Unidad Didáctica 1:

INTRODUCCIÓN A LA PROGRAMACIÓN MULTIPROCESO Y A LA PROGRAMACIÓN CONCURRENTE

1. Conceptos básicos

1.1. Programación Multiproceso

a) Introducción: Aplicaciones, Ejecutables y Procesos.

A simple vista, parece que con los términos aplicación, ejecutable y proceso, nos estamos refiriendo a lo mismo.

Una aplicación es un tipo de programa informático, diseñado como herramienta para resolver de manera automática un problema específico del usuario.
Debemos darnos cuenta de que sobre el hardware del equipo, todo lo que se ejecuta son programas informáticos, que, ya sabemos, que se llama software. Con la definición de aplicación anterior, buscamos diferenciar las aplicaciones, de otro tipo de programas informáticos, como pueden ser: los sistemas operativos, las utilidades para el mantenimiento del sistema, o las herramientas para el desarrollo de software. Por lo tanto, son aplicaciones, aquellos programas que nos permiten editar una imagen, enviar un correo electrónico, navegar en Internet, editar un documento de texto, chatear, etc.
Recordemos, que un programa es el conjunto de instrucciones que ejecutadas en un ordenador realizarán una tarea o ayudarán al usuario a realizarla.
Nosotros, como programadores y programadoras, creamos un programa, escribiendo su código fuente; con ayuda de un compilador, obtenemos su código binario o interpretado. Este código binario o interpretado, lo guardamos en un fichero.
Este fichero, es un fichero ejecutable, llamado comúnmente: ejecutable o binario.

Un ejecutable es un fichero que contiene el código binario o interpretado que será ejecutado en un ordenador.
De forma sencilla, un proceso, es un programa en ejecución. Pero, es más que eso, un proceso en el sistema operativo (SO), es una unidad de trabajo completa; y, el SO gestiona los distintos procesos que se encuentren en ejecución en el equipo. Lo más importante, es que

diferenciemos que un ejecutable es un fichero y un proceso es una entidad activa, el contenido del ejecutable, ejecutándose.

Un proceso es un programa en ejecución.
Un proceso existe mientras que se esté ejecutando una aplicación. Es más, la ejecución de una aplicación, puede implicar que se arranquen varios procesos en nuestro equipo; y puede estar formada por varios ejecutables y librerías.
Al instalar una aplicación en el equipo, podremos ver que puede estar formada por varios ejecutables y librerías. Siempre que lancemos la ejecución de una aplicación, se creará, al menos, un proceso nuevo en nuestro sistema.

1.2. Gestión de procesos.

Como sabemos, en nuestro equipo, se están ejecutando al mismo tiempo, muchos procesos. Por ejemplo, podemos estar escuchando música; al mismo tiempo, estamos programando ; tenemos el navegador web abierto; incluso, tenemos abierto un editor de texto.

Independientemente de que el microprocesador de nuestro equipo sea más o menos moderno (con uno o varios núcleos de procesamiento), lo que nos interesa es que actualmente, nuestros SO son multitarea; como son, por ejemplo, Windows y GNU/Linux.
Ser multitarea es, precisamente, permitir que varios procesos puedan ejecutarse al mismo tiempo, haciendo que todos ellos compartan el núcleo o núcleos del procesador.
Pero, ¿cómo?
Imaginemos que nuestro equipo, es como nosotros mismos cuando tenemos más de una tarea que realizar. Podemos, ir realizando cada tarea una detrás de otra, o, por el contrario, ir realizando un poco de cada tarea. Al final, tendremos realizadas todas las tareas, pero para otra persona que nos esté mirando desde fuera, le parecerá que, de la primera forma, vamos muy lentos (y más, si está esperando el resultado de una de las tareas que tenemos que realizar); sin embargo, de la segunda forma, le parecerá que estamos muy ocupados, pero que poco a poco estamos haciendo lo que nos ha pedido.
Pues bien, el micro, es nuestro cuerpo, y el SO es el encargado de decidir, por medio de la gestión de procesos, si lo hacemos todo de golpe, o una tarea detrás de otra.

a) Introducción.

En nuestros equipos ejecutamos distintas aplicaciones interactivas y por lotes.
Como sabemos, un microprocesador es capaz de ejecutar miles de millones de instrucciones básicas en un segundo (por ejemplo, un i7 puede llegar hasta los 3,4 GHz).
Un micro, a esa velocidad, es capaz de realizar muchas tareas, y nosotros, apreciaremos que solo está ejecutando la aplicación que nosotros estamos utilizando. Al fin y al cabo, al micro, lo único que le importa es ejecutar instrucciones y dar sus resultados, no tiene conocimiento de si pertenecen a uno u otro proceso, para él son instrucciones. Es, el SO el encargado de decidir qué proceso puede entrar a ejecutarse o debe esperar.

