Los primeros programas concurrentes fueron sistemas operativos en ordenadores de un solo procesador, el cual dividía el tiempo entre las distintas tareas. Ahora el mejor ejemplo de aplicaciones concurrentes es Internet, ya que en él se pueden ejecutar varias aplicaciones a la vez.

Con el avance de la tecnología los ordenadores ya cuentan con ***más de un procesador***, esto ***permite la programación en paralelo.*** También permiten sistemas distribuidos en los que cada ordenador ejecuta parte de una aplicación.

Gracias a los ***hilos / threads*** la ejecución de ***una aplicación es más rápida*** y más simple de programar. La programación multihilo es más compleja.

La ***programación multiproceso*** trata de dividir en partes más pequeñas / procesos la ejecución de la aplicación.

—------------------------------------------------------------------------------------------------------------------------

Un ***programa es estático***, un conjunto de instrucciones en código.

Un ***Proceso*** es ***dinámico***, es una ***instancia de un programa en ejecución***, este ***requiere instrucciones y recursos específicos***, además trabaja con independencia del resto.

Estos tienen ***estructuras de datos*** donde guardan la información asociada a la ejecución del proceso ***“ BCP / Bloque de Control de Proceso “.*** Cuando un proceso está en ejecución, se aloja en memoria con sus recursos asignados, cada proceso tiene su posición de memoria asignada y no podrán escribir en la de otro.

El ***Procesador*** es el componente encargado de ***ejecutar instrucciones y procesar datos***. Hay sistemas ***monoprocesadores y multiprocesadores***, en función de los procesadores que tenga el dispositivo.

Los Procesos Concurrentes son aquellos que se ejecutan a la vez, esto aumenta el rendimiento del sistema. La ***concurrencia*** de varios procesos ***en un mismo procesador,*** repartiéndose el tiempo de ejecución se conoce como ***multiprogramación***.

Las técnicas de concurrencia y multiprogramación son más eficientes en ***sistemas multiprocesador***, estos se consideran los que ***tienen más de un procesador o núcleo / CPU.*** Según su arquitectura ***se clasifican en :***

***Sistemas multiprocesadores fuertemente ensamblados :*** En esta arquitectura ***los procesadores comparten la misma memoria*** a la que están conectados mediante un bus, estos sistemas dependen del peso de ejecución de cada procesador.

***Estos se dividen en :***

***Sistemas multiproceso simétricos :*** Los ***procesadores*** son ***de características similares y trabajan entre iguales*** la ejecución de procesos.

***Sistemas multiproceso asimétricos :*** Tiene ***un procesador “maestro”*** ***que tiene control sobre los demás*** procesadores.

***Sistemas multiprocesadores débilmente ensamblados : Los*** ***procesadores no comparten la misma memoria,*** cada uno tiene una ***memoria asignada propia.*** Las ***ejecuciones que necesitan colaboración*** entre procesos usan el ***intercambio de mensajes mediante enlaces de comunicaciones,*** como en una red o ***Internet.***

—------------------------------------------------------------------------------------------------------------------------

La ***concurrencia / ejecución simultánea de procesos*** ***se ejecutarán a la vez o con tiempos compartidos*** como en ***Round Robin. Cuando esta se da en*** un sistema con ***un solo procesador*** se denominará ***multiprogramación,*** en cambio, si tiene ***más de un procesador*** se llamará ***programación paralela.***

Esta última ***puede tener problemas al acceder a los datos***, que no presentaría una ejecución secuencial. ***Para evitarlo se usan dos técnicas : El bloqueo y la comunicación.***

Las ***desventajas de la programación paralela*** son los controles que deben tener la comunicación y sincronización de los procesos para que se ejecuten correctamente. ***Cuando un proceso necesita*** ***datos de otro proceso, estos deben sincronizarse / comunicarse entre sí o esperar que alguno de los dos termine de ejecutarse.***

La ***Programación distribuida*** es un ***tipo de programación concurrente / paralela***, que se da en sistemas situados en ***lugares diferentes,*** interconectados en una red de comunicaciones. ***Los sistemas distribuidos no tienen memoria compartida, deben controlar la coherencia de los datos ya que usan réplicas de datos para su uso concurrente a través de mensajes.***

***Las ventajas del multiprocesamiento :***

Cuando se ejecutan varios procesos a la vez, la velocidad de ejecución global puede aumentar ya que hay más procesadores al servicio del usuario.

