Multiprocesamiento (hilos)

Salvador López Mendoza

Diciembre de 2022

Introducción

En la vida diaria realizamos muchas actividades al mismo tiempo.

Hablar, escuchar, comer, atender la clase.

¿Se pone toda la atención en una actividad?

¡No! Se reparte la atención a varias actividades.

Los seres humanos podemos realizar varias actividades al mismo tiempo.

Computadoras

Modelo sencillo para entender el funcionamiento de una computadora:

- La computadora tiene un procesador.
- El procesador solamente procesa una instrucción a la vez.
- Los programas constan de una secuencia finita de instrucciones.
- El procesador interpreta cada instrucción.

Computadoras (II)

¡Mi computadora realiza muchas cosas al mismo tiempo!

¿Cómo se logra que una computadora (aparentemente) realice varias actividades al mismo tiempo?

Cuando hay varios procesadores trabajando al mismo tiempo se tiene un **sistema en paralelo**.

Cuando un procesador ejecuta varios procesos dando la impresión de que trabajan al mismo tiempo se tiene un **sistema concurrente**.

Recursos compartidos

- Una de las bases de los **sistemas operativos** es compartir su conjunto finito de recursos entre los múltiples procesos.
- Cada proceso corre en su propio *espacio de direcciones* que evita que los procesos interfieran entre sí.
- Esto se ha trabajado desde los 60's y 70's del siglo XX.
  Java tomó varios de estos conceptos y creó los llamados *threads* o hilos.

Procesos

El proceso es el concepto central de todo sistema operativo.

Cuando se pide a la computadora que se ejecute un programa éste se convierte en un proceso,

Se dice que **el proceso es un programa en ejecución.**

Modelo del proceso

Todas las actividades de la computadora se organizan mediante **procesos secuenciales** (procesos).

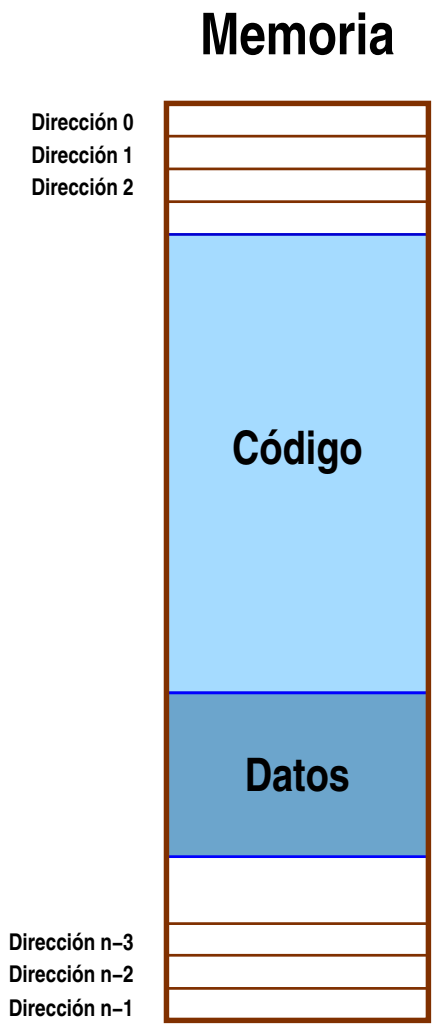
Un proceso tiene asignado un espacio en la memoria principal de la computadora (RAM):

Código Conjunto de bytes que corresponden a la codificación en binario de las instrucciones del programa.

Datos Conjunto de bytes que corresponden al espacio necesario para todas las variables que requiere el programa.
La información se codifica en binario.

Las instrucciones se ejecutan secuencialmente. Se dice que un proceso secuencial tiene un único **hilo de control**.

Proceso en memoria



Tareas complejas

Existen muchas situaciones en las que conviene tener varias actividades trabajando al mismo tiempo.

Los programas que realizan estas tareas se pueden ver como un conjunto de procesos que trabajan (casi) al mismo tiempo. Se pueden crear varios procesos independientes que trabajen concurrentemente.

Programa reproductor de videos

Diseñar una aplicación para reproducir videos que se descargan de Internet.
La aplicación debe permitir que durante la reproducción del video se realicen las siguientes actividades:

- *Aumentar o disminuir el volumen del audio.*
- *Avanzar o retroceder la reproducción del video.*
- *Detener o reanudar la reproducción del video.*

Programa reproductor de videos (II)

Propuestas de solución:

- Un proceso que realice todas las actividades.
  Posiblemente, utiliza interrupciones para manejar los controles durante la reproducción.
- Lanzar un conjunto de procesos que se ejecuten concurrentemente.
  Cada proceso se encarga de alguna de las actividades: control de volumen, control de reproducción, etc.
  Se requiere de mecanismos de comunicación entre los procesos.
- Lanzar un *conjunto de hilos* que se ejecutan concurrentemente.
  Cada hilo se encarga de alguna de las actividades: control de volumen, control de reproducción, etc.

