# **Programación Paralela**

La programación paralela de un sistema multiprocesador se realiza mediante lenguajes que permitan expresar el paralelismo e intercambiar información de los procesadores entre si. (Dolores, 2000)

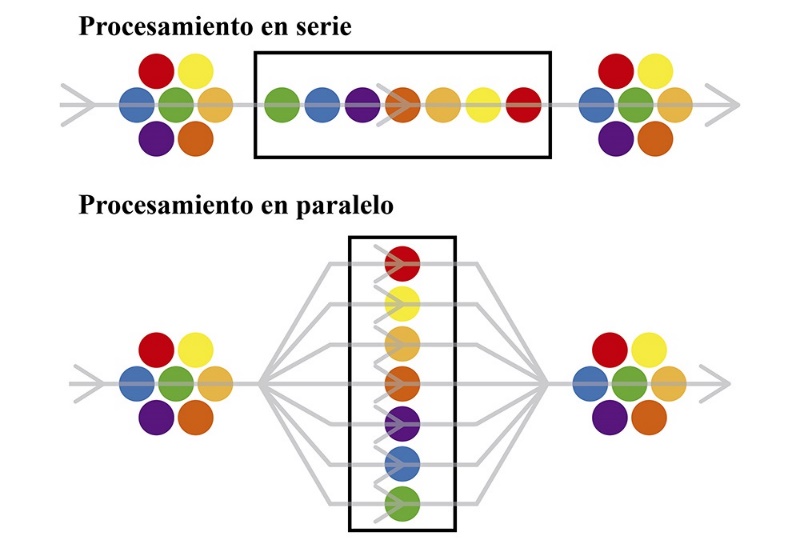
La programación paralela o programación concurrente es una técnica de programación basada en la ejecución simultánea, bien sea en un mismo ordenador (con uno o varios procesadores) o en un cluster de ordenadores, en cuyo caso se denomina computación distribuida.

Los sistemas multiprocesador consiguen un aumento del rendimiento si se utilizan estas técnicas. En los sistemas monoprocesador el beneficio en rendimiento no es tan evidente, ya que la CPU es compartida por múltiples procesos en el tiempo, lo que se denomina multiplexación.

El caso más simple de ejecución paralela consiste en el modelo de multiprogramación, en el cual varios programas secuenciales son ejecutados simultáneamente sobre diferentes procesadores sin ninguna interacción entre ellos.

Existen cuatro alternativas para construir un programa paralelo:

Paralelización manual usando pase de mensajes, Paralelización manual de sistemas de memoria compartida, Paralelización de datos y la Paralelización automática.



# **Ventajas**

* Resuelve problemas que no se podrían realizar en una sola CPU
* Resuelve problemas que no se pueden resolver en un tiempo razonable
* Permite ejecutar problemas de un orden y complejidad mayor
* Permite ejecutar código de manera más rápida (aceleración)
* Permite ejecutar en general más problemas
* Obtención de resultados en menos tiempo
* Permite la ejecución de varias instrucciones en simultáneo
* Permite dividir una tarea en partes independientes

# **Desventajas**

* Mayor consumo de energía
* Mayor dificultad a la hora de escribir programas
* Dificultad para lograr una buena sincronización y comunicación entre las tareas
* Retardos ocasionados por comunicación ente tareas
* Número de componentes usados es directamente proporcional a los fallos potenciales
* Condiciones de carrera
  + Múltiples procesos se encuentran en condición de carrera si el resultado de los mismos depende del orden de su llegada
  + Si los procesos que están en condición de carrera no son correctamente sincronizados, puede producirse una corrupción de datos

# **Paralelismo**

Paralelismo es la ejecución simultánea de dos o más tareas. Se considera una propiedad del hardware, ya que requiere recursos físicos para ejecutar cada tarea simultáneamente, y su objetivo se basa en realizar una tarea en el menor tiempo posible.

El paralelismo acelera la ejecución de una tarea dividiéndola en computaciones independientes y ejecutándola sobre hardware capaz de realizar computaciones simultáneas, como por ejemplo un procesador con varios núcleos. Pero cuando ejecutamos una tarea paralelizada en múltiples ordenadores, en vez de en los múltiples cores de un solo ordenador, decimos que la computación paralela es distribuída.

