Felipe Ham - Santiago Lozano

Entregable 2 - Programación Avanzada - Concurrencia

Descripción del sistema:
El sistema implementado maneja el procesamiento concurrente de pedidos en un entorno de comercio electrónico, dividiendo el flujo de procesamiento en tres etapas: Procesamiento de Pago, Empaquetado de Pedidos y Envío de Pedidos. El objetivo es garantizar que varios pedidos se procesen de forma simultánea, manteniendo la prioridad de los pedidos urgentes, y asegurando la finalización correcta y secuencial de las tres etapas antes de completar un pedido.

Decisiones de diseño:

- Procesador de pedidos y concurrencia:
  El ProcesadorPedidos utiliza la clase ExecutorService con un pool de hilos fijo (FixedThreadPool) para gestionar el procesamiento concurrente de los pedidos. Cada pedido pasa por tres etapas secuenciales y el sistema se encarga de ejecutar cada una en su orden.

  FixedThreadPool fue elegido para mantener un número controlado de hilos concurrentes, lo que permite limitar el uso de recursos del sistema y evitar sobrecargas innecesarias. Esto es útil porque cada etapa del procesamiento tiene tiempos distintos y podría sobrecargar el sistema si no se controla.

- Cantidad de hilos:
  En la implementación final, se ha optado por un pool de 7 hilos (newFixedThreadPool(7)), lo que asegura que hasta 7 pedidos pueden estar siendo procesados simultáneamente. Esta cantidad se determinó como el equilibrio óptimo para manejar múltiples pedidos simultáneamente sin sobrecargar el sistema a partir del análisis de rendimiento (mostrado más abajo).

  Procesamiento de pago: Esta etapa, aunque relativamente rápida, involucra validaciones de pago y conexiones a servicios externos como bancos o brokers de pago, por lo que podría ser una operación de bloqueo. Se asigna un hilo del pool por pedido durante esta etapa para manejar estas operaciones de forma asincrónica, asegurando que el bloqueo de I/O no retarde el procesamiento de otros pedidos en el sistema.

  Empaquetado: Se utiliza un ForkJoinPool para optimizar la paralelización de esta etapa, ya que el empaquetado puede procesarse en paralelo a otros pedidos. Esto permite que, si hay muchos pedidos, varios puedan estar siendo empaquetados al mismo tiempo.

  Envío: El envío es crítico y requiere que todas las etapas anteriores estén completas para cada pedido. Aunque esta etapa también implica espera de I/O, cada pedido requiere un seguimiento individual hasta su finalización. Un hilo se asigna para gestionar el envío tan pronto como un pedido esté listo, asegurando que no haya retrasos en la salida del pedido del sistema.

- Prioridad:
  Para la priorización de pedidos urgentes, se ha utilizado una PriorityBlockingQueue que ordena los pedidos según su prioridad. Los pedidos urgentes se procesan primero, ya que la cola asigna mayor prioridad a aquellos marcados como urgentes (compareTo() compara el atributo booleano urgente).

  Al usar una cola bloqueante, nos aseguramos de que los pedidos sean procesados de manera concurrente sin riesgos de acceso no seguro a recursos compartidos.
  Este mecanismo garantiza que los pedidos urgentes reciban un tratamiento preferencial y que se procesen antes que los pedidos normales, sin necesidad de complejidades adicionales en el manejo de prioridades dentro de ExecutorService.

- Sincronización y orden:
  Cada pedido sigue un flujo claramente definido: primero se procesa el pago, luego se empaqueta, y finalmente se envía. Esto se asegura con la secuencia estricta dentro del método procesarPedido(). Además, el uso de ForkJoinPool en la etapa de empaquetado permite manejar varios empaquetados simultáneamente si es necesario.

  El sistema asegura que, aunque varias tareas pueden ejecutarse en paralelo, la secuencia de procesamiento dentro de un pedido individual es secuencial y está bien sincronizada, evitando conflictos entre etapas.

- Cierre ordenado
  Para garantizar un cierre correcto del sistema, se utiliza el método shutdown() de ExecutorService junto con awaitTermination() para esperar a que todas las tareas se completen antes de terminar. En caso de interrupciones, se asegura una finalización limpia con shutdownNow(), que fuerza el cierre de todos los hilos activos.

Análisis de rendimiento

- Eficiencia del procesamiento
  Con la configuración actual (7 hilos y ForkJoinPool para empaquetado), el sistema es capaz de manejar 100 pedidos concurrentes de forma eficiente. Las pruebas han demostrado que:

  - Pedidos urgentes se procesan primero, reduciendo significativamente los tiempos de respuesta para los pedidos más críticos.
  - La secuencialidad de las etapas dentro de cada pedido asegura que no se complete ningún pedido sin haber pasado por todas las etapas.
  - El uso de ForkJoinPool para paralelizar el empaquetado de múltiples pedidos no mejora el rendimiento o la velocidad del procesamiento, porque en este caso la tarea de empaquetado está simplificada con un sleep y no se divide en subtareas (para lo que sí sería efectivo el ForkJoinPull). Decidimos igualmente utilizarlo porque si lo querríamos llevar a la realidad, allí sí estarían especificadas subtareas y este método se encargaría de optimizar el manejo y división de tareas concurrentes.

- Pruebas de rendimiento (tiempos aproximados)

En el código Main fuimos variando la cantidad de pedidos e hilos, y obtuvimos los siguientes datos:

- 5 pedidos:
  - 1 hilo: 3,4 segundos
  - 2 hilos: 2 segundos
  - 3 hilos: 1,4 segundos
  - 4 hilos: 1,4 segundos
  - 5 hilos: 1 segundo
  - 10 hilos: 1 segundo
- 15 pedidos
  - 1 hilo: el procesamiento con una configuración de un solo hilo llevaba en el entorno de los 10 segundos
  - 3 hilos: 3,4 segundos
  - 5 hilos: 2,3 segundos
  - 6 hilos: 2,1 segundos
  - 7 hilos: 2,1 segundos
  - 8 hilos: 2 segundos
  - 9 hilos: 1,9 segundos
  - 10 hilos: 1,9 segundos
  - 11 hilos: 1,9 segundos
  - 15 hilos: 1,9 segundos
- 100 pedidos
  - 5 hilos: 13,9 segundos
  - 6 hilos: 11,8 segundos
  - 7 hilos: 10,8 segundos
  - 10 hilos: 10,8 segundos
  - 13 hilos: 10,8 segundos

Las decisiones que se tomen respecto a la cantidad óptima de hilos van a considerar un equilibrio entre el tiempo de procesamiento y el costo o cantidad de hilos, y además la cantidad de pedidos que va a procesar el sistema normalmente. Viendo los tiempos de procesamiento con las diferentes cantidades de pedidos, creemos que 7 hilos puede ser un buen número si se tiene en cuenta que el sistema debe manejar hasta 100 pedidos concurrentes de manera eficiente, y si son menos, no habrá problema con tener 7 hilos.