Hilos

Definición

Los **hilos** son procesos que tienen su propia secuencia de instrucciones, pero **comparten** el espacio de direcciones y todos los datos que se encuentran en ese espacio de direcciones.

En algunos casos se les llama *procesos ligeros*.

Crear un nuevo hilo consume menos recursos que crear un nuevo proceso, pues no se tiene que definir un nuevo espacio de direcciones para ese hilo.

Es más rápido y sencillo crear (y destruir) hilos.

Concurrencia en Java

- Una aplicación en Java puede constar de varios hilos de control, cada uno corriendo concurrentemente con otros.
- Todas las aplicaciones en Java tiene al menos un hilo de control llamado `main`, en el cual se ejecuta el método principal.
- Un hilo individual ejecuta sus instrucciones en orden secuencial, una después de otra.

Concurrencia. Ventajas

Al tener varios hilos se puede

- Modelar conceptualmente actividades simultáneas que toman lugar en el mundo real.
- Modularizar el sistema definiendo actividades independientes.
- Utilizar tiempos ociosos de espera para mejor rendimiento, e
- Incrementar la receptividad en tanto se está ocupado con tareas grandes tales como:
  - IO, animación,
  - lectura de e-mails, y descarga de archivos,
  - recibir y despachar eventos en una GUI (clicks de botones, etc.)

Threads en Java

- Un *thread* es una unidad de programa que se ejecuta independientemente de las otras partes del programa.
- Un hilo es un ejemplar de la clase `java.lang.Thread`.
  La clase `Thread` tiene los métodos:
  `run`, `sleep`, `start`, `interrupt`, `join`, `yield`, `isAlive` e `isInterrupted`.
- Hay dos formas de crear hilos:
  - Creando un objeto de la clase `Thread`.
  - Implementando la interfaz `Runnable`.

Threads en Java (II)

- Todos los hilos son ejemplares de la clase `Thread`.
  Lo natural es crearlos extendiendo dicha clase.
- El método `run` es el punto de entrada a la ejecución del hilo, debe sobreescribirse en las subclases.
- La ejecución del hilo termina al terminar la ejecución secuencial de las instrucciones en el método `run`.

```
public class miHilo extends Thread {
  public void run () {
     // Cuerpo
  }
  // Otros métodos
}
```

Extendiendo la clase Thread

- Para iniciar un nuevo hilo definido en `miHilo`, se debe crear un ejemplar de esta clase y llamar al método `start()`, el cual invoca indirectamente a `run`.

  ```
  new miHilo().start();
  ```

- El método `run` no se invoca directamente, hacerlo podría causar que la ejecución fuera en el hilo llamador no en un nuevo hilo.

Ejemplo

```
public class EjemploThread {
  public static void main(String[] args){
    ContadorThread segundoHilo = new ContadorThread();

    if(!segundoHilo.isAlive())
      System.out.println("Segundo hilo aún no está vivo.");

    segundoHilo.start();
    if(segundoHilo.isAlive())
      System.out.println("Segundo hilo está vivo ahora.");

    Thread.yield();
    if(!segundoHilo.isAlive())
      System.out.println("Segundo hilo ya no está vivo.");

    System.out.println("Termina el primer hilo.");
  }
}
```

Ejemplo

```
    // Los objetos de esta clase corren en un hilo aparte.
class ContadorThread extends Thread {
  public void run(){
    final int max = 10;

    System.out.println("Empieza ejecución de este hilo") ;
    for(int i = 1; i <= max; i++)
      System.out.print(i + " ") ;

    System.out.println();
    System.out.println("Termina ejecución de este hilo") ;
  }
}
```

Ejecución

```
Segundo hilo aún no está vivo.
Segundo hilo está vivo ahora.
Empieza ejecución de este hilo
1 2 3 4 5 6 7 8 9 10
Termina ejecución de este hilo
Segundo hilo ya no está vivo.
Termina el primer hilo.
```

Este ejemplo no usa el potencial de los hilos pues no se trabaja concurrentemente, sólo sirve para ilustrar los conceptos básicos de la creación de hilos y los principales estados de su ciclo de vida.

