MEDAC

Instituto Oficial de Formación Profesional

TEMA 4. INTRODUCCIÓN A LA PROGRAMACIÓN PARALELA O MULTIHILO

PROGRAMACIÓN DE
SERVICIOS Y PROCESOS
2º DAM

Curso 2021-2022

TEMA 4. INTRODUCCIÓN A LA PROGRAMACIÓN PARALELA O MULTIHILO

APARTADO 1. OBJETIVOS

- Aprender el concepto de hilo.
- Comprender la diferencia entre hilos y procesos.
- Conocer los estados por los que pueda pasar un hilo en su ejecución.
- Manejar las clases que aporta Java para la gestión de hilos.

¿Por qué a la misma vez que escuchamos música en el navegador podemos descargar un fichero?

Esto es debido a que los navegadores web están usando la programación multihilo o paralela, que es una programación concurrente capaz de ejecutar dos o más hilos o tareas a la vez , normalmente repartiendo el tiempo de proceso entre las tareas.

¿Alguien sabe lo que es un hilo?

No es más que un trozo de código que **se ejecuta dentro de un proceso**, pero con una gran ventaja y es que puede ser ejecutado en paralelo con otros hilos. A los hilos también se le conoce como hebra.

¿Qué significa que los hilos se ejecuten dentro de un proceso?

Que el proceso puede crear y lanzar hilos, pero los hilos solo podrán utilizar los recursos de los que disponga el proceso. Un proceso **NO** tiene acceso a los datos de otros procesos. Sin embargo, un hilo **SÍ** accede a los datos de otro hilo.

Los procesos que estén activos van a seguir en ejecución hasta que todos y cada uno de los hilos que han lanzado terminen de ejecutarse. En ese momento, el proceso podrá ser destruido por el SO y todos los recursos de los que disponía serán liberados.

Cuando ejecutamos un programa, se crean: UN PROCESO y UN HILO PRIMARIO

Cosas importantes a tener en cuenta sobre los hilos:

- Los hilos no pueden existir de forma independiente a un proceso.
- Los hilos no se pueden ejecutar por sí solos.
- Dentro de un mismo proceso vamos a poder tener tantos hilos ejecutándose como necesitemos.

Si contamos con un sólo hilo vamos a tener algo muy parecido a un programa secuencial normal y corriente. La gran diferencia la encontramos al ejecutar varios hilos de forma concurrente.

Entonces, ¿Cuál es la diferencia entre proceso e hilo?

Los hilos se podrán ejecutar sólo dentro de un proceso, es decir, que van a depender de un proceso para ejecutarse. Los procesos son independientes unos de otros teniendo zonas de memoria distintas cosa que en los hilos no es así.

Las ventajas de los hilos podemos decir que aparecen cuando tenemos más de uno, es decir, cuando tenemos un programa multihilo. En este caso dentro de un mismo proceso vamos a tener muchos hilos en ejecución, los cuales estarán realizando actividades totalmente distintas, pudiendo o no colaborar entre ellos.

Si los comparamos con los procesos, los hilos tienen muchas más ventajas, como son:

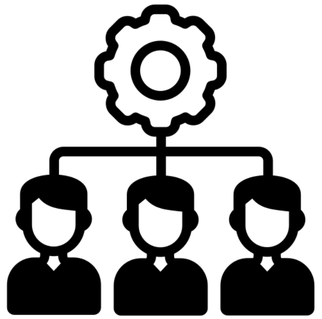
- **Compartir recursos:** los hilos que estén dentro de un proceso van a compartir tanto la memoria como los recursos de los que disponga. Debido a esto todas las operaciones de los hilos serán mucho más rápidas.
- **Uso más eficiente y ahorro de memoria:** como comparten la misma zona de memoria, a la hora de crear nuevos hilos en el proceso no va a suponer una reserva adicional de memoria.
- **Capacidad de respuestas:** al tener varios hilos va a permitir al proceso atender otras peticiones que le envíe el usuario a través de otros hilos.
- **Paralelismo real:** si contamos con un sistema en el que el procesador es multinúcleo, los hilos se pueden ejecutar en núcleos distintos, lo que permite usar el procesador de forma paralela, haciendo que se ejecuten varias instrucciones al mismo tiempo.