Los sistemas multiproceso son flexibles, distribuyen la carga de trabajo de los procesadores entre los procesos, además es fácil añadir más procesadores.

Estos diferencian los procesos por especialización y reservan procesadores para las ejecuciones complejas.

***Las desventajas del multiprocesamiento :***

Se deben al control que hay que realizar sobre los procesos en ejecución, ya que deben compartir información o comunicarse, lo que aumenta la complejidad de la programación. Si el multiprocesamiento se da en un sistema paralelo o distribuido el tráfico de buses se incrementa al número de procesadores que tiene.

—------------------------------------------------------------------------------------------------------------------------

Un ***proceso en segundo plano*** ***se denomina servicio.*** Depende de cómo se ejecuten los procesos serán :

***Foreground / Procesos en primer plano :*** Mantienen la comunicación con el usuario, como un juego por ejemplo.

***Background / Procesos en segundo plano :*** Son los que no se muestran explícitamente.

***Un servicio no tiene interfaz con el usuario, los ejecuta el sistema.***

La ***división de procesos*** en subprocesos se denominan ***hilos / threads.*** Estos ***comparten los recursos del proceso,*** a diferencia de los procesos si ***comparten memoria y usan variables globales.***

El ***cambio de contexto*** es cuando un procesador tiene que ***alternar la ejecución de varios procesos*** ***y restaura la memoria de cada uno al cambiar.***

Que cada ***proceso / entidad pesada*** tenga una ***memoria independiente consume muchos recursos*** del procesador. En cambio ni para la creación de ***hilos / entidades ligeras*** ni la comunicación ***consume tantos recursos*** del procesador.

Un proceso estará en ejecución siempre que uno de sus hilos esté activo, si se fuerza la detención de un proceso, sus hilos también se detendrán. ***Hay dos niveles de hilos :***

***Hilos de usuario / primer nivel :*** Son los que se crean con un lenguaje de programación.

***Hilos de sistema / segundo nivel :*** Son los que crea el sistema para apoyar a los anteriores.

—------------------------------------------------------------------------------------------------------------------------

El sistema operativo es el encargado de decidir qué proceso ejecutar y durante cuánto tiempo hay que hacerlo, este sigue una política de planificación. Los procesos tienen ratos de reposos en los que no usan el procesador porque están esperando datos, si este avisa al sistema se pondrá otro proceso en ejecución hasta que el proceso vuelva a quedar libre y avise de nuevo al sistema.

Los ***planificadores de procesos se basan en el estado de los procesos,*** estos deciden cuándo se ejecutan y cuando se terminan los procesos. ***Los procesos tienen tres estados generales :***

***Nuevo :*** Cuando un proceso es creado. > ***Preparado :*** Cuando está listo para hacer uso del procesador. > ***Ejecución.***

***El estado de ejecución tiene tres posibilidades :***

***Volver al estado Preparado*** y de nuevo al de ejecución. / ***Bloqueado :*** Cuando un proceso está esperando datos o que otro proceso termine / ***Terminado :*** Cuando termine su ejecución.

***Las transiciones del estado de un proceso son :***

***De ejecución a bloqueado :*** Cuando el proceso espera un dato o que otro proceso acabe.

***De ejecución a preparado :*** Cuando pasa mucho tiempo en ejecución y el sistema da paso a otro proceso, dejando al anterior en preparado.

***De preparado a ejecución :*** Se produce mediante el planificador de procesos.

***De bloqueado a preparado :*** El proceso que lo bloqueaba ha terminado o ya ha recibido los datos de e/s.

***De ejecución a terminado :*** Finaliza sus operaciones o el sistema lo finaliza.

Cuando un proceso cambia al estado de ejecución se produce el ***cambio de contexto.***

—------------------------------------------------------------------------------------------------------------------------

La ***planificación de procesos*** es un ***conjunto de protocolos*** que deciden el orden de ejecución de los procesos. Estos protocolos siguen criterios de prioridad según la importancia del proceso, el tiempo de ejecución, etc…

***El núcleo del sistema es el planificador / scheduler y opera a tres niveles diferentes :***

***Planificación a largo plazo :*** Cuando un proceso se crea se decide su planificación, su quantum, tiempo máximo en ejecución, etc…

***Planificación a medio plazo :*** Decide cuándo llevar a la memoria principal los datos de los procesos almacenados y cuáles hay que trasladar de la memoria principal al espacio de intercambio.

***Planificación a corto plazo :*** Minimizan el tiempo necesario para que se produzca un cambio de contexto cuando un proceso abandona el procesador y entra otro.

