



**IES AUGUSTO GONZALEZ DE
LINARES
DEPARTAMENTO DE INFORMATICA**


**GESTIÓN DE PROCESOS PARA
TAREAS EN PARALELO**

**PROGRAMACIÓN DE SERVICIOS Y
PROCESOS**

**GRADO SUPERIOR DE DESARROLLO DE
APLICACIONES MULTIPLATAFORMA**

2023/2024

Díez de Paulino, Albano

	Ciclo: Desarrollo de Aplicaciones Multiplataforma Modulo: Programación de Servicios y Procesos Título: Gestión de procesos para tareas en paralelo
--	--

Índice

1.	Demostración de la funcionalidad de la aplicación	2
2.	Comparación de tiempos entre la ejecución secuencial y paralela.....	7
3.	Conclusiones	10
	Tabla de Ilustraciones.....	11
	Bibliografía	11

1. Demostración de la funcionalidad de la aplicación

Para aprender a manejar la programación de procesos en paralelo se requiere que se realice la siguiente estructura de programas.

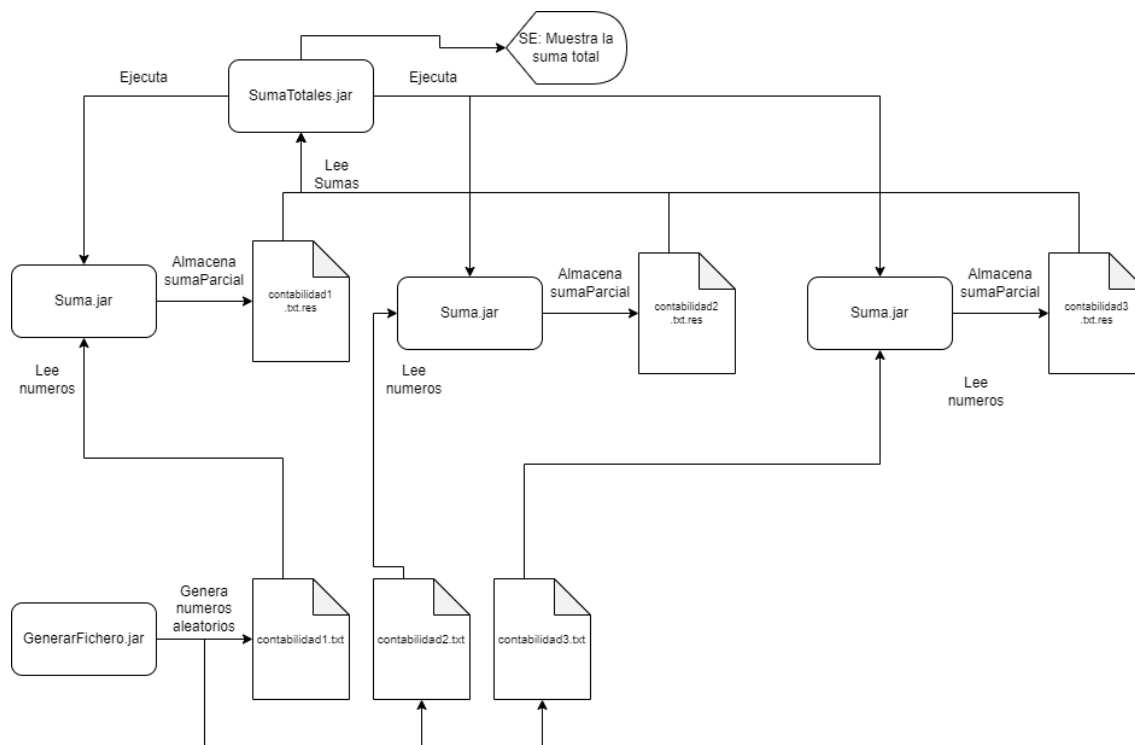


Ilustración 1 – Estructura de procesos planteada

Visto el problema que se plantea, la solución mas optima es realizar una clase que contenga el método de acceso Main desde el cual se llamaran a los 3 programas propuestos, que se encontraran encapsulados en diferentes métodos.

