Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Puebla

TC2037

Implementación de métodos computacionales

Grupo 601

Actividad Integradora 5.3 Resaltador de sintaxis paralelo (evidencia de competencia)

Profesor

Luciano García Bañuelos

Integrantes

Carlos Daniel Díaz Arrazate	A01734902
José Ángel González Carrera	A01552274

10 de junio de 2022

Repositorio: (Tag release evidencia 2)

https://bitbucket.org/di-az/syntaxhighlighter/src

1. **Extiende el programa realizado en la actividad 3.4 para que aplique, de manera secuencial, el resaltado de léxico a múltiples archivos fuente contenidos en uno o varios directorios anidados.**
2. **Desarrolla una nueva versión de tu programa para que realice el proceso de resaltado de léxico de manera paralela con el fin de aprovechar los múltiples núcleos disponibles en los CPUs modernos.**
3. **Asegúrate de utilizar en tu código fuente las convenciones de codificación del lenguaje en el que está implementado tu programa.**
4. **Mide los tiempos de varias ejecuciones de las dos versiones de tu programa. Para hacer una comparación "sistemática", te propongo que pruebes con 50 archivos fuente de entrada, luego con 100, 150 y finalmente con 200 archivos. Calcula el speedup obtenido y genera un gráfico con los tiempos obtenidos para ambas versiones secuencial y concurrente. Reflexiona sobre las soluciones planteadas, los algoritmos implementados y sobre el tiempo de ejecución de estos.**

Para implementar la solución secuencial bastó con utilizar un map que recorra el número de archivos dado, y que por cada archivo llame la función main(), la cual es responsable de ejecutar el syntaxhighlighter a cada archivo. Cabe recalcar que, los archivos son llamados uno por uno, lo que quiere decir que para que se vuelva a llamar a la función main() en otro archivo, la ejecución de la función debe terminarse en el archivo anterior. El tiempo de ejecución de la versión secuencial depende del número de archivos, por lo que se puede determinar que su complejidad es lineal ***O(n)***.

De igual manera, la solución concurrente requirió del uso de un map que recorra la cantidad de archivos dados, llamando por cada archivo a la función main(), con la diferencia de que, el llamado a la función main() es asincrónico puesto que se empleó Task.async() con el objetivo de generar concurrencia. La adición de

Task.async() permite que los distintos archivos llamen a la función main() simultáneamente, con el objetivo de eficientizar la velocidad de ejecución. El tiempo de ejecución de esta versión concurrente también depende del número de archivos, teniendo una complejidad lineal ***O(n)***.

A continuación se muestran los resultados obtenidos tras la ejecución de las versiones secuencial y concurrente con 50, 100, 150, 200, 300 y 500 archivos.

Cantidad de archivos	Secuencial (ms)	Concurrente (ms)
50	162.87	54.24
100	332.06	98.03
150	486.82	152.81
200	624.94	178.03
300	975.18	273.69
500	1590	470

Figura 1. Tabla de tiempos de las versiones secuencial y concurrente.

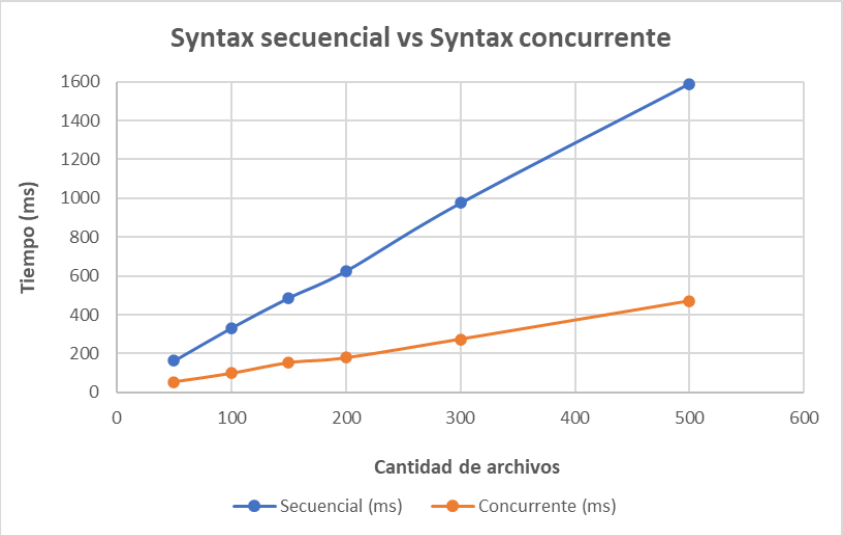

Figura 2. Gráfica de tiempos de las versiones secuencial y concurrente.

