Programación multihilo

Programación de servicios y procesos

Objetivos:

Tras aprender los conceptos básicos de la programación concurrente y ver cómo los procesos pueden colaborar para conseguir multitarea, en este tema vamos a poner la mirada dentro de un proceso.

- Conocer las técnicas básicas para desarrollar aplicaciones multihilo en Java
- Compartir información entre los hilos de un proceso
- Aprender acerca de los problemas de acceso a memoria compartida
- Usar diferentes técnicas de programación para sincronizar la ejecución de los threads

Contenidos:

1) Introducción
2) Hilos: Estados de un hilo. Cambios de estado.
3) Recursos compartidos por Hilos
4) Hilos de usuario vs. hilos de sistema. Modelos de hilos.
5) Hilo principal de un programa.
6) Elementos relacionados con la programación de hilos. Librerías y clases.
7) Gestión de hilos: Creación y ejecución.
    - Suspensión de hilos

Introducción.

En muchas ocasiones las sentencias que forman parte un programa deben ejecutarse de manera secuencial, ya que, entre todas. forman una lista ordenada de pasos a seguir para resolver un problema. Otras veces dichos pasos no tienen porqué ser secuenciales, pudiéndose realizar varios de a la vez.

Otras veces disponer de simultaneidad en la ejecución no es opcional, sino necesario como ocurre en la mayoría de aplicaciones web.

En ambos casos la solución se encuentra en la programación multihilo, una técnica utilizada para conseguir procesamiento simultáneo.

Aunque plena de posibilidades, esta técnica no está exenta de condiciones y restricciones. Veremos la herramientas disponibles en Java tanto para el procesamiento multihilo así como para evitar y prevenir los conflictos que surgen como consecuencia de la programación concurrente.

Concepto de hilo

Hilo (thread) o subproceso:

- Son utilizados por la mayoría de los SO modernos.
- Unidad básica de utilización de la CPU.
- Secuencia de código que está en ejecución, dentro del contexto de un proceso.
- Tienen un identificador único.

Características:

- Un hilo NO puede existir sin un proceso, es decir, se ejecutan dentro de su contexto.
- Dependen de un proceso para ejecutarse, por lo tanto, no se pueden ejecutar por si solos.
- Los procesos tienen/gestionan su propia memoria, sin embargo, los hilos comparten la memoria del proceso.
- Ejecutar múltiples hilos, a la vez, nos permitirá realizar acciones diferentes.

Figura 2.1. *En los programas de un único hilo, la ejecución de las sentencias es secuencial.*

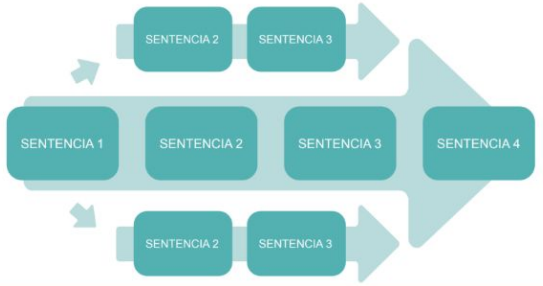


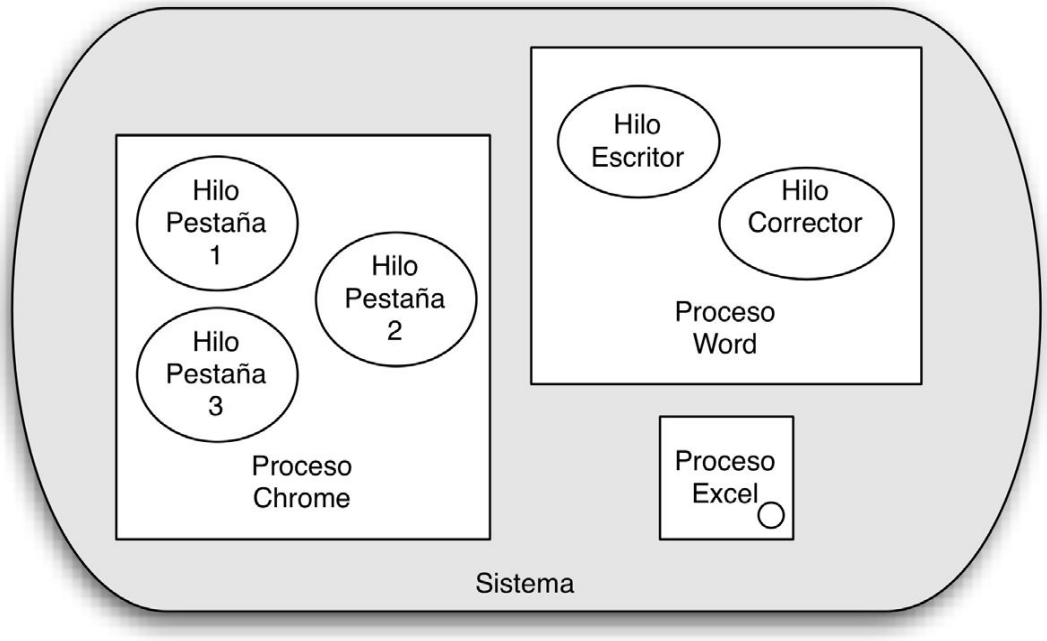
Figura 2.2. *En los programas de múltiples hilos, algunas sentencias se ejecutan de manera concurrente.*

Concepto de hilo

A continuación, se citan algunas de las características que tienen los hilos:

- **Dependencia del proceso:** No se pueden ejecutar independientemente. Siempre se tienen que ejecutar dentro del contexto de un proceso.
- **Ligereza:** Al ejecutarse dentro del contexto de un proceso, no requiere generar procesos nuevos, por lo que son óptimos desde el punto de vista del uso de recursos. Se pueden generar gran cantidad de hilos sin que provoquen pérdidas de memoria.
- **Compartición de recursos:** Dentro del mismo proceso, los hilos comparten espacio de memoria. Esto implica que pueden sufrir colisiones en los accesos a las variables provocando errores de concurrencia.
- **Paralelismo:** Aprovechan los núcleos del procesador generando un paralelismo real, siempre dentro de las capacidades del procesador.
- **Concurrencia:** Permiten atender de manera concurrente múltiples peticiones. Esto es especialmente importante en servidores web y de bases de datos, por ejemplo.

