedará libre.
- La solución deberá implementarse sin usar SimpleConcurrent.

Solución: