IES AUGUSTO GONZÁLEZ DE LINARES

CFGS DAM2º

Suma Paralela de Ficheros

PROGRAMACIÓN DE SERVICIOS Y PROCESOS

Ramiro Gutiérrez Valverde

CURSO 2023/24

Índice

1.- Demostración de la funcionalidad de la aplicación.

Tal como se solicita en el enunciado de la práctica, la primera tarea es la de crear un programa (GeneraFichero), que genere un fichero de n números aleatorios. Lo he hecho mediante un método, generarFicheros() que recibe como parámetros la cantidad de ficheros que se quieren generar y cuántos números va a llevar cada fichero.

```java
/**
 *
 * @author Ramiro gutiérrez Valverde
 */
public class GeneraFichero {

    public static void main(String[] args) {...5 lines }

    /** Método que genera ficheros de números enteros, entre 0 y 100, con formato ...9 lines */
    public static void generarFicheros(int numeroFicheros, int cantidadNumeros) {

        for (int i = 0; i < numeroFicheros; i++) {
            File archivo = new File("contabilidad" + (i + 1) + ".txt");

            if(archivo.exists()){
                archivo.delete();
            }

            try ( FileWriter fw = new FileWriter( file:archivo)) {
                for (int j = 0; j < cantidadNumeros; j++) {
                    fw.write((int) (Math.random() * 100) + "\n");
                }
                System.out.println("contabilidad" + (i + 1) + ".txt escrito con éxito");

            } catch (IOException ex) {
                System.err.println( x:"IO Exception");
            }

        }
    }

}
```

Ilustración 1. Método generarFicheros

Desde el método main(), pasando los parámetros correspondientes, ejecuto el programa obteniendo el siguiente resultado:

```java
/**
 *
 * @author Ramiro gutiérrez Valverde
 */
public class GeneraFichero {

    public static void main(String[] args) {

        generarFicheros( numeroFicheros:3,  cantidadNumeros:1000000);

    }
```

```
utput - Run (GeneraFichero) ×
cd /Users/Ramiro/Desktop/DAMT2/PSP/EVALUACION_1/UD1_Programacion multiproc
Running NetBeans Compile On Save execution. Phase execution is skipped and
Scanning for projects...

--------------------< com.mycompany:GeneraFichero >---------------------
Building GeneraFichero 1.0-SNAPSHOT
--------------------------------[ jar ]---------------------------------

fault-cli) @ GeneraFichero ---
contabilidad1.txt escrito con éxito
contabilidad2.txt escrito con éxito
contabilidad3.txt escrito con éxito
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  2.746 s
Finished at: 2023-10-15T12:29:35+02:00
------------------------------------------------------------------------
```

Ilustración 2. Ejecución exitosa del programa para generar los ficheros.

El segundo paso es crear un programa (Suma) que, recibiendo como argumento del String[] args, el nombre de un archivo haga el sumatorio de los números que contiene. Además, debe escribir un archivo con la respuesta utilizando el nombre del anterior y añadiendo la extensión ".res".

```java
/*
 * @author Ramiro Gutiérrez Valverde
 *
 */
public class Suma {

    public static void main(String[] args) {

        int suma = 0;
        String lectura = null;

        // declaro los ficheros que necesito
        File fichero = new File(args[0]);// para leer (FileReader)
        File ficheroRes = new File(args[0] + ".res");// para escribir (Filewriter)

        if (ficheroRes.exists()) {
            ficheroRes.delete();
        }

        // utilizo el try with resources para iniciar los flujos
        try ( FileReader fr = new FileReader( file:fichero);
              BufferedReader br = new BufferedReader( in:fr);
              FileWriter fw = new FileWriter( file:ficheroRes)) {

            // el condicional me permite leer línea a línea e ir sumándolas
            while ((lectura = br.readLine()) != null) {
                suma += Integer.parseInt( s:lectura);
            }

            // escribo en el .res
            fw.write(suma +""); // (al ponerlo sin las comillas me imprime un caracter, no el número)

            // imprimo por pantalla (salida estándar)
            System.out.println("La suma de los numeros del archivo es: " + suma);

        } catch (FileNotFoundException ex) {
            System.err.println( x:"File not found");
        } catch (IOException ex) {
            System.err.println( x:"IO Exception");
        }

    }

}
```

Ilustración 3. Código del programa Sumar.

Usando la terminal, se ejecuta el comando "java -jar Suma.jar contabilidad.txt" y devuelve el total sumado y genera el archivo contabilidad1.txt.res.

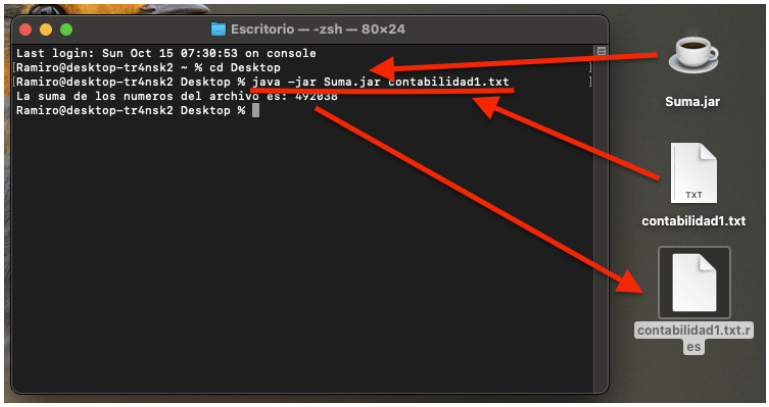

Ilustración 4. Terminal ejecutando el comando anterior.

Finalmente, un tercer programa (SumaTotales), que ejecuta el programa anterior y que lanza n procesos, tantos como ficheros pasados como argumentos que, calculando la suma de los números de dichos ficheros, devuelve la suma total de los totales de dichos ficheros y el tiempo de ejecución, en secuencial y en paralelo.

He creado métodos por separado para ejecutarlo de forma secuencial y paralela. Llamo a ambos desde el método main.

Secuencial:

```java
public class SumaTotales {

    public static void main(String[] args) {...6 lines }

    /** Ejecuta los procesos de forma secuencial ...6 lines */
    public static void sumaTotalSecuencial(String[] args) {

        int sumaTotal = 0;
        //inicio el temporizador
        long start = System.nanoTime();
        System.out.println( x: "\nEjecución SECUENCIAL\n");

        // bucle que lanza el proceso por cada argumento, generando un .res
        for (int i = 0; i < args.length; i++) {

            String[] command = {"/bin/bash", "-c", "java -jar Suma.jar " + args[i]};
            ProcessBuilder pb = new ProcessBuilder(command);
            Process p = null;

            try {
                // lanzo y detengo el proceso
                p = pb.start();
                p.waitFor();
                // leo el archivo .res
                FileReader fr = new FileReader(args[i] + ".res");
                BufferedReader br = new BufferedReader( in: fr);
                // imprimo por pantalla el resultado de cada proceso y lo almaceno en la suma total
                String linea = null;
                while ((linea = br.readLine()) != null) {
                    sumaTotal += Integer.parseInt( s: linea);
                    System.out.println("La suma del documento " + (i + 1) + " es: " + linea);
                }

            } catch (IOException ex) {
                System.err.println( x: "IO Exception");
            } catch (InterruptedException ex) {
                System.err.println( x: "Interrupted Exception");
            }

        }

        // paro el temporizador y calculo el tiempo total
        long finish = System.nanoTime();
        long tiempoTotal = finish - start;
        // finalmente imprimo la información
        System.out.println("El total de la suma de los documentos es: " + sumaTotal);

        System.out.println("Tiempo de ejecución en milisegundos: " + tiempoTotal / 1000000);

    }
```

Ilustración 5. Código de la ejecución secuencial del programa.

Paralela:

```java
/** Ejecuta los procesos de forma paralela ...6 lines */
public static void sumaTotalParalela(String[] args) {

    int sumaTotal = 0;
    long start = System.nanoTime();
    System.out.println( x:"\nEjecución PARALELA\n");
    ProcessBuilder pb = null;
    Process[] p = new Process[args.length];// necesito un array de procesos para realizar lo siguiente:

    // inicio los procesos hijos con un bucle
    for (int i = 0; i < args.length; i++) {
        String[] command = {"/bin/bash", "-c", "java -jar Suma.jar " + args[i]};
        pb = new ProcessBuilder(command);

        try {
            p[i] = pb.start();

        } catch (IOException e) {
            System.err.println( x:"IO EXception");
        }

    }

    // despues de que se hayan iniciado todos, los hago esperar a que terminen con otro bucle
    for (int i = 0; i < args.length; i++) {

        try {
            p[i].waitFor();

        } catch (InterruptedException ex) {
            System.err.println( x:"Interrupted Exception");
        }

    }

    // con otro bucle leo el archivo res, e imprimo y sumo el total
    for (int i = 0; i < args.length; i++) {

        try {
            FileReader fr = new FileReader(args[i] + ".res");
            BufferedReader br = new BufferedReader( in:fr);

            String linea = null;
            while ((linea = br.readLine()) != null) {
                sumaTotal += Integer.parseInt( s:linea);
                System.out.println("La suma del documento " + (i + 1) + " es: " + linea);
            }

        } catch (IOException ex) {
            System.err.println( x:"IO Exception");
        }

    }

    long finish = System.nanoTime();
    long tiempoTotal = finish - start;

    System.out.println("El total de la suma de los documentos es: " + sumaTotal);

    System.out.println("Tiempo de ejecución en milisegundos: " + tiempoTotal / 1000000);

}
}
```

Ilustración 6. Código de la ejecución paralela del programa.

Ejecuto el programa en la terminal del equipo con el siguiente comando “java -jar SumaTotales.jar contabilidad1.txt contabilidad2.txt contabilidad3.txt”, y obtengo los tres archivos.res correspondientes y el siguiente mensaje por consola:

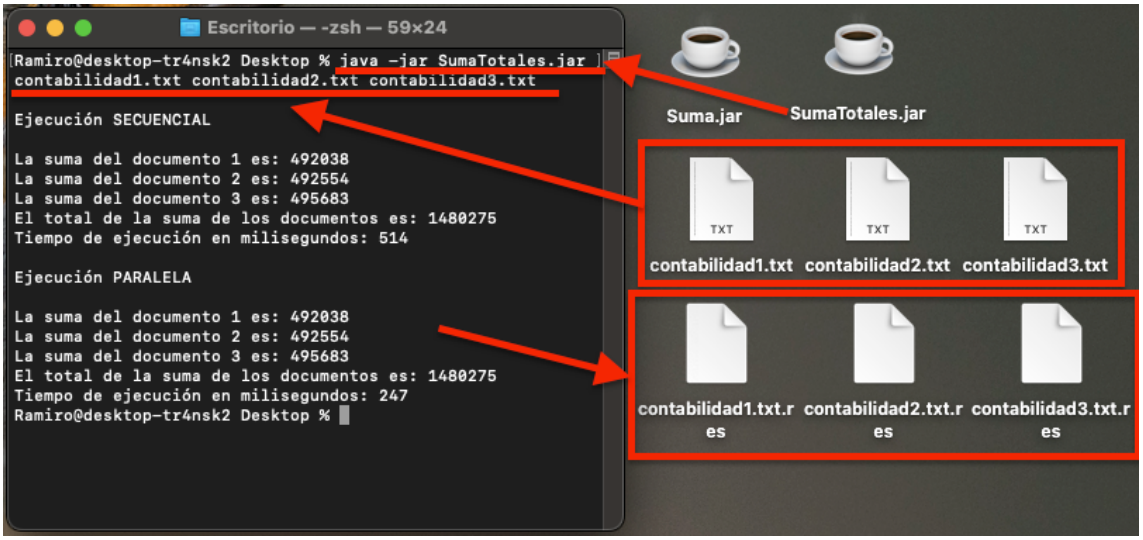

Ilustración 7. Terminal ejecutando el comando anterior.

2.- Comparación de tiempos entre ejecución secuencial y paralela.

Para realizar la comparación he utilizado tres ficheros diferentes en tres versiones cada uno. Con el proyecto GeneraFichero.jar he generado tres archivos de contabilidad en formato “Contabilidad.txt” que guardan escritos una serie de números. Los diferentes casos probados son: 50, 10000 y 10000000 de números por archivo.

Los datos corresponden a la ejecución, con el ordenador recién arrancado con pocos procesos corriendo por detrás. He intentado replicar más tarde, con más programas abiertos y los resultados no han vuelto a ser los mismos. Estos son los resultados obtenidos bajo esas condiciones descritas:

En el caso de los 50 números, la diferencia entre secuencial y paralelo es notable. 351 milisegundos de la ejecución secuencial, frente a 31 de la paralela.

