Instituto Tecnológico de Estudios Superiores de Monterrey

Actividad integradora 5.3

Actividad integradora 5.3 resaltador de sintaxis paralelo

Unidad Formativa:

Implementación de métodos computacionales
(TC2037.600)

Alumnos (Equipo #3):

Jorge Daniel Cruz Case - A01634536

Campus:

Guadalajara

Actividad integradora 5.3 resaltador de sintaxis paralelo

Reflexión de la solución planteada

Para la construcción del resaltador léxico paralelo y secuencial se utilizaron varias herramientas de programación implementadas para la generación de "*Lexers*" de este tipo, permitiendo la construcción de una clase que interprete las expresiones regulares y los alfabetos declarados. Las herramientas utilizadas fue el lenguaje de programación *Java* y el equivalente de *Flex* de Java, llamada *Jflex*. Esto se realizó debido a que la implementación paralela del resaltador léxico, en mi opinión, iba a ser más sencilla en Java que en C, además que el paralelismo en C utilizando las *Poxi Threads* es incompatible con Windows.

Debido a que *Jflex* puede interpretar los mismos tipos de archivos que *Flex* no fue un problema pasar aquello declarado en el archivo anterior a uno compatible con *Jflex*. La dificultad vino en la instalación de *Jflex* y en la implementación de la función que utilizara el archivo generado por la herramienta.

Al ver los tutoriales y la documentación de la herramienta *Jflex* me percate que este tipo de archivos ".lex" o ".flex" funcionan mejor cuando tienen anidados un archivo de *tokens* los cuales pueden ser utilizados en el lexer y ser regresados hacia el programa principal para su interpretación. El lexer regresa un token dependiendo de lo leído, lo cual es recibido por el programa principal e interpretado mediante un switch.

La actividad especificaba que el nuevo programa debía tener la capacidad de resaltar varios archivos. Para esto decidí que a la función de la genera el lexer y resalta los archivos recibiera el archivo que va a leer y el archivo en el que escribirá el resultado resaltado. Estos cambios permitieron que se pudieran resaltar varios archivos individuales en un solo programa.

La instrucción "resaltado de léxico a múltiples archivos fuente contenidos en uno o varios directorios anidados" me causo ruido, ya que no sabia como hacer que el programa pueda reconocer los archivos de entrada en cualquier directorio, por lo cual modifique la función generadora del lexer para permitir la entrada del "path" de dónde se encuentra el archivo. Tal vez esa no era lo que se pedía, pero las instrucciones de la actividad son lo suficientemente vagas como para crear esa confusión.

Por otro lado, no sabía si el resaltado de los archivos de entrada tenía que quedar en un solo archivo de salida, por lo cual, para simplificar la implementación del programa paralelo decidí que cada archivo de entrada creara un archivo resultante.

La simplificación del programa paralelo viene en que cada hilo trabaja de forma independiente de si, y no a la misma velocidad, por lo cual al crearse todo en un solo archivo resultaría en un resaltado de código desordenado. La solución que se me ocurrió pero no implemente, fue pasarle a la función del lexer un 3° dato que represente en que línea debe de empezar a escribir. Pero para esto se debe de poder saber el tamaño de cada archivo que entra y hacer ajustes dependiendo de esto, lo cual, en mi opinión, complica más de lo necesario el resaltado léxico. Tomando esto en cuenta, así como el hecho de que las instrucciones no lo especifican, decidí que cada archivo entrante resulté en un archivo individual de salida. Tal vez no se utilizan la mayor cantidad de núcleos posibles, pero simplifica el programa significantemente.

Por último, los tutoriales sugerían que todo el código que se haga se empaquete todo junto, por lo cual empaquete ambos mains, el lexer, el archivo ".flex", el archivo con los tokens y un archivo ".jar" que sirve como optativa en lugar de instalar *Jflex.*

Cálculo del *speedup*

Para el cálculo del *speedup* se utilizará un promedio de 5 ejecuciones de ambos, el programa secuencial y el programa paralelo, como el tiempo de ejecución en milisegundos de cada versión del programa.

- Programa secuencial (3 archivos): *84 ms en promedio*
- Programa paralelo (3 archivos): *60 ms en promedio*
- Para el cálculo del *speedup* realizaremos el siguiente cálculo:

$$S_P = \frac{T_I}{T_P}$$

$$P = número\ de\ núcleos,$$

$$T_I = tiempo\ de\ programa\ secuencial,$$

$$T_P = tiempo\ de\ prograam\ multihilo\ con\ p\ procesadores,$$

$$S_p = speedup\ usando\ p\ procesadores$$

El *speedup* del programa con 3 archivos es de:

$$S_P = \frac{(84)}{(60)} = 1.4$$

El programa no tiene un speedup tan distinto, lo cual se puede deber a muchas cosas, puede ser debido al multitasking de los núcleos o puede ser que mi implementación no haya sido correcta. De cualquier modo, el *speedup* de los programa es del 14%.

Cálculo de las complejidades

Para ambos programas hay 2 datos que dictan la complejidad de los algoritmos implementados, estos son:

- *n* la cantidad de tokens reconocidos en un archivo
- *m* la cantidad de archivos ingresados para resaltar

En ambos casos, programa secuencial y paralelo, se reciben *m* archivos para resaltar, los cuales identifican *n* tokens, los cuales podrían ser vistos como el tamaño o la *length* del archivo que se está resaltando. En su totalidad, el programa tendría una complejidad de ***O(n x m)***. Pero la función que genera el archivo resaltado sería ***O(n)***. La única diferencia es que el paralelo utiliza 1 hilo por ejecución de la función, lo cual llevaría a la complejidad de cada hilo en ***O(n)*** mientras que el secuencial sería ***O(n x m)***.

Esto puede explicar el porque los tiempos de ejecución son tan similares entre el paralelo y el secuencial, debido a que el valor de *m* es lo suficientemente bajo como para no afectar realmente al tiempo de la ejecución de ambos programas.

Implicaciones éticas del tipo de programa desarrollado en la sociedad

Es un poco extraño pensar en las implicaciones éticas que puede llegar a tener programa como los desarrollados en esta actividad. Cualquier programa puede ser utilizado con malicia o con malas intenciones, especialmente viendo como los programas pueden identificar palabras y familias léxicas.

Este tipo de programas pueden ser utilizados para recopilar información importante, identificándola y guardándola. Un ejemplo sería identificar un número de teléfono, una contraseña o un nombre, robándolo y utilizándolo para estafar o para espiar a alguna persona. Puede ser implementado en algún tipo de malware, que detecte el nombre de archivos importantes y se infiltré en el sistema a través de ahí.

Al final del día, cualquier programa podría ser utilizado de manera maliciosa, solamente depende de aquellas personas que lo hayan creado, el límite del potencial del programa es el límite de la imaginación de aquel que lo creo.