Concepto de hilo



Administrador de tareas

Archivo Opciones Vista

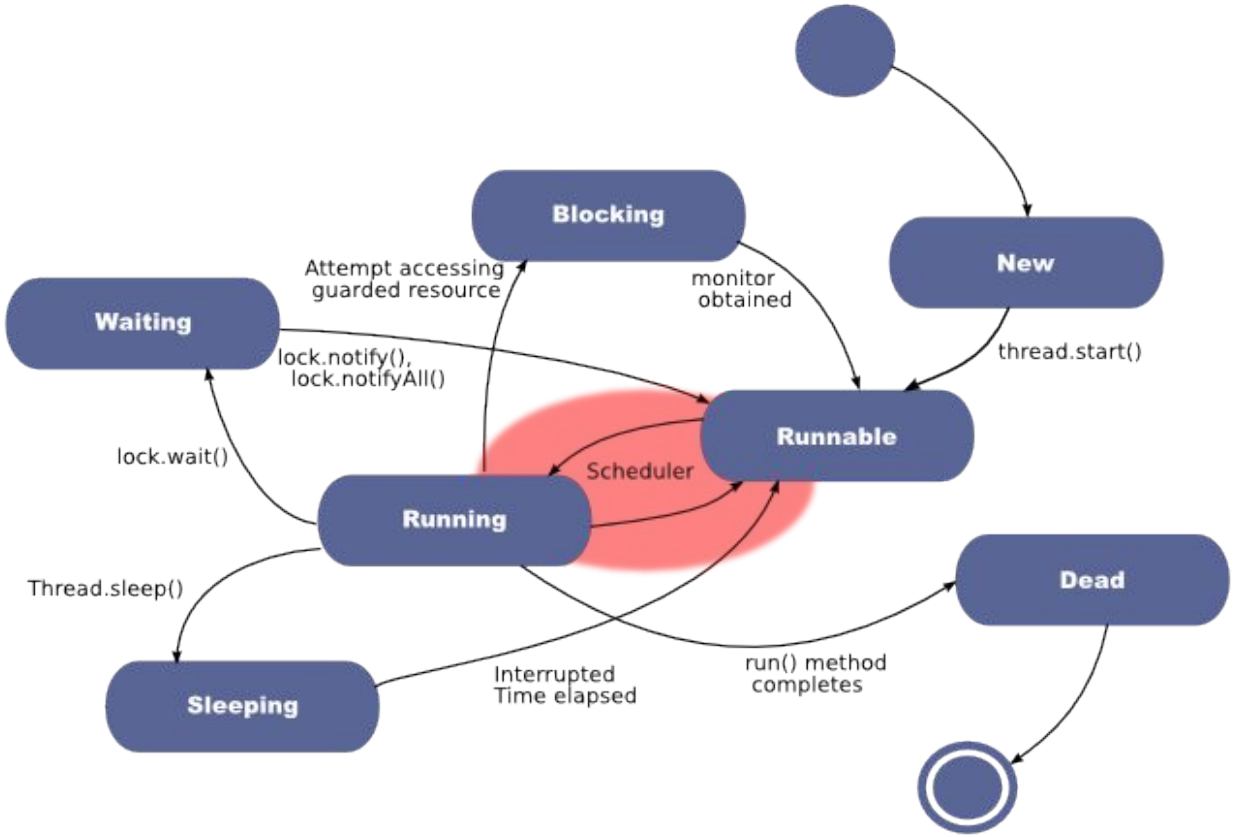
Procesos | Rendimiento | Historial de aplicaciones | Inicio | Usuarios | Detalles | Servicios

Nombre	Estado	5% CPU	57% Memoria	0% Disco	0% Red	2% GPU	Motor de GPU
Explorador de Windows		0,2%	75,1 MB	0 MB/s	0 Mbps	0%	
Google Chrome (14)		0,4%	1.295,6 MB	0 MB/s	0 Mbps	0%	GPU 0 - 3D
Google Chrome		0,1%	494,1 MB	0 MB/s	0 Mbps	0%	
Google Chrome		0%	5,6 MB	0 MB/s	0 Mbps	0%	
Google Chrome		0%	22,1 MB	0 MB/s	0 Mbps	0%	
Google Chrome		0%	3,6 MB	0 MB/s	0 Mbps	0%	
Google Chrome		0%	30,9 MB	0 MB/s	0 Mbps	0%	
Google Chrome		0%	197,1 MB	0 MB/s	0 Mbps	0%	
Google Chrome		0,1%	216,1 MB	0 MB/s	0 Mbps	0%	
Google Chrome		0%	42,9 MB	0 MB/s	0 Mbps	0%	
Google Chrome		0%	50,9 MB	0 MB/s	0 Mbps	0%	
Google Chrome		0%	14,3 MB	0 MB/s	0 Mbps	0%	
Google Chrome		0%	7,6 MB	0 MB/s	0 Mbps	0%	
Google Chrome		0,3%	205,4 MB	0 MB/s	0 Mbps	0%	
McAfee WSS Shared Browser ...		0%	4,3 MB	0 MB/s	0 Mbps	0%	
Procesador de comandos de ...		0%	0,7 MB	0 MB/s	0 Mbps	0%	
Herramienta Recortes		0%	15,3 MB	0 MB/s	0 Mbps	0%	

Estados de un hilo

- Los hilos pueden cambiar de estado a lo largo de su ejecución.

