EXAMEN 1 PROGRAMACIÓN CONCURRENTE: MÁQUINA DE GALTON

Realizado por Pedro Alonso Tapia Lobo y Patrik Paul Sirbu

El objetivo de este examen evaluar la capacidad de coordinar múltiples hilos en ejecución en un entorno de programación concurrente. Se quiere conseguir simular una distribución normal de Gauss en tiempo real. Programación paralela para optimizar el proceso de producción y visualización en tiempo real
Universidad Alfonso X el Sabio

Índice

1. Enunciado
2. Explicación de archivos
3. Metodologías seguidas

1. Enunciado

Título: La Operación de Galton en la Fábrica de Campanas de Gauss

En el año 1850, en un universo paralelo donde la tecnología avanzó rápidamente, Sir Francis Galton creó un dispositivo que llamó la "máquina de la distribución normal". Esta máquina, basada en su tablero de Galton, demostró que al caer bolas a través de una serie de clavos en ángulo, la distribución final de las bolas en los contenedores inferiores formaba una curva de campana, también conocida como una distribución normal o Gaussiana.

Galton fundó una fábrica, "Campanas de Gauss", para producir estas máquinas en masa y demostrar este fenómeno matemático al mundo. En la fábrica, varias estaciones de trabajo simultáneas producen diferentes componentes de la máquina, y el ensamblaje final se lleva a cabo en una línea de producción separada.

Tú, un brillante ingeniero de software de la época, has sido contratado para simular el proceso de producción utilizando hilos de ejecución para modelar la concurrencia de la fabricación. Debes cumplir con los siguientes requisitos:

1. **Cada estación de trabajo de la fábrica es un hilo de ejecución independiente**. Estos hilos deben coordinarse entre sí para asegurar que los componentes de la máquina se produzcan en el orden correcto y en las cantidades correctas para el ensamblaje final.

2. **Implementa un mecanismo de sincronización** para garantizar que los componentes de la máquina se produzcan y ensamblen de manera ordenada. Esto puede implicar el uso de locks, semáforos, o variables de condición.

3. **Implementa la producción y el ensamblaje de la máquina como un problema de productor-consumidor**. Las estaciones de trabajo que producen componentes son los productores, y la línea de ensamblaje es el consumidor.

4. **Usa un modelo de memoria compartida** para permitir la comunicación entre los hilos. Los componentes producidos por las estaciones de trabajo deben colocarse en un búfer compartido del que la línea de ensamblaje pueda retirarlos.

5. **Implementa la distribución de las tareas de producción entre las estaciones de trabajo utilizando un algoritmo de scheduling**. Puede ser round-robin, prioridad, más corto primero, etc.

6. **Usa técnicas de programación paralela y distribuida** para acelerar el proceso de producción. Esto puede implicar la distribución de la carga de trabajo entre varios procesadores o nodos de un sistema distribuido.

Tu misión final es mostrar cómo a medida que el número de bolas (simuladas con hilos) cae a través del tablero, la distribución final se acerca cada vez más a una distribución normal. Este fenómeno debe ser mostrado visualmente en tiempo real a medida que las bolas llegan a los contenedores inferiores.

El objetivo es utilizar el poder de la programación paralela y distribuida para ilustrar este fenómeno en un entorno de producción concurrente, y en el proceso, aprender más acerca de cómo implementar y coordinar hilos de ejecución en un entorno de programación real.

2. Explicación de archivos

Voy a explicarte brevemente el propósito de cada clase dentro de tu proyecto basado en el contenido de los archivos Java de la carpeta src.

1. GaussfactoryExamApplication.java

Esta clase es la clase principal de la aplicación. En ella se inicializa y arranca la aplicación Spring Boot. También actúa como el punto de entrada del sistema, donde se configuran los recursos y se inicia el proceso de simulación.

2. DataLoader.java

Esta clase se encarga de cargar los datos necesarios para la simulación. Lee datos desde un archivo (CSV, por ejemplo) y los convierte en una lista de elementos que serán utilizados en otros componentes de la aplicación.

3. SimulationController.java

El SimulationController actúa como un controlador de la lógica de negocio relacionada con la simulación. Define los endpoints que permiten a otros sistemas o usuarios interactuar con la simulación, posiblemente a través de una API REST. Es el responsable de iniciar y gestionar el proceso de simulación a través de la interfaz gráfica o de peticiones HTTP.

4. GaussChart.java

Esta clase es responsable de la visualización de los datos simulados. Dibuja o representa gráficamente la distribución de las bolas (o los elementos) a medida que la simulación avanza. Esta representación podría ser una curva de campana (distribución normal), que es el objetivo visual de la simulación.

Estas son las clases principales del proyecto y sus responsabilidades a grandes rasgos. Cada una de ellas desempeña un papel importante en la simulación de la fábrica de la distribución normal que has creado. Si necesitas más detalles o tienes preguntas adicionales sobre alguna clase en específico, ¡háblame!

4. Metodologías usadas

Se ha usado test unitarios para hacer las pruebas de las clases

Se ha seguido un patrón factory method.

Se ha utilizado comentarios javadoc, para su posterior creación

Se ha importado datos .csv

Proyecto creado con Spring Boot, con sus correspondientes dependencias.