Los nuevos micros, con varios núcleos, pueden, casi totalmente, dedicar una CPU a la ejecución de uno de los procesos activos en el sistema. Pero no nos olvidemos de que además de estar activos los procesos de usuario, también se estará ejecutando el SO, por lo que seguirá siendo necesario repartir los distintos núcleos entre los procesos que estén en ejecución.

b) Estados de un Proceso.

Si el sistema tiene que repartir el uso del microprocesador entre los distintos procesos, ¿qué le sucede a un proceso cuando no se está ejecutando? Y, si un proceso está esperando datos, ¿por qué el equipo hace otras cosas mientras que un proceso queda a la espera de datos? Veamos con detenimiento, cómo es que el SO controla la ejecución de los procesos. Ya comentamos en el apartado anterior, que el SO es el encargado de la gestión de procesos. En el siguiente gráfico, podemos ver un esquema muy simple de cómo podemos planificar la ejecución de varios procesos en una CPU.

En este esquema, podemos ver:
1. Los procesos nuevos, entran en la cola de procesos activos en el sistema.
2. Los procesos van avanzando posiciones en la cola de procesos activos, hasta que les toca el turno para que el SO les conceda el uso de la CPU.
3. El SO concede el uso de la CPU, a cada proceso durante un tiempo determinado y equitativo, que llamaremos quantum. Un proceso que consume su quantum, es pausado y enviado al final de la cola.
4. Si un proceso finaliza, sale del sistema de gestión de procesos.

Esta planificación que hemos descrito, resulta equitativa para todos los procesos (todos van a ir teniendo su quamtum de ejecución). Pero se nos olvidan algunas situaciones y características de nuestros los procesos: Cuando un proceso, necesita datos de un archivo o una entrada de datos que deba suministrar el usuario; o, tiene que imprimir o grabar datos; cosa que llamamos 'el proceso está en una operación de entrada/salida' (E/S para abreviar). El proceso, queda bloqueado hasta que haya finalizado esa E/S. El proceso es bloqueado, porque, los dispositivos son mucho más lentos que la CPU, por lo que, mientras que uno de ellos está esperando una E/S, otros procesos pueden pasar a la CPU y ejecutar sus instrucciones.

Cuando termina la E/S que tenga un proceso bloqueado, el SO, volverá a pasar al proceso a la cola de procesos activos, para que recoja los datos y continúe con su tarea (dentro de sus correspondientes turnos).
Todo proceso en ejecución, tiene que estar cargado en la RAM física del equipo o memoria principal, así como todos los datos que necesite.
Hay procesos en el equipo cuya ejecución es crítica para el sistema, por lo que, no siempre pueden estar esperando a que les llegue su turno de ejecución, haciendo cola.

Por ejemplo, el propio SO es un programa, y por lo tanto un proceso o un conjunto de procesos en ejecución. Se le da prioridad, evidentemente, a los procesos del SO, frente a los procesos de usuario.
Con todo lo anterior, podemos quedarnos con los siguientes estados en el ciclo de vida de un proceso:

- **Nuevo**. Proceso nuevo, creado.
- **Listo**. Proceso que está esperando la CPU para ejecutar sus instrucciones.
- **En ejecución**. Proceso que actualmente, está en turno de ejecución en la CPU.

- **Bloqueado**. Proceso que está a la espera de que finalice una E/S.
- **Suspendido**. Proceso que se ha llevado a la memoria virtual para liberar, un poco la RAM del sistema.
- **Terminado**. Proceso que ha finalizado y ya no necesitará más la CPU.
  El siguiente gráfico, nos muestra las distintas transiciones que se producen entre uno u otro estado:

1.3. Servicios. Hilos.

La conclusión que estamos sacando, es, que todo lo que se ejecuta en un equipo es un programa y que, cuando está en ejecución, se llama proceso. Entones ¿qué son los servicios? ¿y los hilos?
En este apartado, haremos una breve introducción a los conceptos servicio e hilo.

El ejemplo más claro de hilo o thread, es un juego. El juego, es la aplicación y,mientras que nosotros controlamos uno de los personajes, los 'malos' también se mueven, interactúan por el escenario y quitan vida. Cada uno de los personajes del juego es controlado por un **hilo**. Todos los hilos forman parte de la misma aplicación, cada uno actúa siguiendo un patrón de comportamiento. El comportamiento es el algoritmo que cada uno de ellos seguirá. Sin embargo, todos esos hilos comparten la información de la aplicación: el número de vidas restantes, la puntuación obtenida hasta eso momento, la posición en la que se encuentra el personaje del usuario y el resto de personajes, si ha llegado el final del juego, etc. Como sabemos, esas informaciones son variables. Pues

bien, un proceso, no puede acceder directamente a la información de otro proceso. Pero, los hilos de un mismo proceso están dentro de él, por lo que comparten la información de las variables de ese proceso.

Realizar cambios de contexto entre hilos de un mismo proceso, es más rápido y menos costoso que el cambio de contexto entre procesos, ya que sólo hay que cambiar el valor del registro contador de programa de la CPU y no todos los valores de los registros de la CPU.
Un proceso, estará formado por, al menos, un hilo de ejecución.
Un proceso es una unidad pesada de ejecución. Si el proceso tiene varios hilos,cada hilo, es una unidad de ejecución ligera.

Un servicio es un proceso que, normalmente, es cargado durante el arranque del sistema operativo. Recibe el nombre de servicio, ya que es un proceso que queda a la espera de que otro le pida que realice una tarea. Por ejemplo, tenemos el servicio de impresión con su típica cola de trabajos a imprimir. El servicio de impresión, es el encargado de ir enviando los datos de forma correcta a la impresora para que el resultado sea el esperado. Además, las impresoras, no siempre tienen suficiente memoria para guardar todos los datos de impresión de un trabajo completo, por lo que el servicio de impresión se los dará conforme vaya necesitándolos.
Como este, hay muchos servicios activos o en ejecución en el sistema, y no todos son servicios del sistema operativo, también hay servicios de aplicación, instalados por el usuario y que pueden lanzarse al arrancar el sistema operativo o no, dependiendo de su configuración o cómo los configuremos.
Un servicio, es un proceso que queda a la espera de que otros le pida que realice una tarea.

2. Programación concurrente

2.1. Introducción

La idea de programación concurrente siempre ha estado asociada a los sistemas operativos: Un sólo procesador de gran capacidad debía repartir su tiempo entre muchos usuarios. Para cada usuario, la sensación era que el procesador estaba dedicado para él.

La programación de estos sistemas se hacía a bajo nivel (ensamblador). Posteriormente aparecerían lenguajes de alto nivel con soporte para este tipo de programación. Su utilización y potencial utilidad se apoya en: threads o hilos, java e internet.

La programación concurrente ha ido ganando interés y actualmente se utiliza muy a menudo en la implementación de numerosos sistemas.

Tres grandes hitos se nos antojan importantes para que la programación concurrente actualmente sea tan importante:
• La aparición del concepto de thread o hilo que hace que los programas puedan ejecutarse con mayor velocidad comparados con aquellos que utilizan el concepto de proceso.
• La aparición más reciente de lenguajes como Java, lenguaje orientado a objetos de propósito general que da soporte directamente a la programación concurrente mediante la inclusión de primitivas específica.
• La aparición de Internet que es un campo abonado para el desarrollo y la utilización de programas concurrentes. Cualquier programa de Internet en el que podamos pensar tales como un navegador, un chat, etc. están programados usando técnicas de programación concurrente.

Concepto de programación concurrente

Según el diccionario de la Real Academia Española, una de las acepciones de la palabra concurrencia es

"Acaecimiento o concurso de varios sucesos en un mismo tiempo".

Si en esta definición sustituimos la palabra suceso por proceso ya tenemos una primera aproximación a lo que va a ser la concurrencia en computación.

2.2. **Concurrencia**

Dos procesos serán concurrentes cuando la primera instrucción de uno de ellos se ejecuta después de la primera instrucción del otro y antes de la última.
Es decir, existe un solapamiento en la ejecución de sus instrucciones. No tienen por qué ejecutarse exactamente al mismo tiempo, simplemente es

suficiente con el hecho de que exista un intercalado entre la ejecución de sus instrucciones. Si se ejecutan al mismo tiempo los dos procesos, entonces tenemos una situación de programación paralela. La programación concurrente es un paralelismo potencial. Dependerá del hardware subyacente como veremos más adelante.

Dicho de otra forma, un programa, al ponerse en ejecución, puede dar lugar a más de un proceso, cada uno de ellos ejecutando una parte del programa.
Continuando con el miso ejemplo, el programa de navegador de Internet puede dar lugar a más de un proceso: uno que controla las acciones del usuario con la interfaz, otro que hace las peticiones al servidor, etc.

Cuando varios procesos se ejecutan concurrentemente puede haber procesos que colaboren para un determinado fin mientras que puede haber otros que compitan por los recursos del sistema. Incluso aquellos procesos que colaboran deberán competir a la hora de obtener tiempo de procesador.

Para llevar a cabo las tareas de colaboración y competencia por los recursos se hace necesaria la introducción de mecanismos de comunicación y sincronización entre procesos. Del estudio de estos mecanismos trata la programación concurrente.