En mi caso he decidió crear una clase extra para que solo tenga los 3 métodos, y los declaro estáticos para no tener que declarar un objeto de la clase extra.

```
package com.cafeconpalito.main;

/**
 *
 * @author Albano Díez de Paulino
 */
public class Tarea2 {

    public static void main(String[] args) {

        //PARA ELEGIR QUE PROYECTO QUIERES USAR SOLO HAY QUE DESCOMENTAR EL CODIGO

        //Metodo para generar ficheros.txt con 10 numeros aleatorios, se le puede indicar el numero de ficheros
        Metodos.generarFichero(numeroFicheros: 3,numeros:100);

        //Metodo que suma los numeros de los ficheros generados en el apartado anterior y lo escribe en un fichero .res
        Metodos.suma(args[0]);

        //Metodo para hacer las sumas totales en secuencia
        Metodos.sumaTotalesSecuencial(argumentos: args);

        //Metodo para hacer las sumas totales en paralelo
        Metodos.sumaTotalesParalelo(argumentos: args);

    }
}
```

Código 1 - Método Main con llamada a los métodos

- **Método generarFichero (int numerosFicheros, int números)**

Método para generar n ficheros con extensión .txt con m números aleatorios, para ello se le pasa por parámetros el numero de ficheros a generar (primer parámetro) y la cantidad de números aleatorios que se quieren generar.

```
/**
 * Metodo para generar n ficheros con extension .txt con m numeros aleatorios
 *
 * @param numeroFicheros numero de ficheros a generar
 * @param numeros cantidad de numeros aleatorios que quieres generar
 */
public static void generarFichero(int numeroFicheros, int numeros) {
    for (int i = 0; i < numeroFicheros; i++) {
        File archivo = new File("contabilidad" + (i + 1) + ".txt");

        try (FileWriter fw = new FileWriter(file: archivo)) {

            for (int j = 0; j < numeros; j++) {
                fw.write((int) (Math.random() * 50) + "\n");
            }

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

    }
}
```

Código 2 - Método generarFichero

- **Método** suma (String nombre del fichero)

Método que suma los números del fichero que le pases como parámetro y genera un fichero .res

```
/**
 * Metodo que suma los numeros del fichero que le pases como parametro y
 * genera un fichero .res
 *
 * @param nombreFichero Fichero para sumar
 */
public static void suma(String nombreFichero) {

    long suma = 0;

    try (BufferedReader br = new BufferedReader(new FileReader(nombreFichero)); FileWriter fw = new FileWriter(nombreFichero + ".res")) {

        String cadena;

        while ((cadena = br.readLine()) != null) {
            suma += Long.parseLong(cadena);
        }

        fw.write(String.valueOf(suma));
        System.out.println(suma);

    } catch (FileNotFoundException e) {
        Logger.getLogger(Metodos.class.getName()).log(Level.SEVERE, null, e);
    } catch (IOException e) {
        Logger.getLogger(Metodos.class.getName()).log(Level.SEVERE, null, e);
    }
}
```

Código 3 - Método suma

Para ello se ha usado la clase FileReader que te permite leer carácter a carácter un fichero de texto, ya que leer de esa forma es poco lógica, se usa la clase BufferedReader, ya que esta clase crea un buffer para poder leer línea a línea, realizada la lectura solo queda escribir en otro fichero la suma parcial usando la clase FileWriter.

- **Método** sumaTotalesSecuencial(String[] argumentos)

Método para sumar las sumas Parciales del método suma, de forma secuencial.

Se le pasa por parámetro un array de String con el nombre de los ficheros .res generados con el método anterior.

Para lanzar los subprocessos desde este método se debe generar un .jar que solo realice el método suma, para ello desde el Main solo se debe llamar al método deseado.

```

/**
 * Metodo para sumar las sumas Parciales del metodo suma de forma secuencial
 *
 * @param argumentos ficheros de contabilidad
 */
public static void sumaTotalesSecuencial(String[] argumentos) {
    long startTime = System.nanoTime();

    long sumaTotal = 0;

    for (String i : argumentos) {
        String linea = null;

        ProcessBuilder pb = new ProcessBuilder(command:"CMD", command:"/c", "java -jar suma.jar " + i);

        Process p;
        try {
            //Se lanza el proceso hijo
            p = pb.start();
            //Se espera a finalizar
            p.waitFor();
        } catch (IOException ex) {
            Logger.getLogger(name: Metodos.class.getName()).log(level: Level.SEVERE, msg:null, thrown: ex);
        } catch (InterruptedException ex) {
            Logger.getLogger(name: Metodos.class.getName()).log(level: Level.SEVERE, msg:null, thrown: ex);
        }

        //Se lee la suma parcial y se suma a la total
        try (BufferedReader br = new BufferedReader(new FileReader(i + ".res"))) {

            String cadena;

            while ((cadena = br.readLine()) != null) {
                sumaTotal += Long.parseLong(:" cadena);
            }

        } catch (FileNotFoundException e) {
            Logger.getLogger(name: Metodos.class.getName()).log(level: Level.SEVERE, msg:null, thrown: e);
        } catch (IOException e) {
            Logger.getLogger(name: Metodos.class.getName()).log(level: Level.SEVERE, msg:null, thrown: e);
        }

        //Repetir mientras queden sumas parciales
    }
    System.out.println(":" sumaTotal);
    long endTime = System.nanoTime();
    long timeElapsed = endTime - startTime;
    System.out.println("Tiempo de ejecución: " + (timeElapsed / 1000000) + " milisegundos");
}

```

Código 4 - Método sumaTotalesSecuencial

- **Método sumaTotalesParalelo (String[] argumentos)**

Método para sumar las sumas Parciales del método suma de forma paralela.

Se le pasa por parámetro un array de String con el nombre de los ficheros .res generados con el método suma.

Al igual que en el método anterior hay que generar un .jar que solo realice el método suma.

```
public static void sumaTotalesParalelo(String[] argumentos) {
    long startTime = System.nanoTime();

    long sumaTotal = 0;
    ProcessBuilder pb = null;
    Process[] p = new Process[argumentos.length];

    //Bucle para lanzar todos los procesos hijos
    for (int i = 0; i < argumentos.length; i++) {
        String linea = null;

        pb = new ProcessBuilder(command: "CMD", command: "/c", "java -jar suma.jar " + argumentos[i]);

        try {
            p[i] = pb.start();
        } catch (IOException ex) {
            Logger.getLogger(name: Metodos.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
        }
    }
    //Bucle para esperar ha que terminen todos los procesos hijos lanzados antes
    for (Process i : p) {
        try {
            i.waitFor();
        } catch (InterruptedException ex) {
            Logger.getLogger(name: Metodos.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
        }
    }
    //Bucle para leer todas las sumas parciales
    for (String i : argumentos) {
        try (BufferedReader br = new BufferedReader(new FileReader(i + ".res"))) {

            String cadena;

            while ((cadena = br.readLine()) != null) {
                sumaTotal += Long.parseLong(s: cadena);
            }

        } catch (FileNotFoundException e) {
            Logger.getLogger(name: Metodos.class.getName()).log(level: Level.SEVERE, msg: null, thrown: e);
        } catch (IOException e) {
            Logger.getLogger(name: Metodos.class.getName()).log(level: Level.SEVERE, msg: null, thrown: e);
        }
    }
    System.out.println("=: sumaTotal);

    long endTime = System.nanoTime();
    long timeElapsed = endTime - startTime;
    System.out.println("Tiempo de ejecución: " + (timeElapsed / 1000000) + " milisegundos");
}
```

Código 5 – Método sumaTotalesParalelo

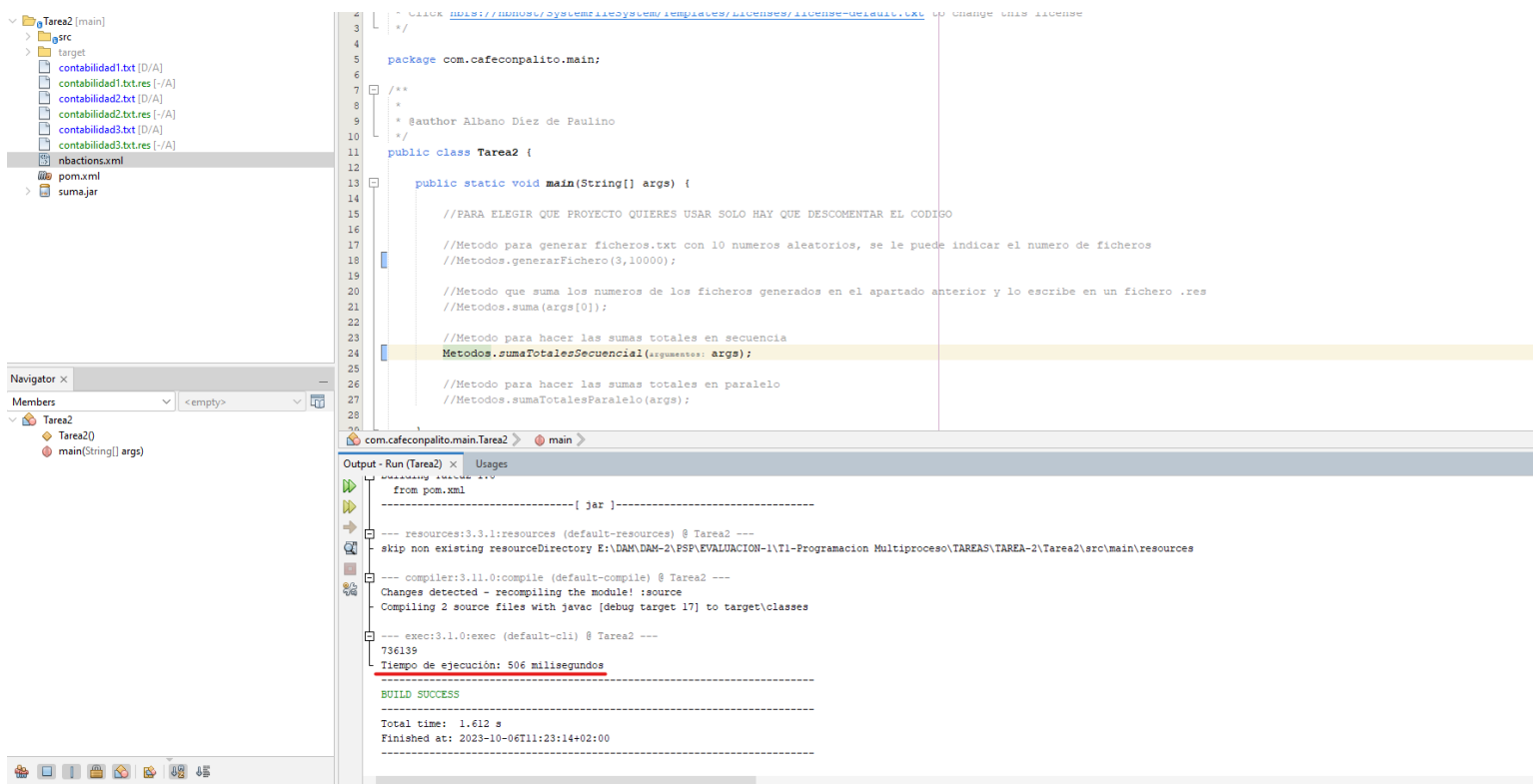
2. Comparación de tiempos entre la ejecución secuencial y paralela

Tras la realización del código se ha procedido a la comprobación del correcto funcionamiento de este, tras corregir pequeños fallos, por ejemplo declarar las sumas parciales y total como tipo int provocó que al tratar de sumar una cifra de números muy elevada, las variables desbordaran y dieran sumas falsas, se ha procedido a comparar los tiempos de ejecución entre lanzar los subprocesos de forma secuencial (*Lanzar uno y esperar a lanzar el siguiente*) y lanzar los subprocesos de forma paralela (*Lanzar todos los a la vez y esperar hasta que terminen todos*).

Desde un inicio se decidió usar unos ficheros de texto con 10.000 números ya que realizar la suma de menos números, el ordenador la iba a realizar en unidades más pequeñas que milisegundos y no sería útil para comparar tiempos.

Los tiempos con 10.000 números en tres ficheros son los siguientes:

- **Secuencial:** 506 milisegundos
- **Paralelo:** 208 milisegundos



```

1  package com.cafecanpalito.main;
2
3  /**
4   *
5   * @author Albano Díez de Paulino
6   */
7  public class Tarea2 {
8
9      public static void main(String[] args) {
10
11          //PARA ELEGIR QUE PROYECTO QUIERES USAR SOLO HAY QUE DESCOMENTAR EL CODIGO
12
13          //Metodo para generar ficheros.txt con 10 numeros aleatorios, se le puede indicar el numero de ficheros
14          //Metodos.generarFichero(3,10000);
15
16          //Metodo que suma los numeros de los ficheros generados en el apartado anterior y lo escribe en un fichero .res
17          //Metodos.suma(args[0]);
18
19          //Metodo para hacer las sumas totales en secuencia
20          Metodos.sumaTotalesSecuencial(args);
21
22          //Metodo para hacer las sumas totales en paralelo
23          //Metodos.sumaTotalesParalelo(args);
24      }
25  }

```

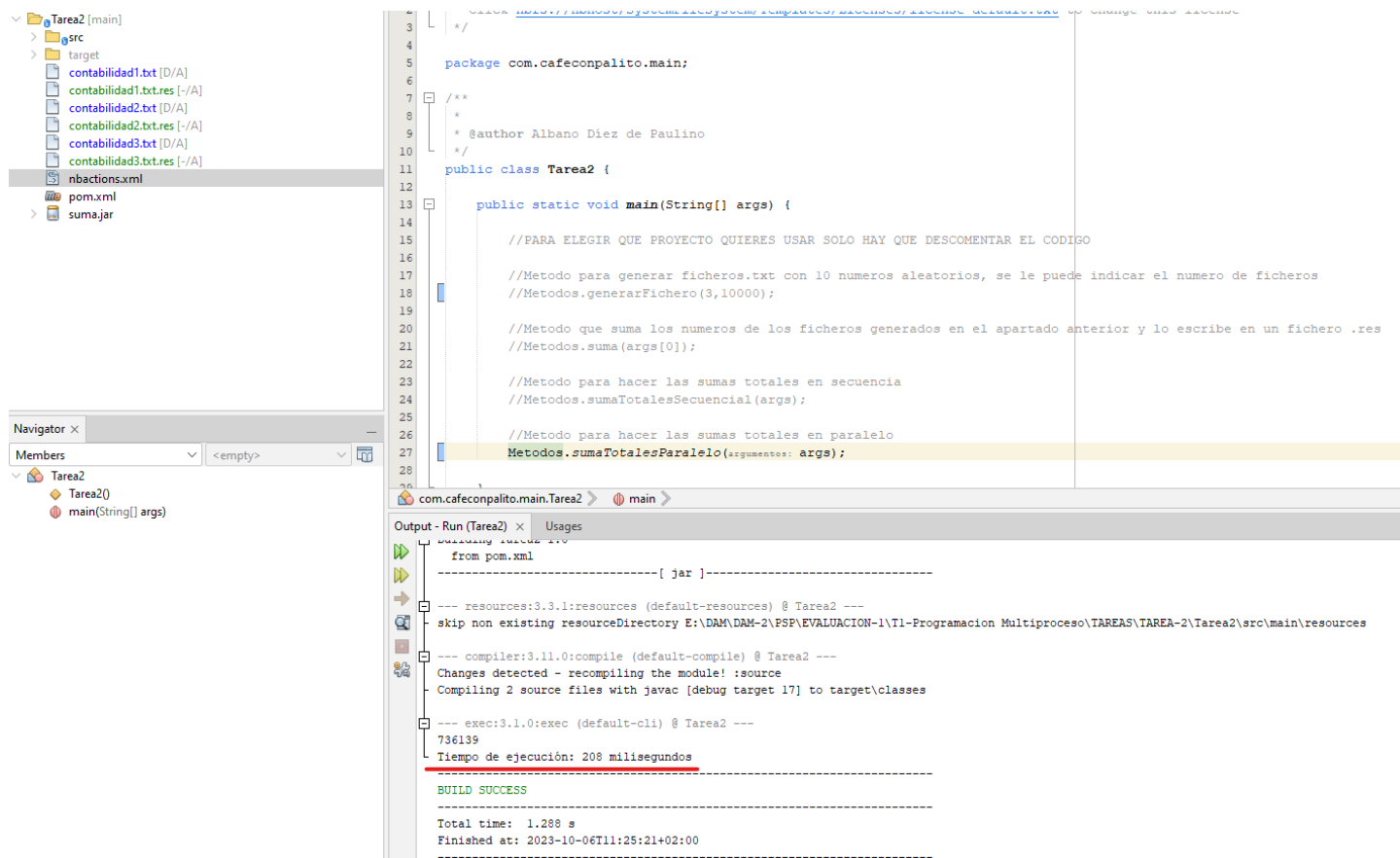
Output - Run (Tarea2) X Usages

```

from pom.xml
-----[ jar ]-----
--- resources:3.3.1:resources (default-resources) @ Tarea2 ---
skip non existing resourceDirectory E:\DAM\DAM-2\FSP\EVALUACION-1\T1-Programacion Multiproceso\TAREAS\TAREA-2\Tarea2\src\main\resources
--- compiler:3.11.0:compile (default-compile) @ Tarea2 ---
Changes detected - recompiling the module! :source
Compiling 2 source files with javac [debug target 17] to target\classes
--- exec:3.1.0:exec (default-cli) @ Tarea2 ---
736139
Tiempo de ejecución: 506 milisegundos
BUILD SUCCESS
Total time: 1.612 s
Finished at: 2023-10-06T11:23:14+02:00

```

Código 6 – Tiempo de ejecución 10K números de forma secuencial



```

1  package com.cafeconpalito.main;
2
3  /**
4   *
5   * @author Albano Díez de Paulino
6   */
7
8  public class Tarea2 {
9
10     public static void main(String[] args) {
11
12         //PARA ELEGIR QUE PROYECTO QUIERES USAR SOLO HAY QUE DESCOMENTAR EL CODIGO
13
14         //Metodo para generar ficheros.txt con 10 numeros aleatorios, se le puede indicar el numero de ficheros
15         //Metodos.generarFichero(3,10000);
16
17         //Metodo que suma los numeros de los ficheros generados en el apartado anterior y lo escribe en un fichero .res
18         //Metodos.suma(args[0]);
19
20         //Metodo para hacer las sumas totales en secuencia
21         //Metodos.sumaTotalesSecuencial(args);
22
23         //Metodo para hacer las sumas totales en paralelo
24         Metodos.sumaTotalesParalelo(argumentos: args);
25     }
26 }

```

Output - Run (Tarea2) x Usages

```

from pom.xml
-----[ jar ]-----
--- resources:3.3.1:resources (default-resources) @ Tarea2 ---
skip non existing resourceDirectory E:\DAM\DAM-2\PSP\EVALUACION-1\T1-Programacion Multiproceso\TAREAS\TAREA-2\Tarea2\src\main\resources
--- compiler:3.11.0:compile (default-compile) @ Tarea2 ---
Changes detected - recompiling the module! :source
Compiling 2 source files with javac [debug target 17] to target\classes
--- exec:3.1.0:exec (default-cli) @ Tarea2 ---
736139
Tiempo de ejecución: 208 milisegundos
BUILD SUCCESS
Total time: 1.288 s
Finished at: 2023-10-06T11:25:21+02:00

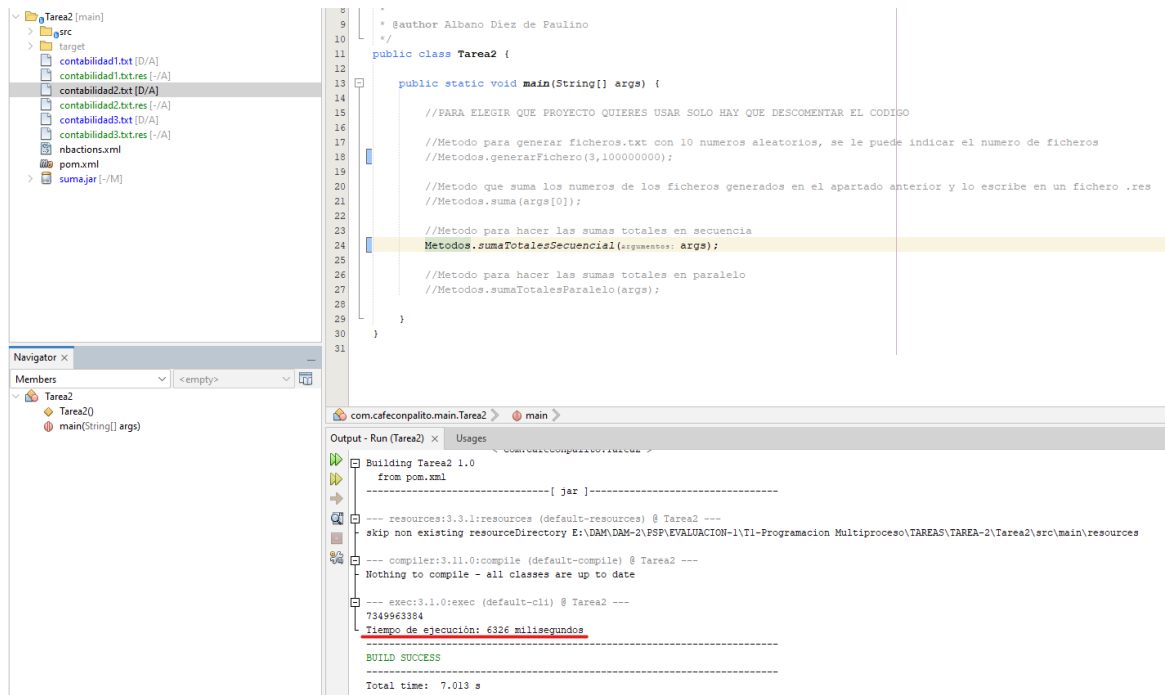
```

Código 7 – Tiempo de ejecución 10K números de forma paralela

Es evidente con esta prueba que la ejecución paralela es más rápida que secuencial, pero se decidió realizar la misma prueba, pero con 100 Millones de números por fichero, para descubrir si escalando la carga de trabajo se mantiene la premisa.