- Se definen los siguientes estados:
  - **Nuevo**: El hilo está preparado para su ejecución pero todavía no se ha realizado la llamada correspondiente en la ejecución del código del programa. Estado tras reservar su memoria en el sistema, **new**.
  - **Listo**: El proceso no se encuentra en ejecución aunque está preparado para hacerlo, está esperando a ser llamado *start()*.
  - **Runnable**: El hilo está preparado para ejecutarse y puede estar ejecutándose o no, está a la espera de ser asignado a la CPU.



Estados de un hilo

- Los hilos pueden cambiar de estado a lo largo de su ejecución.

- Se definen los siguientes estados:
  - **Bloqueado**: El hilo está bloqueado por diversos motivos:
    - Está dormido, método sleep().
    - Está esperando que finalice una operación de entrada/salida.
    - Ha llamado al método wait() y está esperando a ser liberado mediante un notify(), notifyAll().
    - Se ha llamado al método *suspend()* y está a la espera de un *resume()*.
  - **Terminado**: El hilo ha finalizado su ejecución, puede ser de forma normal (llegó al final del método *run()*) o no (método *stop()*, excepción,…).

Estados de un hilo

Tabla 2.1. Estados de los hilos y su correspondencia en Java

Estado	Valor en Thread.State	Descripción
Nuevo	NEW	El hilo está creado, pero aún no se ha arrancado.
Ejecutable	RUNNABLE	El hilo está arrancado y podría estar en ejecución o pendiente de ejecución.
Bloqueado	BLOCKED	Bloqueado por un monitor.
Esperando	WAITING	El hilo está esperando a que otro hilo realice una acción determinada.
Esperando un tiempo	TIME_WAITING	El hilo está esperando a que otro hilo realice una acción determinada en un período de tiempo concreto.
Finalizado	TERMINATED	El hilo ha terminado su ejecución.

El método getState() de la clase Thread, lo veremos más adelante, devuelve el estado de un hilo.

Programación de aplicaciones multihilo.

La programación **multihilo** permite la ejecución de varios hilos al mismo tiempo, tantos hilos como núcleos tenga el procesador, para realizar una tarea común.

A la hora de realizar un programa multihilo cooperativo, se deben seguir las fases:

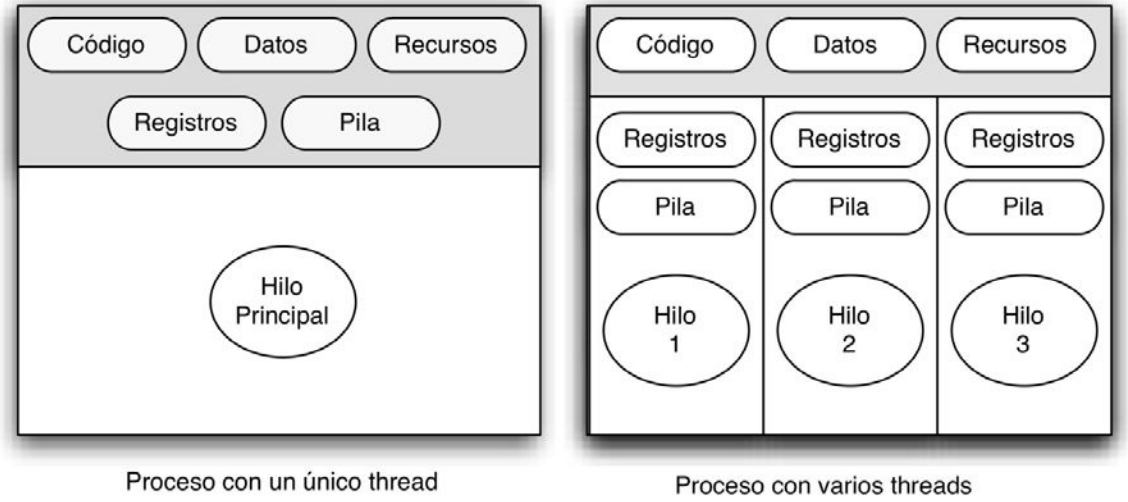
- **Descomposición funcional**: Es necesario identificar previamente las diferentes tareas que debe realizar la aplicación y las relaciones existentes entre ellas.
- **Partición**: La comunicación entre hilos se realiza principalmente a través de la memoria. Los tiempos de acceso son muy rápidos por lo que no supone una pérdida de tiempo significativa. Ojo: Problemas de sincronización.
- **Implementación**: Se utiliza la clase Thread o la interfaz Runnable como punto de partida. Utilizar mecanismos de sincronización.

Recursos compartidos por un hilo.

Los procesos mantienen su propio espacio de direcciones y recursos de ejecución mientras que los hilos dependen del proceso. Los hilos:

- **Comparten entre ellos:**
  - ✔ la sección de código, sección de datos,…
  - ✔ Datos (ejemplo, variables globales, **static**).
  - ✔ Recursos del sistema, por ejemplo, ficheros.

- Cada hilo **tiene asociado**:
  - ✔ Identificador único.
  - ✔ Contador de programa.
  - ✔ Conjunto de registros de la CPU.
  - ✔ Pila de variables.



- **Ventajas** de su uso:
  - ✔ Consumen menos recursos que un proceso, tanto en su ejecución como en su lanzamiento.
  - ✔ Menor tiempo de creación y terminación que un proceso.
  - ✔ Conmutación más rápida que en los procesos.

Beneficios

El uso de hilos repercutirá en:

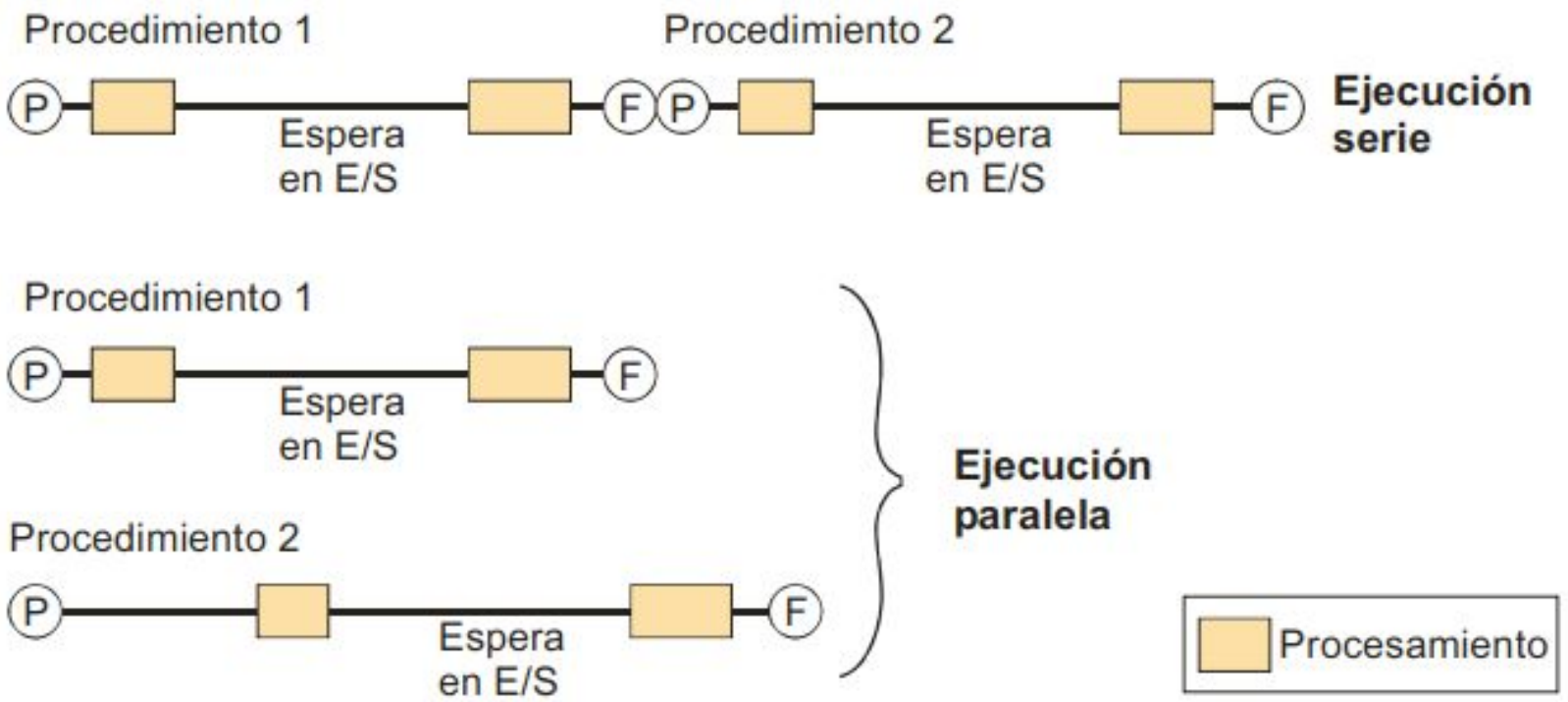
- **Capacidad de respuesta:** Los hilos permiten a los procesos continuar atendiendo peticiones del usuario aunque alguna de las tareas (hilo) que esté realizando el programa sea muy larga.

- **Compartición de recursos**: Por defecto, los threads comparten la memoria y todos los recursos del proceso al que pertenecen.
  - La creación de nuevos hilos no supone ninguna reserva adicional de memoria por parte del sistema operativo.

- **Paralelismo real**: La utilización de *threads* permite aprovechar la existencia de más de un núcleo en el sistema en arquitecturas *multicore*.

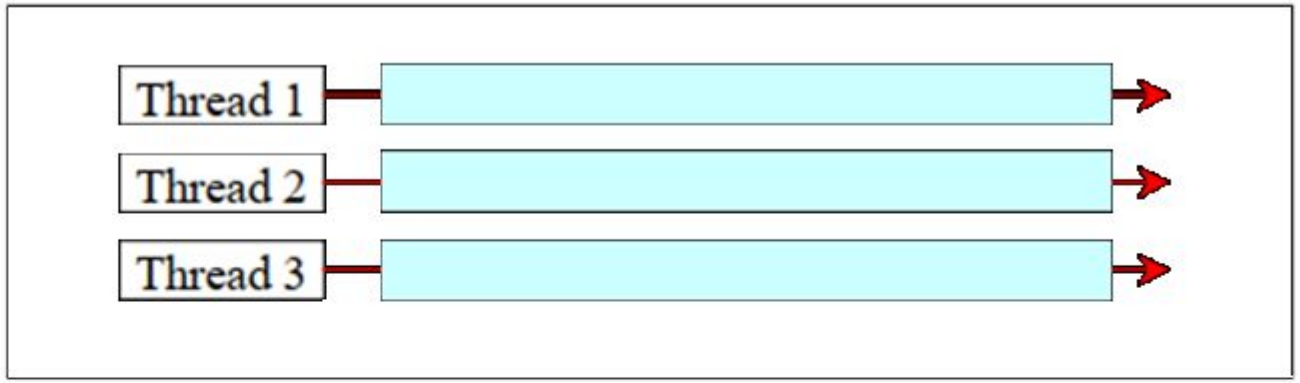
- **Prioridad de ejecución de las tareas**: Los threads permiten establecer prioridad de ejecución para manejar tareas en tiempo crítico.