5. **Calcula la complejidad de tu algoritmo basada en el número de iteraciones y contrástala con el tiempo obtenido en el punto 4.**

En cuanto a la complejidad de tiempo del algoritmo, como se puede observar en la gráfica que compara el tiempo de ejecución con respecto a la cantidad de archivos, se puede apreciar un comportamiento lineal para ambas soluciones, lo cual permite inducir que se trata de una complejidad de **O(n)** para **ambos casos**, esto debido a que conforme se agregan más archivos, el tiempo que le toma a ambas soluciones aumenta en función de cuánto tiempo le tomará el procesar los nuevos archivos.

Por otro lado, con respecto a la complejidad del espacio, la situación es distinta. Para la solución **secuencial**, se está hablando de que se procesa a cada archivo de manera individual, por lo tanto, asumiendo que se hace correcto uso de memoria por parte del lenguaje y/o el compilador. Primeramente se reserva memoria para poder procesar el archivo, una vez concluido el análisis del archivo se libera la memoria de este y se procede a reservar memoria para el siguiente archivo a procesar; por ende, se puede entender que, a pesar de que la cantidad de archivos incrementa, la cantidad de memoria necesaria se mantiene constante (Dependiendo del tamaño del archivo), por lo tanto, se está hablando de una complejidad de **O(1)**.

En cuanto a la complejidad de espacio de la solución **concurrente**, se está hablando de una complejidad de **O(n)**, puesto que, para poder procesar los archivos de manera paralela, es necesario contar con estos archivos en memoria, lo cual implica que se está reservando una cantidad de memoria equivalente a la cantidad de archivos a procesar; entre mayor cantidad de archivos, mayor cantidad de memoria.

6. **Plasma en un breve reporte de una página las conclusiones de tu reflexión en los puntos 5 y 6. Agrega además una breve reflexión sobre las implicaciones éticas que el tipo de tecnología que desarrollaste pudiera tener en la sociedad.**

Carlos Daniel Díaz Arrazate

Sin duda alguna, la concurrencia es una tecnología de extrema importancia en la actualidad, se puede observar en infinidad de sistemas y aplicaciones. Para el caso del procesamiento de varios archivos, sin duda alguna la concurrencia es el camino a seguir, como se pudo observar en los resultados presentes en la gráfica.

Para esta clase de proyectos, la implementación de concurrencia es bastante sencilla, puesto que no hubo que tomar en cuenta problemas derivados de la implementación de la misma. Sin embargo, para proyectos más complejos, puede que la implementación de concurrencia pueda resultar en mayores problemas.

En cuanto a las implicaciones éticas de la tecnología, el aspecto principal es el manejo de la información, puesto que se está manipulando archivos de terceros. Por ende, es importante mantener confidencialidad y hacer un uso adecuado en donde se proporciona al usuario que es lo que se realiza con su información.

Por otra parte, en cuanto a la concurrencia, hay que tener en cuenta que la concurrencia implica el uso de una mayor cantidad de recursos, por lo que también hay que procurar que se haga un correcto manejo de los recursos, tanto del equipo, como del usuario. Esto puede causar problemas para el usuario, debido a que el programa está abarcando recursos a los que otros procesos no tienen acceso debido a que el recurso está siendo ocupado, causando pérdida de tiempo e inconvenientes.

José Ángel González Carrera

La importancia que tiene la concurrencia en la programación actual resulta de gran magnitud por las distintas ventajas que trae consigo su implementación. La concurrencia permite que distintos procesos sean ejecutados simultáneamente, lo

que incrementa la eficiencia de los programas; además de que, promueve una mejor utilización de los recursos de una manera organizada.

En la sociedad, el tema de la concurrencia se puede hacer presente para facilitar diversas tareas y procesos, los cuales pueden ir desde sistemas de bancos, sistemas de reservación de viaje, videojuegos multijugador, maquinaria en fábricas, entre otros campos.

Así que, dada la relevancia de la concurrencia en distintas áreas de la sociedad, se debe tener en observación los posibles problemas que puede traer su implementación. Uno de los principales puntos a considerar con la concurrencia es cuando múltiples procesos usan variables comunes o cambian el valor de ellos, por lo que la adición de un estado al programa se hace fundamental.