2.3. Programación concurrente

Es la disciplina que se encarga del estudio de las notaciones que permiten especificar la ejecución concurrente de las acciones de un programa, así como las técnicas para resolver los problemas inherentes a la ejecución concurrente, que son básicamente comunicación y sincronización.

El principal problema de la programación concurrente corresponde a no saber en que orden se ejecutan los programas (en especial los programas que se comunican). Se debe tener especial cuidado en que este orden no afecte el resultado de los programas.

Como puede intuirse, el trabajar con procesos concurrentes va a añadir complejidad a la tarea de programar. Cabe entonces hacerse la pregunta de ¿cuáles son los beneficios que aporta la programación concurrente?

2.4. Beneficios de la programación concurrente

Existen diversos motivos por los que la programación concurrente es útil. Destacaremos aquí dos de ellos: velocidad de ejecución y solución de problemas de naturaleza concurrente y mejor aprovechamiento de la CPU.

a) Velocidad de ejecución

Cuando la puesta en ejecución de un programa conlleva la creación de varios procesos y el sistema consta de más de un procesador, existe la posibilidad de asignar un proceso a cada procesador de tal forma que el programa se ejecuta de una forma más rápida. Los programas de cálculo numérico son grandes beneficiados de este hecho.

2.5. Solución de problemas inherentemente concurrentes

Existen algunos problemas cuya solución es más fácil de abordar mediante el uso de programación concurrente pues su naturaleza es eminentemente concurrente.

a) Sistemas de control

Son aquellos sistemas en los que hay una captura de datos, normalmente a través de sensores, un análisis de esos datos y una posible actuación posterior en función del resultado del análisis. La recolección de datos se puede estar haciendo de diversas entidades físicas como por ejemplo edificios o estancias dentro de edificios. No sería tolerable un sistema secuencial que vaya capturando los datos uno a uno de las distintas estancias. Podría ocurrir que al llegar a capturar los datos de la última estancia, la primera ya haya sido pasto de las llamas. Tanto la captura de datos desde cada estancia como su tratamiento y posterior actuación son candidatos a ser procesos distintos y de naturaleza concurrente. Esto nos

garantiza que nuestro sistema de control pueda responder a las alarmas que se produzcan.

b) Tecnologías Web

La mayoría de los programas relacionados con la Web son concurrentes: los servidores Web que son capaces de atender concurrentemente múltiples conexiones de usuarios; los programas de chat que permiten mantener la conversación de varios usuarios; los servidores de correo que permiten que múltiples usuarios puedan mandar y recibir mensajes al mismo tiempo; los propios navegadores que permiten que un usuario pueda estar haciendo una descarga mientras navega por otras páginas, o se ejecuta un applet de Java, etc.

c) Aplicaciones basadas en interfaces de usuarios.

La concurrencia en este tipo de aplicaciones va a permitir que el usuario pueda interaccionar con la aplicación aunque ésta esté realizando alguna tarea que consume mucho tiempo de procesador. Un proceso controla la interfaz mientras otro hace la tarea que requiere un uso intensivo de la CPU. Esto facilitará que tareas largas puedan ser abortadas a mitad de ejecución.

d) Simulación

Los programas secuenciales encuentran problemas al simular sistemas en los que existen objetos físicos que tienen un comportamiento autónomo independiente. La programación concurrente permitirá modelar esos objetos físicos y ponerlos en ejecución de forma independiente y concurrente, sincronizándolos de la forma apropiada.

e)SGBD

En Sistemas Gestores de Bases de Datos la concurrencia juega un papel muy importante cuando se va a permitir a varios usuarios interactuar con el sistema. Cada usuario puede ser visto como un proceso. Obviamente hay que implementar la política adecuada para evitar situaciones en las

que dos usuarios modifican al mismo tiempo un registro. Sin embargo, a varios usuarios que quieran acceder a un mismo registro para consultarlo y no modificarlo, debe permitírseles un acceso concurrente.

2.6. Problemas inherentes a la programación concurrente.

A la hora de crear un programa concurrente podemos encontrarnos con dos problemas:
Exclusión mutua: es muy típico que varios procesos accedan a la vez a una variable compartida para actualizarla. Esto se debe evitar , ya que puede producir inconsistencia de datos: uno puede estar actualizando la variable a la vez que otro la puede estar leyendo. Por ello es necesario conseguir la exclusión mutua de los procesos respecto a la variable compartida. Como solución tenemos la llamada región critica. Cuando dos o mas procesos comparten una variable, el acceso a dicha variable debe efectuarse siempre dentro de la región critica asociada a la variable. Solo uno de los procesos podrá acceder para actualizarla y los demás deberán esperar, el tiempo de estancia es finito.
Condición de sincronización:Hace referencia a la necesidad de coordinar los procesos con el fin de sincronizar sus actividades. Puede ocurrir que un proceso P1 llegue a un estado X que no pueda continuar hasta que otro proceso P2 haya llegado a un estado Y de su ejecución. La programación concurrente proporciona mecanismos para bloquear procesos a la espera de que ocurra un evento y para desbloquearlos cuando esto ocurra.