Paralelización de un aplicación

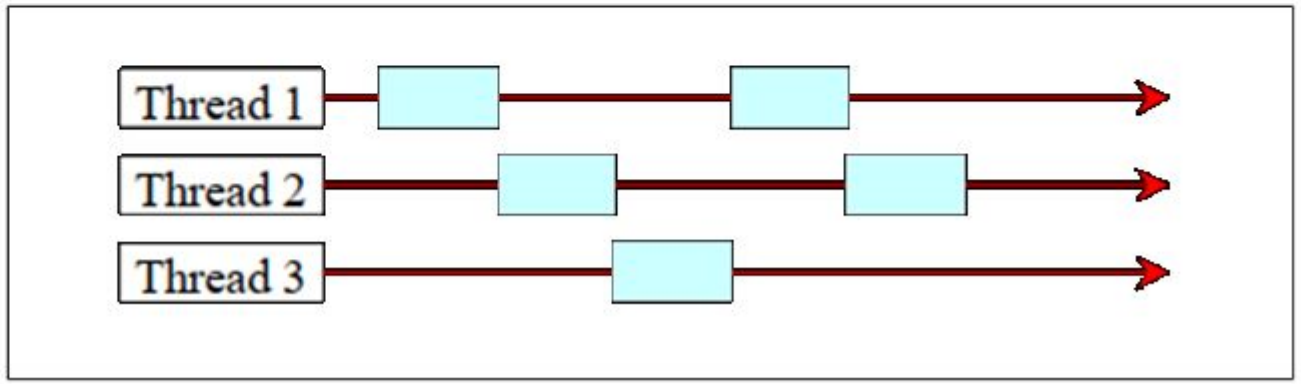


Paralelización de un aplicación

Threads múltiples en múltiples CPUs



Threads múltiples compartiendo una CPU



Hilos de usuario vs. hilos de sistema.

Hasta ahora hemos hablado de los hilos en sentido genérico, pero a nivel práctico los hilos pueden ser implementados a nivel de usuario o a nivel de kernel.

- **Hilos a nivel de usuario:** son implementados en alguna librería. Estos hilos se gestionan sin soporte del SO, el cual solo reconoce un hilo de ejecución.

- **Hilos a nivel de kernel:** el SO es quien crea, planifica y gestiona los hilos. Se reconocen tantos hilos como se hayan creado.

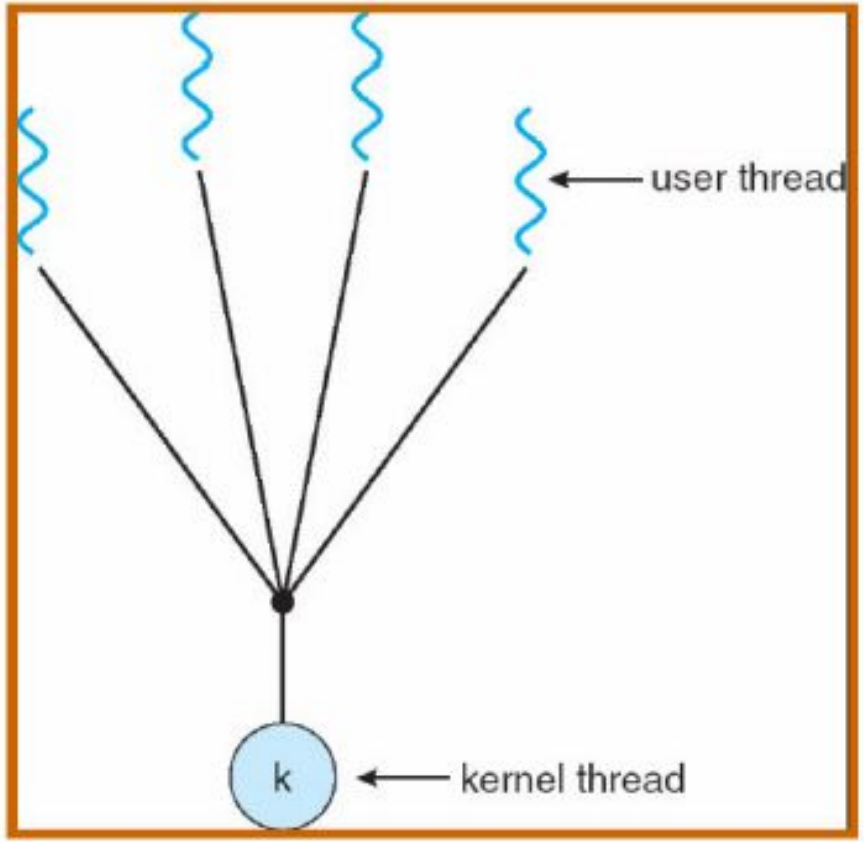
Los hilos a nivel de usuario tienen como beneficio que su cambio de contexto es más sencillo que el cambio de contexto entre hilos de kernel. Además, se pueden implementar aún si el SO no utiliza hilos a nivel de kernel. Otro de los beneficios consiste en poder planificar diferente a la estrategia del SO.

Los hilos a nivel de kernel tienen como gran beneficio poder aprovechar mejor las arquitecturas multiprocesadores, y que proporcionan un mejor tiempo de respuesta, ya que si un hilo se bloquea, los otros pueden seguir ejecutando.

Modelos de hilos.