Los tiempos que salieron de esta prueba fueron los siguientes:

- **Secuencial:** 6326 milisegundos
- **Paralelo:** 2831 milisegundos



```

9  * @author Albano Díez de Paulino
10  */
11  public class Tarea2 {
12
13      public static void main(String[] args) {
14
15          //PARA ELEGIR QUE PROYECTO QUIERES USAR SOLO HAY QUE DESCOMENTAR EL CODIGO
16
17          //Metodo para generar ficheros.txt con 10 numeros aleatorios, se le puede indicar el numero de ficheros
18          //Metodos.generarFichero(3,100000000);
19
20          //Metodo que suma los numeros de los ficheros generados en el apartado anterior y lo escribe en un fichero .res
21          //Metodos.suma(args[0]);
22
23          //Metodo para hacer las sumas totales en secuencia
24          Metodos.sumaTotalesSecuencial(args);
25
26          //Metodo para hacer las sumas totales en paralelo
27          //Metodos.sumaTotalesParalelo(args);
28
29      }
30
31  }

```

com.cafeconpalito.main.Tarea2 > main

Output - Run (Tarea2) x Usages

Building Tarea2 1.0 from pom.xml

-----[jar]-----

--- resources:3.3.1:resources (default-resources) @ Tarea2 ---

skip non existing resourceDirectory E:\DAM\DAM-2\PSP\EVALUACION-1\T1-Programacion Multiproceso\TAREAS\TAREA-2\Tarea2\src\main\resources