Algunas herramientas para manejar la concurrencia son: la región critica, los semáforos, región critica condicional, monitores, etc..

2.7. Concurrencia y arquitecturas hardware.

Parece obvio pensar que si dos procesos van a ejecutarse de forma concurrente vamos a necesitar dos procesadores, uno para cada proceso. Sin embargo, esto no tiene por qué ser así. Dependerá, aunque no exclusivamente, del hardware disponible y su topología.

Cuando hablamos de hardware nos estamos refiriendo fundamentalmente al número de procesadores en el sistema. Así, se puede hacer una primera distinción entre aquellos sistemas donde sólo hay un procesador, sistema monoprocesador, y aquellos en los que hay más de un procesador,

sistemas multiprocesador. En ambos sistemas es posible tener concurrencia.

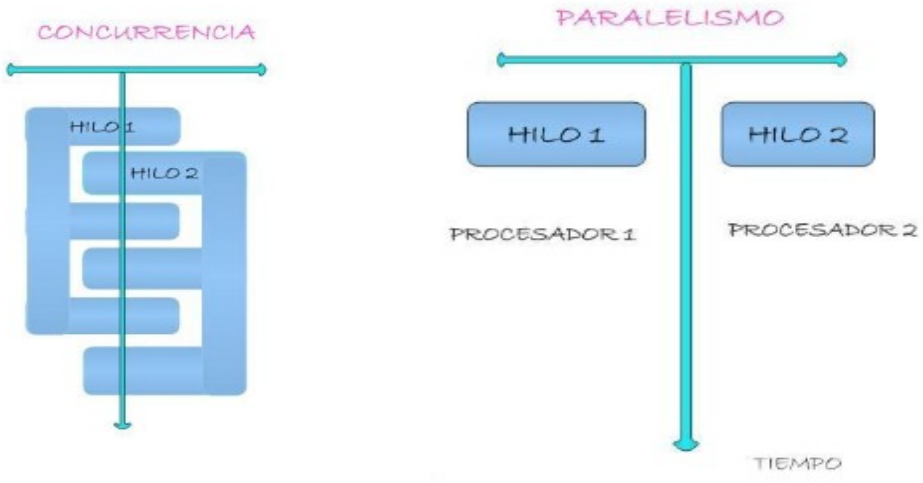
3. Diferencias entre los diferentes tipos de programación

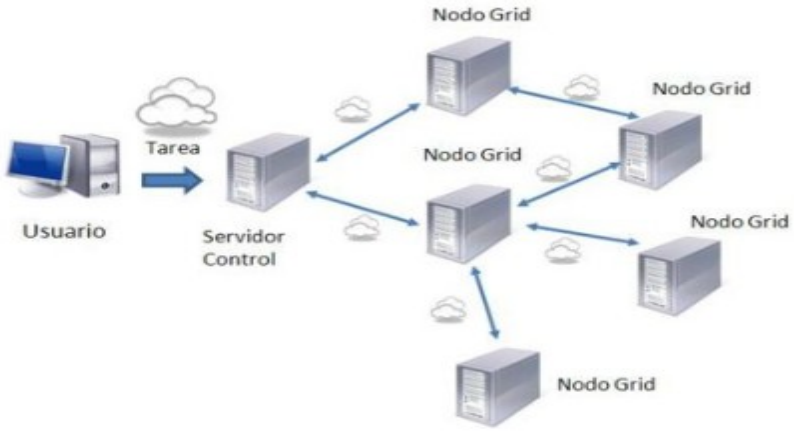
La programación concurrente define un conjunto de acciones que pueden ser ejecutadas simultáneamente.

La programación paralela es un tipo de programación concurrente diseñado para ejecutarse en un sistema multiprocesador.

La programación distribuida es una programación en paralelo que está diseñada para ejecutarse en un sistema distribuido, es decir, en una red de procesadores autónomos que no comparten una memoria común.

Nos ocuparemos de los programas concurrentes, no haciendo ningún tipo de suposición sobre el número de procesadores y su topología. Un programa concurrente debe funcionar independientemente de que lo ejecutemos en un sistema monoprocesador o en un sistema multiprocesador.





4. Aparición de los hilos

Habíamos definido un proceso como un programa en ejecución. Pero, ¿cómo se ejecuta?, ¿en que momento esta ejecutándose?, ¿ cuándo finaliza?. Todo proceso en un sistema operativo presenta un estado que indica la situación de la ejecución en que se encuentra. El número de posibles estados varía de un sistema operativo a otro.