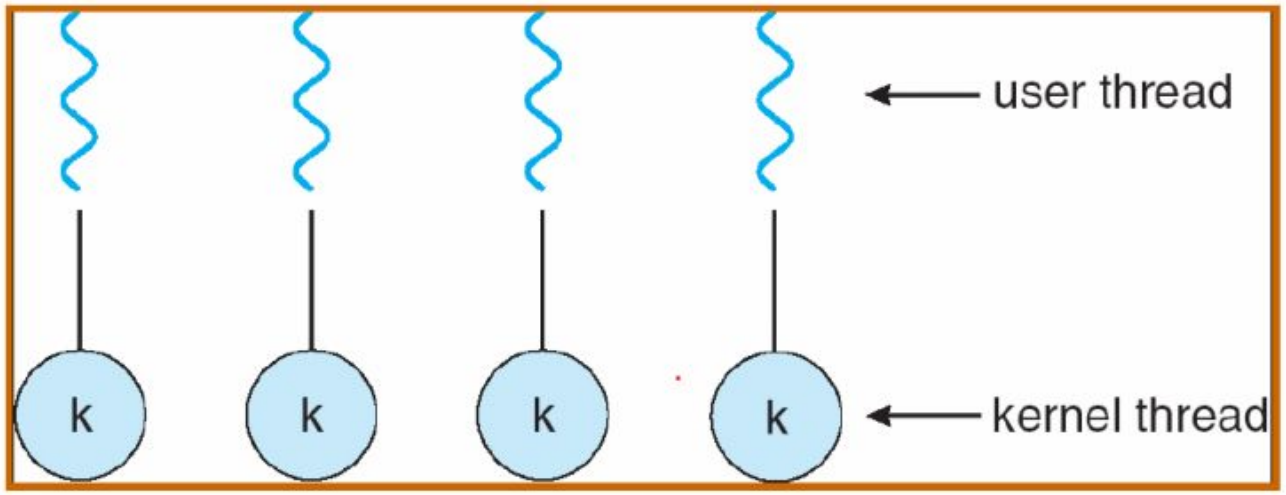
Existen 3 formas para establecer la relación

- **Modelo Mx1 (Many to one):**
  - El modelo asigna múltiples hilos de usuario a un hilo del kernel.
  - Este caso se corresponde a los hilos implementados a nivel de usuario, ya que el sistema solo reconoce un hilo de control para el proceso.
  - Tiene como inconveniente que si un hilo se bloquea, todo el proceso se bloquea.
  - También, dado que solo un hilo puede acceder al kernel cada vez, no podrán ejecutarse varios hilos en paralelo en múltiples CPUs



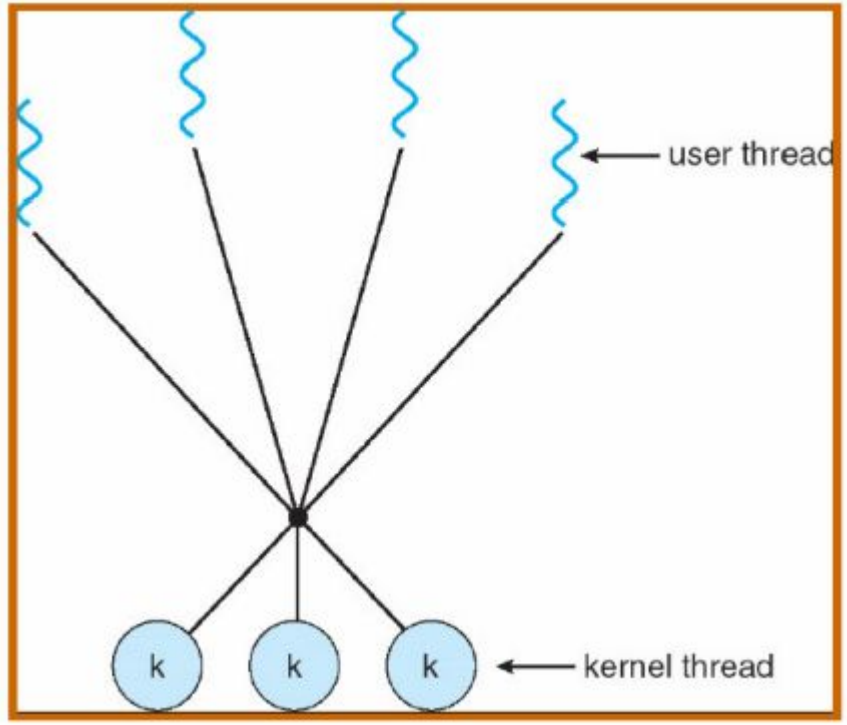
Modelos de hilos.

- **Modelo 1x1 (one to one):**
  - El modelo asigna cada hilo de usuario a un hilo del kernel.
  - Proporciona una mayor concurrencia que el modelo anterior, permitiendo que se ejecute otro hilo si uno se bloquea.
  - Tiene como inconveniente que cada vez que se crea un hilo a nivel de usuario, se crea un hilo a nivel del kernel, y la cantidad de hilos a nivel del kernel están restringidos en la mayoría de los sistemas.



Modelos de hilos.

- **Modelo MxN (many to many):**
  - El modelo multiplexa muchos hilos de usuario sobre un número menor o igual de hilos del kernel.
  - Cada proceso tiene asignado un conjunto de hilos de kernel, independientemente de la cantidad de hilos de usuario que haya creado.
  - No posee ninguno de los inconvenientes de los dos modelos anteriores, ya que saca lo mejor de cada uno.
  - El usuario puede crear tantos hilos como necesite y los hilos de kernel pueden ejecutar en paralelo.
  - Asimismo, cuando un hilo se bloquea, el kernel puede planificar otro hilo para su ejecución.
  - Entonces, el planificador a nivel de usuario asigna los hilos de usuario a los hilos de kernel, y el planificador a nivel de kernel asigna los hilos de kernel a los procesadores.



Hilo principal de un programa.

De cara a la gestión de los hilos, nos encontramos con dos operaciones elementales:

- **Creación y arranque de hilos**: Cualquier programa a ejecutarse es un proceso que tiene un hilo de ejecución principal. Este hilo puede a su vez crear nuevos hilos que ejecutarán código diferente o tareas, es decir el camino de ejecución no tiene por qué ser el mismo.

- **Espera de hilos**: Como varios hilos comparten el mismo procesador, si alguno no tiene trabajo que hacer, es bueno suspender su ejecución para que haya un mayor tiempo de procesador disponible.

Librerías y clases.

