Practica 3 de Programación Concurrente y Distribuída

Miguel Tellería, Laura Barros, J.M. Drake

26,27 Oct 2011

Resumen

Esta práctica pretende familiarizar al alumno con el concepto de ThreadPool y su implementación en Java

1. Punto de partida

Se parte de la versión concurrente y con espera bloqueante de la Red Ferroviaria (`RedFerroviariaEspera Bloqueante` disponible en la web de la asignatura (sección ejemplos). Notad que este código presenta la clase `SwingSelector` que se especificará más adelante.

2. Objetivo y tareas a realizar

La idea es implementar la red ferroviaria haciendo uso de un `SchedulerThreadPoolExecutor` por cada `CentroRegulacion` manejado y encapsulado por éste.

- **Miércoles 26**: Se utilizarán **tantos threads como trenes hay por color** y la tarea a planficicar corresponderá al *ciclo completo de una vuelta* de un tren.
- **Jueves 27**: Se utilizará **un thread UNICO por cada color** que controlará todos los trenes de su color. La tarea a planificar consistirá a la llamada al método `actualiza()`.

En ambos casos los trenes *se crean y destruyen bajo demanda* mediante el formulario de la clase `SwingSelector` que tiene los siguientes valores:

r+ Creación de un nuevo tren rojo en el `CentroRegulación` correspondiente.

r- Terminación del tren rojo *que más próximo se encuentre al `CentroRegulacion` cuando se haya dado este comando*.

b+ Creación de un nuevo tren azul en el `CentroRegulación` correspondiente.

b- Terminación del tren azul *que más próximo se encuentre al `Centro Regulacion` cuando se haya dado este comando*.

e Se para el ThreadPool y se espera que terminen las tareas pendientes. Por simplicidad esto quiere decir:

- En el caso de los miércoles los trenes "pendientes" volverán al `CentroRegulación` antes de terminarse.
- En el caso de los jueves el sistema se quedará parado sin actualizarse más.

El procedimiento sugerido para realizar esta práctica es el siguiente.

1. En la clase `Tren`:
   a) Hacerla clase pasiva (no `Thread`) implementando la interfaz `Runnable`.
   b) Ajustar el método `run()` a la duración de la tarea planificable, circuito completo (miércoles) o actualiza (jueves).
   c) Verificar si las zonas de sincronismo siguen siendo necesarias.
   d) Ajustar la terminación del tren al executor del centroRegulación.
2. En la clase `CentroRegulacion`
   a) Colocar el `ScheduledThreadPoolExecutor` que gobernará los trenes de su color con el número de threads dados en el enunciado (5 para el miércoles, 1 para el jueves).
   b) Modificar el método `entro` dando como parámetro extra el tren para planificar éste en el `executor`. Elegir entre planficación *oneShot*, planificación con *delay constante* o planificación *con periodo constante* segun se vea oportuno.
   c) Si el `executor` se definió como atributo privado, proveer métodos `termina_tren` para cancelar la ejecución de un tren y `termina_executor` para cerrar el executor.
3. En el `ProgramaPrincipal` hay que cambiar la *lógica secuencial* de crear trenes, esperar, avisar y sincronizarse por una *lógica de comandos* mediante un `switch` de eventos retornados por el `SwingSelector`.