Pero, como todo, los hilos también tienen algunas desventajas:

- No son soportados por todos los lenguajes de programación, aunque, actualmente, casi todos lo hacen.
- El programador es el encargado de controlar todos los problemas que estén relacionados con la comunicación y sincronización de los hilos.

Por lo tanto, ¿Qué es mejor utilizar hilos o procesos?

Pues dependerá del problema a resolver, si tenemos un único problema con varias partes identificables, podremos usar **hilos**. Pero, si contamos con varios problemas diferentes dentro del mismo, podremos usar **procesos**.



Los hilos están formados por los siguientes elementos:

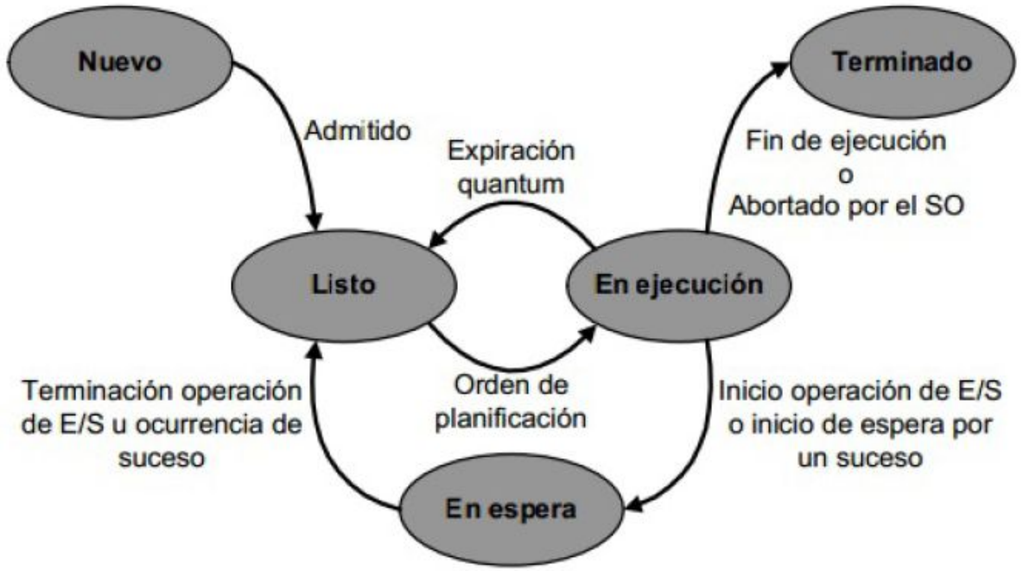
- Un **identificador único**, con el que podremos identificar a cada hilo de forma rápida.
- Un **contador de programa**, a través del cual el hilo podrá ejecutar su código de forma independiente.
- Un **conjunto de registros asociados**, con los que el hilo puede realizar todas las operaciones aritmético-lógicas que necesite de forma independiente.
- Una **pila** propia, con las que el hilo puede realizar las llamadas a las funciones que necesite de forma independiente.

Los hilos, entre ellos, también pueden compartir recursos, por ejemplo:

- El código a ejecutar.
- Variables globales que se encontrarán en la zona crítica.

APARTADO 6. ESTADOS DE UN HILO

Igual que los procesos, los hilos tienen un ciclo de vida determinado por una serie de estados. Estos estados son iguales tanto para los hilos que crea el usuario como para los que crea el sistema (hilos demonio o de sistema). El comportamiento de cada hilo va a depender del estado en el que se encuentre, por lo que va a definir la operación que está realizando.