# **Hilo**

**Un Hilo o Thread es una parte del código que tiene la característica de ser ejecutado en forma paralela** es decir al mismo tiempo que otra parte del código. También tiene la limitación del hardware ya que la multitarea depende de la cantidad de núcleos que tenga el procesador, es decir si tiene 2 núcleos podrá ejecutar dos tareas al mismo tiempo.  
Existen dos planos de ejecución para un software el primer plano y el segundo plano

El Primer plano o **Foreground, es lo que el usuario ve en su pantalla**, que trabaja con botones, ventanas, campos de texto, imágenes, etc. Los cálculos más comunes se realizan en el primer plano o hilo principal de la aplicación.

El Segundo plano o **Background, es donde se procesan los hilos o la multitarea.** Mientas se muestra alguna pantalla, en el segundo plano deberíamos ejecutar todos los procesos pesados, que el usuario no ve que ocurren. Aqui ejecutan los servicios de software, los enlaces de red, los conexiones a bases de datos, los cálculos complejos, etc. Aquí los procesos pueden tardar el tiempo que necesiten mientras el usuario realiza otras tareas.

De la misma manera que un Sistema Operativo puede ejecutar varios procesos al mismo tiempo bien sea por concurrencia o paralelismo, dentro de un proceso puede haber varios hilos de ejecución.

Los hilos nos permiten separar tareas complejas en tareas más simples y concretas, así si un proceso falla no bloquea toda la aplicación.

# **Ejemplo de Hilo en Java**

Existen 2 formas de trabajar con hilos en cuanto a su creación se refiere:

* Heredando de la clase Thread
* Implementando la interfaz Runnable

mythread = new Thread() {

public void run() {

try {

System.out.println("...");

Thread.sleep(1000);

System.out.println("...");

} catch(InterruptedException e) {

System.out.println(e);

}

}

};

mythread.start();

Las aplicaciones Java ejecutan el método main() al iniciarse, allí es donde veremos iniciar también los hilos necesarios para nuestra aplicación salvo que necesitemos por requerimientos que algún evento los ponga en funcionamiento.

# **Programación paralela en java**

En Java para utilizar la multitarea debemos de usar la clase Thread (es decir que la clase que implementemos debe heredar de la clase Thread) y la clase Thread implementa la Interface Runnable.

public CajeraRunnable(Cliente cliente, long initialTime) {

this.cliente = cliente;

this.initialTime = initialTime;

}

@Override

public void run() {

System.out.println(""La cajera " + Thread.currentThread().getName()

+ "" COMIENZA A PROCESAR LA COMPRA DEL CLIENTE " + this.cliente.getNombre()

+ " EN EL TIEMPO: " + (System.currentTimeMillis() - this.initialTime) / 1000 +

"seg");

for (int i = 0; i < this.cliente.getCarroCompra().length; i++) {

*// Se procesa el pedido en X segundos*

this.esperarXsegundos(cliente.getCarroCompra()[i]);

System.out.println("Procesado el producto " + (i + 1) + " del " +

this.cliente.getNombre()+ "->Tiempo: " + (System.currentTimeMillis() –

this.initialTime) / 1000 + "seg");

}

System.out.println(""La cajera " + Thread.currentThread().getName() + ""

HA TERMINADO DE PROCESAR " + this.cliente.getNombre() + " EN EL TIEMPO: "

+ (System.currentTimeMillis() - this.initialTime) / 1000 + "seg");

}

private void esperarXsegundos(int segundos) {

try {

Thread.sleep(segundos \* 1000);

} catch (InterruptedException ex) {

Thread.currentThread().interrupt();

}

}

El concepto de **Java Parallel Stream** es un concepto sencillo de entender. En muchas ocasiones podemos tener un flujo de trabajo que necesitemos mejorar su rendimiento permitiendo su ejecución en paralelo a través de varios Threads. Es aquí donde el manejo de streams es una ventaja.

Sin embargo, podemos utilizar de una forma sencilla la programación en paralelo y modificar el número de Threads que se encargan de procesar este Stream.

1. public static void **main** (String[] args) {
2. long numero1=System.currentTimeMillis();
3. IntStream lista= IntStream.range(1, 10);
4. int total=lista.parallel().map(Principal::duplicar).sum();
5. long numero2=System.currentTimeMillis();
6. System.out.println(numero2-numero1);
7. System.out.println(total);
8. }
9. public static int **duplicar**( int numero) {
10. **try** {
11. Thread.sleep(1000);
12. } **catch** (InterruptedException e) {
13. // TODO Auto-generated catch block
14. e.printStackTrace();
15. }
16. **return** numero\*2;
17. }
18. }