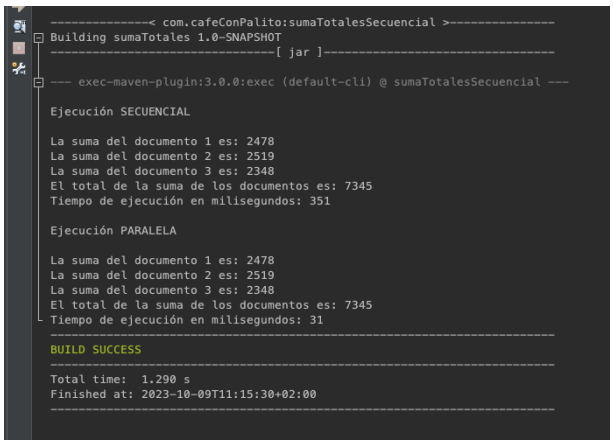

Ilustración 8. Ejecución del primer caso. 50 números por archivo.

Con 10000 números la diferencia aumenta, lógicamente. 411 milisegundos de la ejecución secuencial, frente a 30 de la paralela.

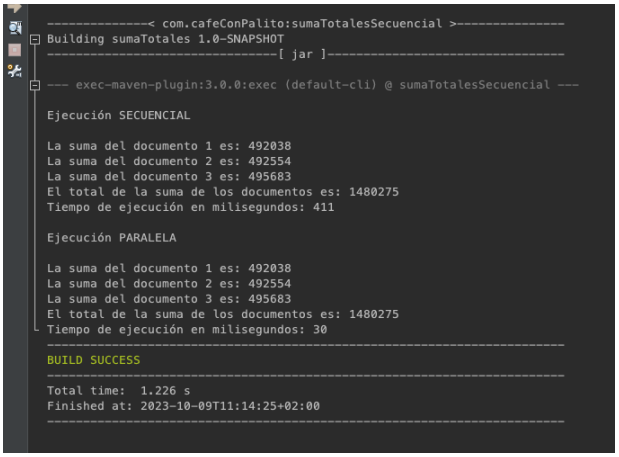

Ilustración 9. Ejecución del segundo caso. 10000 números por archivo

Con 10000000 de números, como era de esperar, la diferencia aumenta aún más. 939 milisegundos de la ejecución secuencial, frente a 32 de la paralela.

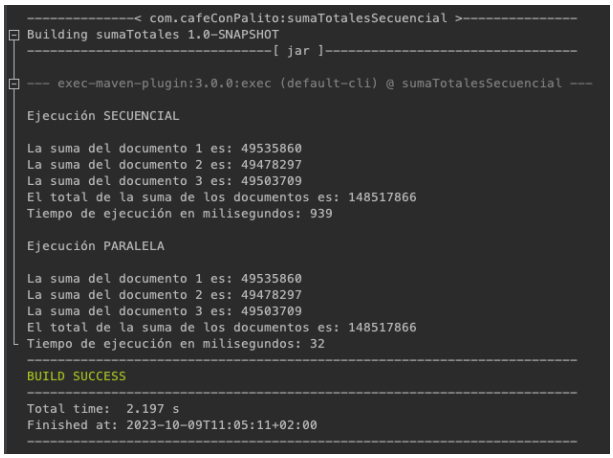

Ilustración 10. Ejecución del tercer caso. 10000000 de números por archivo.

3.- Conclusiones.

Ejecutar procesos en paralelo, en combinación, si fuese necesario, con la espera a fin de ejecución, resulta ser una forma muy efectiva de optimizar los tiempos de ejecución de un programa con varios procesos hijos. Si bien es cierto que no todos los procesos se pueden ejecutar en paralelo (aunque existen recursos para optimizar la espera a otros procesos), porque algunos dependen de otros, cuando es posible, es recomendable para agilizar y optimizar. Esta conclusión está apoyada en todo lo documentado anteriormente, comparando la ejecución secuencial con la paralela.

Resumiendo, las conclusiones son las siguientes:

1. Al ejecutar un programa en paralelo el tiempo de ejecución disminuye sustancialmente.
2. Al aumentar la complejidad del cálculo a realizar, la diferencia de los tiempos de ejecución entre secuencial y paralelo aumenta también. Para procesos sencillos la diferencia en mucho menor que para procesos complejos.
3. Pese a que requiere más trabajo por parte del programador, la diferencia en tiempos de ejecución en programas de alta complejidad, podría llegar a ser tan alta que, sin duda alguna, siempre que sea posible utilizar la ejecución paralela está más que justificado el trabajo extra.

Bibliografía.

- Medir tiempo de ejecución: https://www.techiedelight.com/es/measure-elapsed-time-execution-time-java/
- UD01-Programación multiproceso, Joaquín Franco Ros, Programación de Servicios y Procesos, 2023.

Ilustraciones.