- **Creación de hilos**:
  I. Se realizará mediante la ***implementación de la interfaz Runnable (dentro del paquete java.lang)***, la cual proporciona la capacidad de añadir la funcionalidad de un hilo a una clase simplemente implementando dicha interfaz. Esta interfaz, debería ser utilizada si la clase solamente va a utilizar la funcionalidad run de los hilos.
  II. Extendiendo de la ***clase Thread (también dentro del paquete java.lang)*** mediante la creación de una subclase. La clase Thread es responsable de producir hilos para otras clases e implementa la interfaz Runnable

- **Arranque de hilos**:
  - El método ***run()*** implementa la operación create conteniendo el código a ejecutar por el hilo. Dicho método contendrá el hilo de ejecución.
  - La clase Thread define también el método ***start()*** para implementar la operación create. Este método es el que comienza la ejecución del hilo de la clase correspondiente.

Librerías y clases. Ejemplo “Ratón”

El siguiente ejemplo programado en Java ilustra cómo se comporta un programa que se ejecuta en un único hilo, así como las consecuencias que esto implica. El programa está compuesto por una única clase (representa un ratón) compuesta por dos atributos: el nombre y el tiempo en segundos que tarda en comer.

El método comer simula el tiempo en que tarda en comer un ratón. El método sleep está en la clase thread y “duerme” elñ proceso el número de milisegundos que le indiques. Lo utilizaremos mucho para simular proceso de negocio simulando el tiempo que tardan en ejecutarse.

En el método main se instancian varios objetos (ratones) y se invoca al método comer de cada uno de ellos. Este método muestra un texto por pantalla cuando comienza, realiza una pausa de la duración en segundos (con el método sleep de la clase Thread) que indica el parámetro tiempoAlimentacion y, finalmente, muestra otro texto por pantalla cuando finaliza.

Librerías y clases. Ejemplo “Ratón”

- **Programación secuencial (0_Ejemplo_Raton.java)**
  I. Cada ratón comienza a alimentarse, emplea la cantidad de tiempo indicada en la construcción del objeto y termina de comer.
  II. Este tipo de ejecución se conoce como secuencial o no concurrente: cada sentencia debe esperar a que se ejecute la sentencia anterior. Pese a que en el ejemplo no hay ninguna dependencia entre los procesos de alimentación de cada ratón, se están produciendo por cómo se ha programado la aplicación. Como consecuencia, cada ratón tardará en comenzar a comer tanto tiempo como tarden los ratones que comenzaron antes que él.
  III. El tiempo total del proceso es, por lo tanto, la suma de todos los tiempos parciales (en este caso, 18 segundos). Es fácil imaginar qué ocurriría si una tienda de comercio online funcionase de manera secuencial cuando varios miles de clientes acceden para comprar sus productos.

En sí, la ejecución secuencial no es necesariamente un problema. Muchos programas funcionan de manera secuencial bien porque no es posible una ejecución concurrente, o bien porque no se ha considerado oportuno que así sea. Sea como sea, los programas de ejecución secuencial no aprovechan los recursos del sistema para reducir los tiempos de proceso.

Librerías y clases. Ejemplo “Ratón”

El ejemplo anterior se puede programar de manera concurrente de forma sencilla, ya que no hay ningún recurso compartido. Para ello, basta con convertir cada objeto de la clase Ratón en un hilo (thread) y **programar aquello que se quiere que ocurra concurrentemente dentro del método run**. Una vez creadas las instancias, se invoca al método start de cada una de ellas, lo que provoca la ejecución del contenido del método run en un hilo independiente.

- **Programación concurrente o multihilo ((0_Ejemplo_Raton_Hilos.java)**
  I. Todos los ratones han comenzado a alimentarse de inmediato, sin esperar a que termine ninguno de los demás.
  II. El tiempo total del proceso será, aproximadamente, el tiempo del proceso más lento (en este caso, 6 segundos). La reducción del tiempo total de ejecución es evidente.

Sea lo que sea lo que el thread tenga que hacer, debe estar incluido en la implementación del método run tanto si usamos la interfez Runnable como si extendemos de Thread.

Librerías y clases. Ejemplo “Ratón”

Error común: Llamar a run () en vez de a start()

Cuando empezamos a trabajar con hilos, un error muy común es llamar directamente al método run en vez de llamar al método start():

```
Thread newThread = new Thread(MyRunnable()); newThread.run(); //should be start();
```

or

```
MyRunnable runnable = new MyRunnable(); runnable.run();
```

En principio no notamos ningún error ya que el código de run() se ejecuta y podemos ver los resultados. Sin embargo, ese código **no es ejecutado por el nuevo thread** que acabamos de crear. El método run() es ejecutado por el thread que ha creado el objeto, es decir, el mismo thread que ha ejecutado las líneas anteriores a la llamada a run().

Para hacer que el método run, de una instancia que implemente Runnable o de una que herede de Thread, sea ejecutado por un el nuevo thread que acabamos de crear, newThread, debemos llamar al método newThread.start().

Librerías y clases. Clase Thread

MÉTODO	QUÉ HACE
int getPriority()	Devuelve prioridad del hilo
setPriority(int p)	Cambia prioridad del hilo al entero p
void interrupt()	Interrumpe ejecución del hilo
boolean interrupted()	Comprueba si hilo ha sido interrumpido
Thread currentThread()	Devuelve referencia al hilo que se está ejecutando
boolean isDaemon()	Comprueba si hilo es Daemon (prioridad baja ejecuta en segundo plano)
setDaemon(boolean on)	Establece hilo como daemon, con true o false (pasa a hijo de usuario)

Librerías y clases. Clase Thread

MÉTODO	QUÉ HACE
start()	Comienza ejecución del hilo, la máquina virtual llama a run()
boolean isAlive()	Comprueba si hilo sigue vivo
sleep(long mils)	Hilo en ejecución pasa a bloqueado durante milisegundos indicados
run()	Construye hilo, llamado por start()Si devuelve el control, hilo se detieneÚnico método de interfaz Runnable
String toString	Devuelve hilo representado en cadena, con nombre, prioridad y grupo de hilos
long getId()	Devuelve identificador del hilo
void yield	Detiene hilo en ejecución temporalmente para dar paso a otros hilos
getName(String name)	Cambia nombre al hilo por el que se le pasa

También hay un setName

Ejercicio

Crea dos clases (hilos) Java que extiendan la clase Thread.