APARTADO 6. ESTADOS DE UN HILO

Los estados por los que pasa un hilo son:

- **Nuevo**: aquí el hilo ha sido creado y está listo para ejecutarse, aunque aún no haya sido elegido para empezar a ejecutarse.
- **Listo o ejecutable**: el hilo va a entrar en este estado cuando indiquemos, con su método correspondiente, que está listo para poder ejecutarse.
- **En ejecución**: aquí el hilo está listo para ejecutarse y el SO lo ha seleccionado para que se ejecute.
- **Bloqueado**: el hilo entra en este estado cuando necesita algún recurso de E/S que debe proporcionarle el usuario o el SO. En este estado, no se le va a asignar tiempo de CPU al hilo, aunque estará listo para volver a ser usado.
- **Finalizado**: Cuando el hilo termina su ejecución, pasa a este estado, en espera de que el SO lo destruya y libere sus recursos.

Java nos ofrece varias clases para realizar programas que usen la programación paralela o multihilo. Dentro del paquete **java.lang** encontramos todas estas utilidades:

- Clase **Thread**: con esta clase podemos crear hilos a los que le podremos asignar el código que queramos para que lo ejecuten. Si queremos que una clase sea hilo deberá heredar de esta.
- Interfaz **Runnable**: con esta interfaz, vamos a poder añadir la funcionalidad de hilo a cualquier otra clase por el mero hecho de implementarla.
- Clase **ThreadDeath**: con esta clase podremos manejar y notificar errores en el uso de los hilos. Hereda de la clase Error.
- Clase **ThreadGroup**: con esta vamos a manejar un grupo de hilos de forma conjunta, haciendo que se ejecuten de una forma bastante más eficiente.

Dentro de la clase Thread, podemos encontrar, entre muchos otros, los siguientes métodos:

- **new()**: crea un hilo.
- **start()**: indica que el hilo está listo para ejecutarse.
- **run()**: ejecuta el hilo, que empezará a ejecutarse de forma paralela.
- **sleep()**: hace que el hilo se bloquee una cantidad de tiempo dada en milisegundos.
- **wait()**: hace que el hilo espere la ejecución de otra tarea para volver a ejecutarse.
- **getState()**: nos devuelve el estado en el que se encuentra actualmente el hilo.

CLASE THREAD

Para crear un hilo lo haremos mediante una instancia de la clase **Thread**, que nos va a permitir gestionar completamente nuestros hilos.

Para crear una clase con la funcionalidad de hilo en Java, debemos seguir los siguientes pasos:

- Crear una clase que herede de la clase Thread.
- Reescribir el método run, en el que debemos poner todo el código que queramos que ejecute nuestro hilo.
- Crear un objeto de la nueva clase, que será el hilo con el que trabajaremos.

```
public class Hilo1 extends Thread{

    @Override
    public void run(){

    }

}
```

Dentro de la clase, podremos crear todas las variables y métodos que necesitemos, incluidos *constructores*, métodos *get*, *set* y *toString*, ya que lo que estamos creando es una clase sin ningún tipo de peculiaridad.

Mediante los constructores, podremos crear los hilos con los datos que necesitemos, y con los métodos get y set, podremos obtener o modificar sus variables sin ningún tipo de problema.

Normalmente, dentro del método run, deberemos crear un bucle infinito, ya que la hebra estará ejecutándose de forma constante hasta que decidamos finalizarla o hasta que finalice el propio programa.

INTERFAZ RUNNABLE

Además de la forma anterior, Java nos permitirá crear hilos implementando la interfaz Runnable, la cual nos ofrecerá, igualmente, gestionar completamente los hilos de nuestros programas.

Para crear una clase con la funcionalidad de hilo de esta forma en Java, debemos seguir los siguientes pasos:

- Crear una clase que implemente la interfaz Runnable.
- Reescribir el método run, en el que debemos poner todo el código que queramos que ejecute nuestro hilo.
- Crear un objeto de la nueva clase, que será el hilo con el que trabajaremos.

```
public class Hilo2 implements
Runnable{

    @Override
    public void run(){

    }

}
```

Al implementar una interfaz, tendremos que reescribir el método run de forma obligatoria, cosa que no ocurría al heredar de Thread. Si en este caso, no reescribimos el método, no tendrá funcionalidad.

Al igual que ocurría con el método anterior, en esta clase podremos crear todos los atributos y métodos que vayamos a necesitar, incluyendo *constructores*, métodos *get*, *set* y *toString*.