El programador no necesita controlar esto, lo hace el sistema operativo. Sin embargo un programa multihilo mal hecho puede dar lugar problemas como los siguientes:

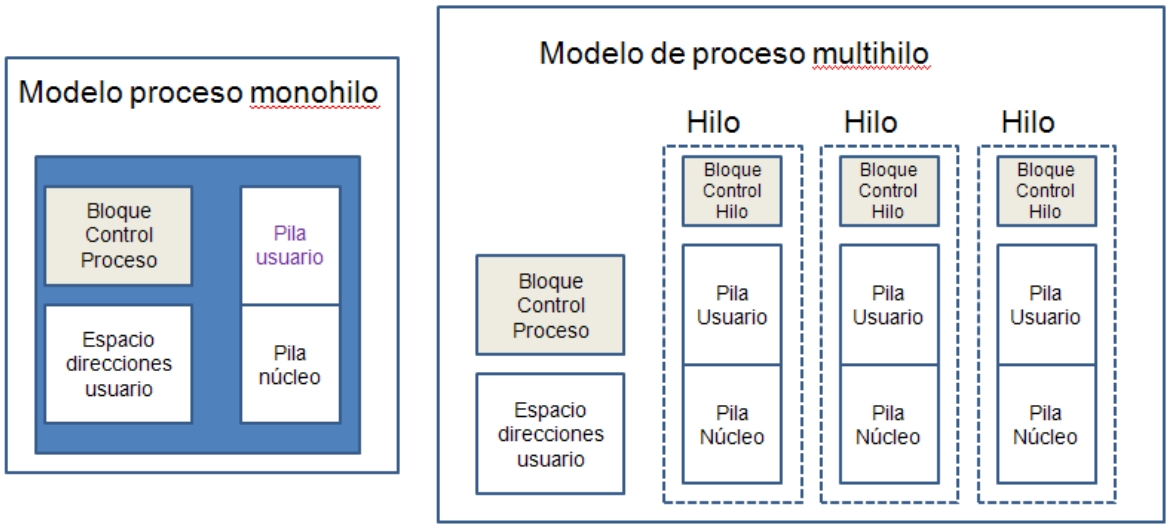
- Interbloqueo. Se produce cuando las peticiones y las esperas se entrelazan de forma que ningún proceso puede avanzar.
- Inanición. Ningún proceso consigue hacer ninguna tarea útil y por lo tanto hay que esperar a que el administrador del sistema detecte el interbloqueo y mate procesos (o hasta que alguien reinicie el equipo).

Cada proceso tiene sus propias características: código, variables,id, contadores, pila, etc. La gestión y el cambio de contexto de cada proceso es muy costoso. Se debe actualizar los registros de uso de memoria, y controlar los estados en los que quedan los procesos.

Se vio la necesidad de crear entidades mas ligeras en el espacio del usuario y cuyos cambio de contexto fuera mucho mas ligero.

Un thread(hilo de ejecución), en sistemas operativos es una característica que permite a una aplicación realizar varias tareas a la vez(concurrentemente). Los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos,etc. El hecho de que los hilos de ejecución de un mismo proceso compartan los recursos hace que cualquier hilo pueda modificarlos. Cuando un hilo modifica un dato en la memoria, los otros hilos acceden a ese dato modificado inmediatamente. Lo que es propio de cada hilo es el contador de programa , la pila de ejecución y el estado de la CPU. Esta técnica permite simplificar el diseño de una aplicación que debe llevar a cabo funciones simultáneamente.

Un hilo es básicamente una tarea que puede ser ejecutada en paralelo con otra tarea.



Un proceso sigue en ejecución mientras al menos uno de sus hilos de ejecución siga activo. Cuando el proceso finaliza, todos sus hilos de ejecución también han terminado. Asimismo en el momento en el que todos los hilos de ejecución finalizan, el proceso no existe mas y todos sus recursos son liberados.

Los threads/hilos pueden estar en dos niveles: a nivel usuario (p.ej. java) o a nivel del sistema operativo (hilos del sistema).

Los threads/hilos de sistema dan soporte a los threads/hilos de usuario mediante un API (Application Program Interface).