Interfaz Runnable

Otra forma de crear hilos es extendiendo la interfaz `Runnable` que tiene sólo el método `run`.

```
public interface Runnable {
  public abstract void run() ;
}
```

Interfaz Runnable (II)

1. Escribir una clase que implemente la interfaz `Runnable`.
2. Colocar el código para la tarea en el método `run` de la clase.

```
public class MiRunnable implements Runnable {
      public void run() {
         . . .
      }
   }
```

3. Crear un objeto de esta clase.

   ```
   Runnable r = new MiRunnable();
   ```

4. Construir un objeto de la clase `Thread` pasándole como parámetro el objeto de la clase `Runnable`

   ```
   Thread t = new Thread(r);
   ```

5. Llamar al método `start` del hilo.

   ```
   t.start();
   ```

Esta es mejor opción para poder usarlo en una clase que extienda otra.

Ejemplo

```
public class EjemploRunnable {
  public static void main(String[] args){
    Runnable contador = new Contador();
    Thread segundoHilo = new Thread(contador,"contador");
    if(!segundoHilo.isAlive()){
      System.out.println("Segundo hilo aún no está vivo.");
    }
    segundoHilo.start();
    if(segundoHilo.isAlive()){
      System.out.println("Segundo hilo está vivo ahora.");
    }
    Thread.yield();
    if(!segundoHilo.isAlive()){
      System.out.println("Segundo hilo ya no está vivo.");
    }
    System.out.println("Termina el primer hilo.");
  }
}
```

Ejemplo (II)

```
class Contador implements Runnable {
  public void run(){
    final int max = 10;
     System.out.println("Empieza la ejecución.");
    for(int i = 1; i <= max; i++){
      System.out.print(i + " ");
    }
    System.out.println();
    System.out.println("Termina la ejecución.");
  }
}
```

Planeación y prioridad de los hilos

- Cuando hay más de un hilo ejecutable, Java necesita decidir a cuál hilo asignar el CPU y hacer la ejecución concurrente.
- Esta tarea se deja a un sistema llamado *scheduler* o planificador, el cual tiene una cola de prioridad con los hilos que están en estado ejecutable.
- La prioridad del hilo está en el rango de `MIN_PRIORITY(1)` y `MAX_PRIORITY(10)`, asignada por la clase `Thread` al crearlo.
- Una vez que un hilo está en ejecución, los otros hilos tienen oportunidad de ejecutarse dependido de la implementación del planificador:
  - Algunos planeadores permiten que el hilo en ejecución retenga el control del CPU mientras su estado sea `ejecutable`.
  - Algunos planeadores asignar una *rebanada de tiempo* limitada al proceso actual. La cantidad de tiempo se conoce como un *quantum*.

Planeación y prioridad de los hilos (II)

- Con ambos enfoques, un hilo puede ser forzado a ceder el control antes que haya terminado.
- La falta de predictibilidad de cuánto trabajo debe hacer un hilo durante el tiempo que tiene el procesador es una de las cosas que hace difícil la programación de hilos seguros.
- Un hilo puede dejar de ser el hilo actual cediendo voluntariamente el control.
- La prioridad de un hilo puede ser independiente del nivel heredado de su creador vía el mensaje `setPriority`, como se ve en el ejemplo:

Asignación de prioridad

```
public class EjemploPrioridad {
  public static void main(String[] args){
    Runnable contador = new Contador();
    Thread segundoHilo = new Thread(contador,"contador");
    if(!segundoHilo.isAlive())
      System.out.println("Segundo hilo aún no está vivo.");
    Thread miHilo = Thread.currentThread();
    segundoHilo.setPriority(miHilo.getPriority()+1);
    segundoHilo.start();
    if(segundoHilo.isAlive())
      System.out.println("Segundo hilo está vivo ahora.");
    Thread.yield();
    if(!segundoHilo.isAlive())
      System.out.println("Segundo hilo ya no está vivo.");
    System.out.println("Termina el primer hilo.");
  }
}
```

Asignación de prioridad (II)

```
class Contador implements Runnable {
  public void run(){
    final int max = 10;
     System.out.println("Empieza la ejecución.");
    for(int i = 1; i <= max; i++){
      System.out.print(i+" ");
    }
    System.out.println();
    System.out.println("Termina la ejecución.");
  }
}
```

Ejecución

La salida de este programa es:

```
Segundo hilo aún no está vivo.
Empieza la ejecución.
1 2 3 4 5 6 7 8 9 10
Termina la ejecución.
Segundo hilo ya no está vivo.
Termina el primer hilo.
```