--- compiler:3.11.0:compile (default-compile) @ Tarea2 ---

Nothing to compile - all classes are up to date

--- exec:3.1.0:exec (default-cli) @ Tarea2 ---

7349963384

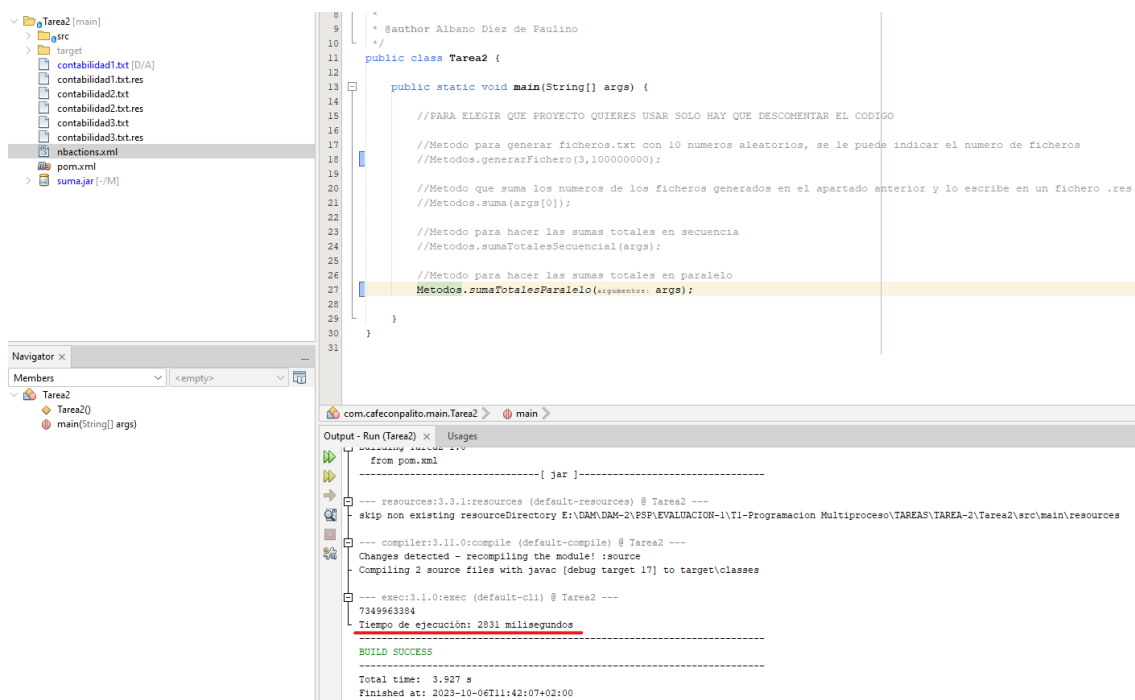
Tiempo de ejecución: 6326 milisegundos

BUILD SUCCESS

Total time: 7.013 s

Finished at: 2023-10-06T11:42:07+02:00

Código 8 - Tiempo de ejecución 100 M números de forma secuencial



```

9  * @author Albano Díez de Paulino
10  */
11  public class Tarea2 {
12
13      public static void main(String[] args) {
14
15          //PARA ELEGIR QUE PROYECTO QUIERES USAR SOLO HAY QUE DESCOMENTAR EL CODIGO
16
17          //Metodo para generar ficheros.txt con 10 numeros aleatorios, se le puede indicar el numero de ficheros
18          //Metodos.generarFichero(3,100000000);
19
20          //Metodo que suma los numeros de los ficheros generados en el apartado anterior y lo escribe en un fichero .res
21          //Metodos.suma(args[0]);
22
23          //Metodo para hacer las sumas totales en secuencia
24          //Metodos.sumaTotalesSecuencial(args);
25
26          //Metodo para hacer las sumas totales en paralelo
27          Metodos.sumaTotalesParalelo(args);
28
29      }
30
31  }

```

com.cafeconpalito.main.Tarea2 > main

Output - Run (Tarea2) x Usages

Building Tarea2 1.0 from pom.xml

-----[jar]-----

--- resources:3.3.1:resources (default-resources) @ Tarea2 ---

skip non existing resourceDirectory E:\DAM\DAM-2\PSP\EVALUACION-1\T1-Programacion Multiproceso\TAREAS\TAREA-2\Tarea2\src\main\resources

--- compiler:3.11.0:compile (default-compile) @ Tarea2 ---

Changes detected - recompiling the module! :source

Compiling 3 source files with javac [debug target 17] to target\classes

--- exec:3.1.0:exec (default-cli) @ Tarea2 ---

7349963384

Tiempo de ejecución: 2831 milisegundos

BUILD SUCCESS

Total time: 3.527 s

Finished at: 2023-10-06T11:42:07+02:00

Código 9 - Tiempo de ejecución 100 M números de forma Paralela



3. Conclusiones

Tras las pruebas realizadas se ha llegado a las siguientes conclusiones:

- Si se desea realizar que el programa tarde lo menos posible en ejecutar múltiples tareas, la mejor opción es realizarlas de forma paralela, antes que secuencial.
- Lanzar procesos de forma paralela es más difícil ya que se debe coordinar todos los procesos.
- Lanzar procesos de forma secuencial no requiere mucha coordinación.
- No se debe lanzar procesos con mucha carga de trabajo a la vez en ordenador con procesadores lentos, ya que puedes saturar el sistema.


	Ciclo: Desarrollo de Aplicaciones Multiplataforma Modulo: Programación de Servicios y Procesos Título: Gestión de procesos para tareas en paralelo
--	--

Tabla de Ilustraciones

Ilustración 1 – Estructura de procesos planteada.....	2
Código 1 - Método Main con llamada a los métodos	3
Código 2 - Método generarFichero	3
Código 3 - Método suma	4
Código 4 - Método sumaTotalesSecuencial	5
Código 5 – Método sumaTotalesParalelo	6
Código 6 – Tiempo de ejecución 10K números de forma secuencial	7
Código 7 – Tiempo de ejecución 10K números de forma paralela	8
Código 8 - Tiempo de ejecución 100 M números de forma secuencial	9
Código 9 - Tiempo de ejecución 100 M números de forma Paralela.....	9

Bibliografía

<https://www.techiedelight.com/es/measure-elapsed-time-execution-time-java/>

[ProcessBuilder \(Java Platform SE 8 \) \(oracle.com\)](#)

[BufferedReader \(Java Platform SE 8 \) \(oracle.com\)](#)

[FileReader \(Java Platform SE 8 \) \(oracle.com\)](#)

[FileWriter \(Java Platform SE 8 \) \(oracle.com\)](#)

[Process \(Java Platform SE 8 \) \(oracle.com\)](#)