5. Hilos en Java

En Java, el proceso que siempre se ejecuta es el llamado main que es a partir del cual se inicia prácticamente todo el comportamiento de nuestra aplicación,y en ocasiones a la aplicación le basta con este solo proceso para funcionar de manera adecuada, sin embargo, existen algunas aplicaciones que requieren mas de un proceso (o hilo) ejecutandose al mismo tiempo(multithreading), por ejemplo se tiene una aplicación de una tienda de la cual se actualizan los precios y mercancías varias veces al día a través de la red, se verifican los nuevos descuentos y demás pero que a su vez es la encargada de registrar las compras y todos sus movimientos que se realice con la mercancía dentro de la tienda, si se decide que dicha aplicación trabajara de la manera simple y con un solo proceso(o hilo), el trabajo de actualización de precios y mercancías debe de finalizar antes de que alguien pueda hacer algún movimiento con un producto dentro de la tienda, o viceversa, ya que la aplicación no es capaz de mantener el proceso de actualización en segundo plano mientras registra el movimiento. En este caso el sistema monohilo representa inconvenientes por lo que debería tomarse un sistema multi-hilo para corregirlo.

5.1. Elementos relacionados con la programación de hilos

Ahora que ya entendemos lo que son los hilos pasaremos a una definición un poco mas especifica de Java. En Java un hilo o Thread puede ser dos cosas:

- Una instancia de la clase java.lang.Thread
- Un proceso en ejecución.

Java proporciona un API para el uso de hilos: clase Thread dentro del paquete java.lang.Thread. Es de gran utilidad tener un lenguaje de alto nivel para programar concurrentemente utilizando threads/hilos, de ahí el potencial y la fama de Java.
Cuando arranca un programa existe un hilo principal (main), y luego se pueden generar nuevos hilos que ejecutan código en objetos diferentes o el mismo.

5.2. Recursos compartidos por los hilos.

Un hilo lleva asociados los siguientes elementos:
Un identificador único.
Un contador de programa propio.
Un conjunto de registros.
Una pila (variables locales).
Por otra parte, un hilo puede compartir con otros hilos del mismo proceso los siguientes recursos:
Código.
Datos (como variables globales).
Otros recursos del sistema operativo, como los ficheros abiertos y las señales.

El hecho de que los hilos compartan recursos (por ejemplo, pudiendo acceder a las mismas variables) implica que sea necesario utilizar esquemas de bloqueo y sincronización, lo que puede hacer más difícil el desarrollo de los programas y así como su depuración. Realmente, es en la sincronización de hilos, donde reside el arte de programar con hilos; ya que de no hacerlo bien, podemos crear una aplicación totalmente ineficiente o inútil, como por ejemplo, programas que tardan

horas en procesar servicios, o que se bloquean con facilidad y que intercambian datos de manera equivocada.

5.3. Ventajas y uso de hilos.

Como consecuencia de compartir el espacio de memoria, los hilos aportan las siguientes ventajas sobre los procesos:

- Se consumen menos recursos en el lanzamiento y la ejecución de un hilo que en el lanzamiento y ejecución de un proceso.
- Se tarda menos tiempo en crear y terminar un hilo que un proceso.
- La conmutación entre hilos del mismo proceso o cambio de contexto es bastante más rápida que entre procesos.

Es por esas razones, por lo que a los hilos se les denomina también procesos ligeros.

Y ¿cuándo se aconseja utilizar hilos? Se aconseja utilizar hilos en una aplicación cuando:

- La aplicación maneja entradas de varios dispositivos de comunicación.
- La aplicación debe poder realizar diferentes tareas a la vez.
- Interesa diferenciar tareas con una prioridad variada. Por ejemplo, una prioridad alta para manejar tareas de tiempo crítico y una prioridad baja para otras tareas.
- La aplicación se va a ejecutar en un entorno multiprocesador.

Por ejemplo, imagina la siguiente situación:

Debes crear una aplicación que se ejecutará en un servidor para atender peticiones de clientes. Esta aplicación podría ser un servidor de bases de datos, o un servidor web.

Cuando se ejecuta el programa éste abre su puerto y queda a la escucha, esperando recibir peticiones.

Si cuando recibe una petición de un cliente se pone a procesarla para obtener una respuesta y devolverla, cualquier petición que reciba mientras tanto no podrá atenderla, puesto que está ocupado.

La solución será construir la aplicación con múltiples hilos de ejecución. En este caso, al ejecutar la aplicación se pone en marcha el hilo principal, que queda a la escucha.

Cuando el hilo principal recibe una petición, creará un nuevo hilo que se encarga de procesarla y generar la consulta, mientras tanto el hilo principal sigue a la escucha recibiendo peticiones y creando hilos.

De esta manera un gestor de bases de datos puede atender consultas de varios clientes, o un servidor web puede atender a miles de clientes.

Resumiendo, los hilos son idóneos para programar aplicaciones de entornos interactivos y en red, así como simuladores y animaciones.

Los hilos son más frecuentes de lo que parece. De hecho, todos los programas con interfaz gráfico son multihilo porque los eventos y las

rutinas de dibujado de las ventanas corren en un hilo distinto al principal. Por ejemplo en Java, AWT o la biblioteca gráfica Swing usan hilos.