***Los sistemas multiprocesadores tienen dos componentes de planificación específicos :***

***Componente de planificación temporal :*** Es en el que se encuentran definidos las políticas / protocolos de planificación para cada procesador.

***Componente de planificación especial :*** En él se define cómo repartir los procesos entre los procesadores. Este usa criterios para aplicar las políticas / protocolos de asignación.

La ***concurrencia explícita*** es cuando ***el programador define*** en el algoritmo las ***pautas del paralelismo.***

En la ***concurrencia implícita*** es el ***sistema operativo*** el que ***decide*** cómo y cuáles procesos dividir, facilitando la tarea.

—------------------------------------------------------------------------------------------------------------------------

La ***concurrencia se da*** en Java mediante la ***clase java.lang.Thread*** y la ***interfaz java.lang.Runnable.*** Estas hacen el trabajo más complicado, hay dos librerías más eficientes para este tipo de programación :

***Executor :*** Es una mejora para la creación de hilos, divide el problema en otros más pequeños que se ejecutan por separado, en paralelo. La máquina virtual de Java es la que lo gestiona, en él sólo hay que indicar una tarea o tareas en instancias de interfaces Runnable (No devuelve nada) o Callable (Devuelve un tipo de dato primitivo).

***En la librería Executor tenemos tres tipos de gestores :***

***ThreadPoolExecutor :*** Clase base para un ***gestor genérico.*** Solucionan problemas que se resuelven ***ejecutando las tareas por partes.***

Hay tres ***formas de configurar*** estas instancias ***con métodos*** de la clase ***Executors :***

***newCachedThreadPool() :*** Crea un pool que genera hilos cuando los necesita y reutiliza los que están inactivos.

***newFixedThreadPool(int numThreads) :*** Crea un pool con el número fijo de hilos indicado en el parámetro. Las tareas se almacenan en cola y se asocian a los hilos cuando quedan libres.

***newSingleThreadExecutor() :*** Crea un pool con un solo hilo que procesa todas las tareas.

***execute() :*** Introduce una tarea Runnable/Callable cada vez.

***invokeAll() :*** Añade toda una lista de tareas a la vez.

***getPoolSize() :*** Devuelve el número de hilos en ejecución.

***getCompleteTaskCount() :*** Devuelve el número de tareas finalizadas.

***ShedulerThreadPoolExecutor :*** Es un gestor de hilos con ejecuciones de secuencia temporal, una o varias veces, como por ejemplo revisar correo cada x tiempo. Esta clase se instancia usando el ***método static de*** la clase padre ***Executor : newSchedulerThreadPool(int numThreads).***

Usa los siguientes métodos para gestionar el retraso :

***Schedule (Callable task, long delay, TimeUnit timeunit) :*** Crea y ejecuta una tarea que se llama después de un delay / retardo y se expresa en las unidades de tiempo indicadas (timeunit).

***Schedule (Runnable task, long delay, TimeUnit timeunit) :*** Igual que el anterior, sólo que se ejecutará una sola vez.

***ScheduleAtFixedRate (Runnable task, long initialDelay, long period, TimeUnit timeunit) :*** Crea y ejecuta una tarea de forma periódica. La primera vez se ejecutará después de un retraso inicial (initialDelay) y el resto en cada período de tiempo determinado (period), expresado en la unidad de tiempo indicada.

***ScheduleWithFixedDelay (Runnable task, long initialDelay, long delay, TimeUnit timeunit) :*** Crea y ejecuta una tarea (task) de forma periódica. La primera vez se invocará después de un retraso inicial (initialDelay) y posteriormente , al terminar la última tarea lanzada, no se empezará la siguiente, hasta que no haya pasado el tiempo

indicado en delay . El tiempo se mide en las unidades indicadas (timeunit).

Fork-Join : Es una evolución de la librería Executor. Este también tiene un gestor / clase ForkJoinPool que procesa las tareas con el algoritmo por robo work-stealing, que busca hilos poco activos y los intercambia por tareas en espera. Este método es ideal para los problemas recursivos.

ForkJoinTask es la clase padre de ForkJoinPool, de los esta derivan los métodos fork() y join().

El método fork()

El método join()

Si un sistema tiene un solo procesador, aunque tenga varios núcleos no será programación paralela, sino multiprogramación.

Executor es una interfaz que contiene tres clases.

Las interfaz RecursiveTask equivale a Callable y la interfaz RecursiveAction equivale a Runnable.