Patrones Concurrentes Taller 4 – Corte II

David Santiago Téllez Melo

En el taller implemente las dos versiones de un sistema concurrente basado en el patrón productor-consumidor. La primera versión emplea hilos tradicionales (`threading`) y la segunda utiliza un `ThreadPoolExecutor` para gestionar la concurrencia de manera automática. En este documento analizo el flujo de trabajo, el rendimiento y la eficiencia de ambas versiones, y reflexiono sobre la modularidad y escalabilidad del diseño.

Resultados de Ejecución

Versión con hilos tradicionales

```
============================================================
RESULTADOS (hilos):
Total producidos: 40
Total consumidos: 40
Tiempo total: 5.13 segundos
Throughput (consumidos / s): 7.79
============================================================
```

Versión con ThreadPoolExecutor

```
================================================
RESULTADOS (ThreadPoolExecutor):
Total producidos: 40
Total consumidos: 40
Tiempo total de ejecución: 4.82 segundos
Throughput (consumidos / s): 8.29
================================================
```

Figure 1:

Análisis del flujo de trabajo concurrente

En ambas versiones hay un patrón típico productor-consumidor: los productores generan ítems que se colocan en una cola con tamaño máximo limitado, y los consumidores procesan estos ítems según su disponibilidad.

- **En los hilos tradicionales:** cada hilo se gestiona manualmente, alternando producción y consumo. La cola limita la cantidad de ítems pendientes, evitando saturación.
- **En ThreadPoolExecutor:** la asignación de tareas a los hilos se realiza automáticamente, simplificando el control de concurrencia y reduciendo overhead.

Rendimiento comparativo

La versión con `ThreadPoolExecutor` mostró un tiempo total ligeramente menor y un throughput superior, indicando un desempeño más eficiente al hacer tareas concurrentes. La diferencia es pequeña por tamaño limitado de la prueba, pero es representativa de cómo un pool de hilos puede optimizar la ejecución en sistemas más grandes.

Eficiencia en el uso de recursos

- **Hilos tradicionales:** consumen recursos adicionales por cada hilo creado y requieren manejo explícito de sincronización.
- **ThreadPoolExecutor:** reutiliza hilos existentes y gestiona automáticamente la asignación de tareas, reduciendo overhead y consumo de recursos.

Ambas versiones utilizan una cola con tamaño máximo (`maxsize`) para controlar la saturación, evitando picos innecesarios de consumo de CPU o memoria.

Diferencias entre versiones y los patrones que aplique

- La versión con hilos tradicionales da un control más explícito del flujo de trabajo, pero incrementa la complejidad en la gestión de sincronización.
- La versión con `ThreadPoolExecutor` abstrae la gestión de hilos, facilitando escalabilidad y mantenimiento, manteniendo el patrón productor-consumidor.
- Ambas versiones aplican el patrón de concurrencia productor-consumidor con cola limitada para sincronización entre productores y consumidores.

modularidad y escalabilidad

Aunque como se comenta este taller se implementa como un solo script, tambien se puede valorar una estructura más modular inspirada en el patrón MVC:

- **Modelo:** producción y almacenamiento de datos (productores y cola).
- **Vista:** registro y monitoreo de eventos (logs de producción y consumo).
- **Controlador:** gestión del flujo de ejecución y asignación de hilos.

Esta separación permitiría escalar el sistema para proyectos más grandes, facilitando la extensión de funcionalidades, la optimización de recursos y el mantenimiento del código.