También, en el método run, debemos crear un bucle infinito para que la hebra se quede en segundo plano, ejecutándose constantemente hasta que decidamos finalizarla o hasta que finalice el propio programa

```
public class Hilo1 extends Thread{

    @Override
    public void run(){
        while(true){
          /* Código a ejecutar por
            el hilo*/
        }
    }
}
```

Cuando creamos un objeto de una clase que tiene la funcionalidad de hilo mediante la herencia de Thread, no bastará con crear el objeto con **new()**.

Para que este hilo sea ejecutado y pase a segundo plano, debemos hacer que pase al estado Ejecutándose y, para ello, debemos iniciarlo mediante el método **start()**. Hay que tener mucho cuidado de no utilizar el método **run()**, ya que no hará que se ejecute el hilo de forma paralela. A fin y al cabo, el método **start()** hará que se ejecute el método **run()** de forma paralela.

El método start() va a realizar las siguientes tareas:

- Reservar todos los recursos necesarios para la correcta ejecución del hilo.
- Llamar al método run() y hacer que se ejecute de forma paralela.

```
public static void main (String[] args){
  //Creamos el objeto
  Hilo1 hilo= new Hilo1();
  //Lanzamos el hilo
  hilo.start()
}
```

Si, por el contrario, hemos creado un hilo implementando la interfaz Runnable, debemos seguir los siguientes pasos:

- Crear un objeto de la clase hilo.
- Crear un objeto de la clase Thread mediante el objeto anterior.
- Llamar al método run() y hacer que se ejecute de forma paralela.

Hay que tener en cuenta que, una vez hemos llamado al método start() de un hilo, no podremos hacerlo de nuevo, ya que el hilo se está ejecutando. En caso de hacerlo, tendremos una excepción del tipo IllegalThreadStateException.

VÍDEO

Cuando estamos trabajando con hilos, necesitaremos un bucle infinito para que el hilo se ejecute constantemente en segundo plano, pero, puede ocurrir que necesitemos que nuestros hilos ejecuten una acción cada cierto tiempo, lo cual es imposible, a no ser que consigamos detener el hilo de alguna forma

Java nos proporciona los siguientes métodos para suspender hilos de forma segura:

- Método **sleep()**: con este método, conseguiremos que nuestro hilo «se duerma» una cierta cantidad de milisegundos, que tendremos que pasarle como parámetro. En realidad, el hilo pasará al estado *bloqueado* durante esa cantidad tiempo, volviendo a ejecutarse después. En caso de que necesitemos reactivar el hilo antes de que se cumpla el tiempo de bloqueo, podemos hacerlo mediante el método **interrupt()**.

- Método **wait()**: Mediante este método, conseguiremos que el hilo se quede bloqueado hasta nuevo aviso. Para desbloquear el hilo, debemos utilizar los métodos **notify()** o **notifyAll()**. La diferencia entre estos dos métodos es que notifiyAll() notifica a todos y cada uno de los hilos que estén bloqueados. Al método notify() también podremos pasarle como parámetro una cantidad de milisegundos para que el hilo esté bloqueado, desbloqueándose cuando se cumpla esa cantidad de tiempo o cuando le llegue una señal notify, lo que antes ocurra.

Otra operación que podemos realizar es la de finalizar un hilo (que en condiciones normales finalizaría al terminar su tarea) directamente por nosotros mismos.

Para esto, podemos utilizar los siguientes métodos:

- Método **isAlive()**: Este método nos indicará si un hilo está vivo o no, es decir, nos devolverá ‘verdadero’ en caso de que el hilo esté en un estado diferente al de finalizado y ‘falso’ si está finalizado.

- Método **stop()**: Este método permite finalizar un hilo, pero es extremadamente peligroso utilizarlo, ya que puede dar situaciones de interbloqueo de hilos y hacer que nuestra aplicación se bloquee sin posibilidad de arreglo.

VÍDEO

MEDAC

Instituto Oficial de Formación Profesional

- Muchas gracias -