5.4. Multihilo en Java. Librerías y clases.

Utilidades de concurrencia del paquete java.util.concurrent.

El paquete java.util.concurrent incluye una serie de clases que facilitan enormemente el desarrollo de aplicaciones multihilo y aplicaciones complejas, ya que están concebidas para utilizarse como bloques de diseño.
Concretamente estas utilidades están dentro de los siguientes paquetes:

- java.util.concurrent. En este paquete están definidos los siguientes elementos:
  - Clases de sincronización. Semaphore, CountDownLatch, CyclicBarrier y Exchanger.
  - Interfaces para separar la lógica de la ejecución, como por ejemplo Executor, ExecutorService, Callable y Future.
  - Interfaces para gestionar colas de hilos. BlockingQueque, LinkedBlokingQueque, nArrayBlockingQueque, SynchronousQueque, PriorityBlockingQueque y DelayQueque.

5.5. Creación de hilos.

En Java, un hilo se representa mediante una instancia de la clase java.lang.thread. Este objeto thread se emplea para iniciar, detener o cancelar la ejecución del hilo de ejecución. Los hilos o threads se pueden implementar o definir de dos formas:

a) Extendiendo la clase thread.
b) Mediante la interfaz Runnable .

En ambos casos, se debe proporcionar una definición del método run(), ya que este método es el que contiene el código que ejecutará el hilo, es decir, su comportamiento.
El procedimiento de construcción de un hilo es independiente de su uso, pues una vez creado se emplea de la misma forma. Entonces, ¿cuando utilizar uno u otro procedimiento?
Extender la clase thread es el procedimiento más sencillo, pero no siempre es posible. Si la clase ya hereda de alguna otra clase padre, no

será posible heredar también de la clase thread (recuerda que Java no permite la herencia múltiple), por lo que habrá que recurrir al otro procedimiento.
Implementar Runnable siempre es posible, es el procedimiento más general y también el más flexible.
Por ejemplo, piensa en la programación de applets, cualquiera de ellos tiene que heredar de la clase java.applet.Applet; y en consecuencia ya no puede heredar de thread si se quiere utilizar hilos. En este caso, no queda más remedio que crear los hilos implementando Runnable.
Cuando la Máquina Virtual Java (JVM) arranca la ejecución de un programa, ya hay un hilo ejecutándose, denominado hilo principal del programa, controlado por el método main(), que se ejecuta cuando comienza el programa y es el último hilo que termina su ejecución, ya que cuando este hilo finaliza, el programa termina.
Siempre hay un hilo que ejecuta el método main(), y por defecto, este hilo se llama "main".
Para saber qué hilo se está ejecutando en un momento dado, el hilo en curso, utilizamos el método currentThread() y que obtenemos su nombre invocando al método getName(), ambos de la clase thread.

5.6. Creación de hilos extendiendo la clase Thread.

Para definir y crear un hilo extendiendo la clase thread, haremos lo siguiente:
Crear una nueva clase que herede de la clase thread.
Redefinir en la nueva clase el método run() con el código asociado al hilo. Las sentencias que ejecutará el hilo.
Crear un objeto de la nueva clase thread. Éste será realmente el hilo.
Una vez creado el hilo, para ponerlo en marcha o iniciarlo:
Invocar al método start() del objeto thread (el hilo que hemos creado).

5.7. Creación de hilos mediante la interfaz Runnable.

Para definir y crear hilos implementando la interfaz Runnable seguiremos los siguientes pasos:
Declarar una nueva clase que implemente a Runnable.
Redefinir en la nueva clase el método run() con el código asociado al hilo. Lo que queremos que haga el hilo.
Crear un objeto de la nueva clase.
Crear un objeto de la clase thread pasando como argumento al constructor, el objeto cuya clase tiene el método run(). Este será realmente el hilo.
Una vez creado el hilo, para ponerlo en marcha o iniciarlo:

Invocar al método start() del objeto thread (el hilo que hemos creado).

En cualquier caso tenemos que tener siempre en cuenta las siguientes consideraciones:

- Siempre se debe sobreescribir (Override) el método run() e implementar allí lo que tiene que hacer el hilo.
- Podemos hacer que el hilo haga un número finito de cosas o bien que esté siempre en segundo plano (tendremos que asegurar que el método run()se ejecuta de forma continuada)(¿cómo se hace eso?)
- Los problemas vienen cuando existen varios hilos. Hay que tener en cuenta que pueden compartir datos y código y encontrarse en diferentes estados de ejecución
- La ejecución de nuestra aplicación será thread-safe si se puede garantizar una correcta manipulación de los datos que comparten los hilos de la aplicación sin resultados inesperados (más adelante veremos cómo)
- Además, en el caso de aplicaciones multihilo, también nos puede interesar sincronizar y comunicar unos hilos con otros.