Uno de los hilos debe visualizar por pantalla en un bucle infinito la palabra TIC y el otro hilo la palabra TAC.

Dentro del bucle utiliza el método sleep() para que nos dé tiempo a ver las palabras que se imprimen cuando lo ejecutamos (tendrás que añadir un bloque try-catch para capturar la excepción InterruptedException).

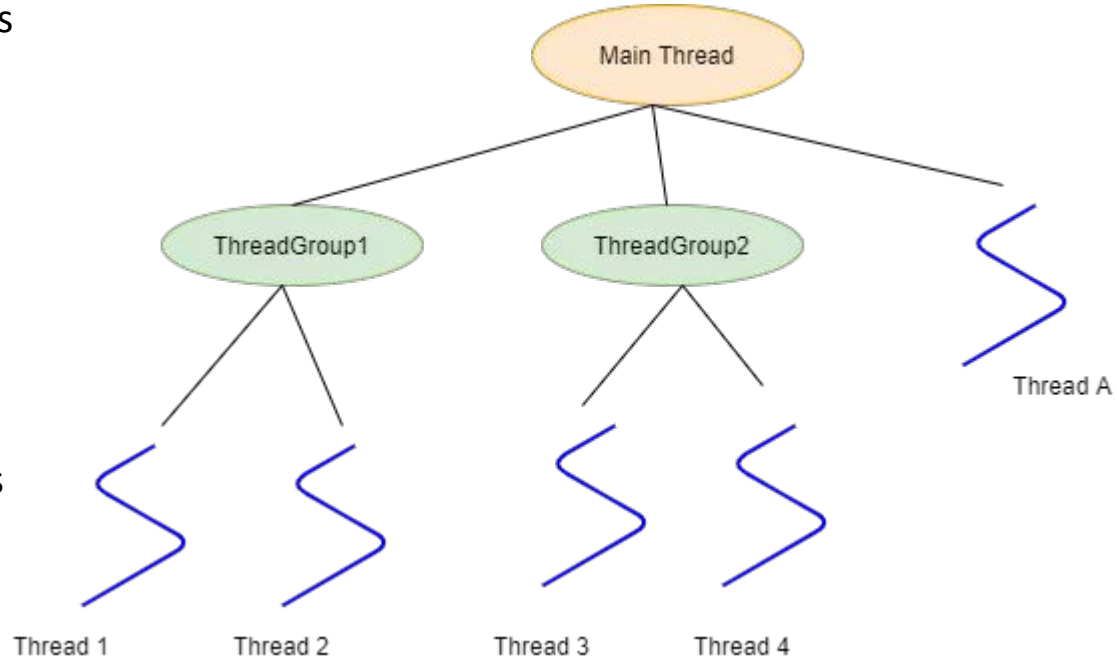
Crea después la clase Main que haga uso de los hilos anteriores. ¿Se visualizan los textos de forma ordenada (es decir TIC TAC TIC TAC TIC TAC…)?

Librerías y clases.

Todo hilo de ejecución en Java debe formar parte de un grupo. Por defecto, si no se especifica ningún grupo en el constructor, los hilos serán miembros del grupo main, que es creado por el sistema cuando arranca la aplicación Java.

La clase **ThreadGroup** se utiliza para manejar grupos de hilos en las aplicaciones Java. La clase Thread proporciona constructores en los que se puede especificar el grupo del hilo que se está creando en el mismo momento de instanciarlo.

La principal ventaja de un grupo de subprocesos es que podemos realizar operaciones como suspender, reanudar o interrumpir para todos los subprocesos utilizando una única llamada de función. Cada subproceso en un grupo de subprocesos tiene un subproceso principal excepto el subproceso inicial y, por lo tanto, representa una estructura de árbol.



Librerías y clases. Interfaz Runnable

Para añadir la funcionalidad de hilo a una clase que deriva de otra clase, siendo esta distinta de Thread, se utiliza la interfaz Runnable.

Esta interfaz añade la funcionalidad de hilo a una clase con solo implementarla. La interfaz Runnable proporciona un único método, el método run(). Este es ejecutado por el objeto hilo asociado. La forma general de declarar un hilo implementando la interfaz Runnable podemos verla en el ejemplo 0_Ejemplo_Raton_Runnable.java.

Ejercicio

Haz el ejercicio del TIC TAC implementando la interfaz Runnable en lugar de la clase Thread.

Gestión de hilos. Creación y arranque de hilos

¿Cuál de las dos opciones debemos utilizar?

- Mediante la interfaz **Runnable**:
  - Es más general, es decir, el objeto puede ser una subclase de una clase distinta de Thread (recuerda la herencia en Java).
  - Solamente podrá utilizarse el run() , además de aquellas funcionalidad que incluya el programador.

- Mediante la extensión de la clase **Thread**:
  - Sencillez, ya que esta define una serie de métodos útiles para la administración de hilos.
  - Está limitada, esto se debe a que las clases creadas como hilos deben (herencia en Java) ser descendientes únicamente de dicha clase.

Gestión de hilos. Join.

Además existen otro tipo de operaciones que permiten gestionar el flujo de ejecución de los hilos:

- **Join**: La ejecución del hilo puede suspenderse esperando hasta que el hilo correspondiente por el que espera finalice su ejecución.

Lo vemos con el ejemplo “0_Ejemplo_Raton_Join.java”.