Practica 2 de Programación Concurrente y Distribuída

Miguel Tellería, Laura Barros, J.M. Drake

19,20 Oct 2011

Resumen

Esta práctica pretende familiarizar al alumno con la detección de necesidades de sincronización en código concurrente así como la solución de estas necesidades mediante técnicas de espera activa y de espera bloqueante usando primitivas del lenguaje Java.

1. Punto de partida

Se parte de la versión secuencial de la aplicación *RedFerroviariaSecuencial.zip* disponible en la web de la asignatura (sección ejemplos).

2. Tareas a realizar

1. Modificar el `ProgramaPrincipal` y la clase `Tren` para que los trenes funcionen como threads autónomos.
   Nota: En este punto no es necesario que se respeten las reglas de coherencia de la aplicación.
2. Identificar las necesidades de sincronismo rellenando para cada una la tabla siguiente:

Vbles a proteger	**Extensión de las regiones críticas**	**Ámbito (threads) concernido**

3. Resolver los sincronismos de `Tunel` y `CentroRegulacion` de la manera siguiente.

   - **Miércoles 19** Implementar un *sincronismo bloqueante* en el acceso al `Tunel` y un *sincronismo de espera activa* en el acceso al `CentroRegulacion`.
   - **Jueves 20** Implementar un *sincronismo de espera activa* en el acceso al `Tunel` y un *sincronismo bloqueante con timeout* en el `CentroRegulacion`.

   Por *sincronismo de espera activa* entendemos que se preguntará de manera periódica la disponibilidad de un recurso, mientras que por *sincronismo bloqueante* entendemos que la disponibilidad de un recurso se mirará *sólo una vez* antes de suspenderse y esperar a ser rearrancado cuando haya surgido un cambio en el predicado lógico.

Una vez resuelto los casos de sincronismo se tiene que seguir manteniendo la funcionalidad original del código secuencial, es decir:

- En el `Tunel` el primer tren que llega define el color admitido de paso y sólo cuando vuelve a quedar libre se permite cambiar de color autorizado.
- En el `CentroRegulacion` el criterio de salida es de naturaleza temporal y consiste en que se respete el `tiempo_minimo_entre_salidas`.
- Cuando el programa principal invoca `termina()` en los trenes, éstos vuelven a su `CentroRegulacion` antes de terminar su ejecución.