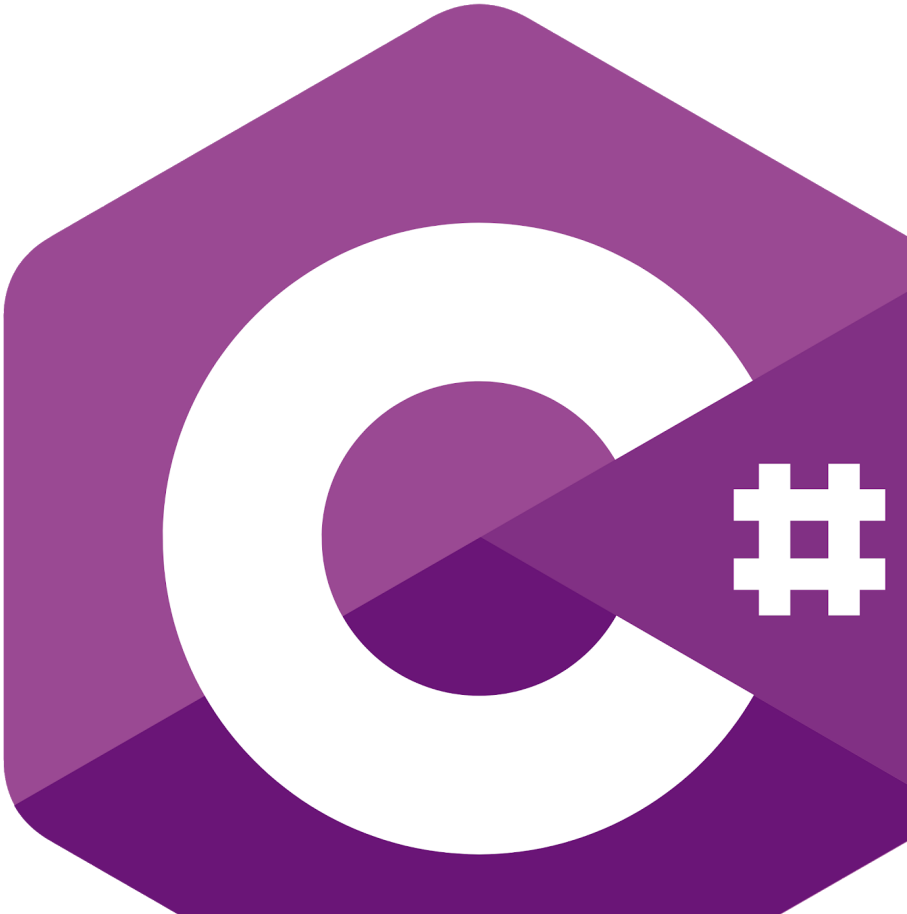
Programación Multihilo y Asíncrona

TECNOLOGÍAS DE DESARROLLO DE SOFTWARE IDE

Integrantes:
43697 Andrés Botello
43952 Franco Villegas
43740 Joaquín Romero



Introducción

¿Que es un hilo?

Un hilo es una tarea o secuencia de tareas encadenadas muy pequeña que se pueden ejecutar al mismo tiempo. Aquellos hilos que comparten recursos se pueden llamar *procesos*.

Programación Multihilo

La programación multihilo se basa en que, dentro del desarrollo de la aplicación, hemos definido diversas tareas para que se ejecuten a la vez. Todas estas tareas forman parte de un mismo proceso.

¿Cuando utilizamos programación multihilo?

El hecho de usar programación multihilo no implica que siempre vamos a tener una mejora de rendimiento y es potestad del programador el decidir cuándo conviene usar este tipo de programación. En líneas generales, si la aplicación que hemos desarrollado es muy simple no tiene sentido plantearse este tipo de programación. Tampoco puede que sea buena idea emplear el multihilo para aplicaciones excesivamente complejas pues provocará excesiva sobrecarga y sea más un problema que una ventaja.

Por regla general, se considera que una aplicación es candidata a ser programada en multihilo cuando cumple las siguientes premisas:

1. La aplicación va a requerir de realizar diversas tareas claramente diferenciadas con un alto coste computacional que sea mayor al que se genera en un cambio de contexto, es decir, de liberación del espacio de memoria asignado.
2. El resultado de las tareas que se ejecuten en diferentes hilos no debe depender del resultado de otras tareas, ya que, de lo contrario, las tareas se estarían esperando unas a otras y al final el rendimiento sería menor del esperado.
3. Se prevea que pueda haber tareas retenidas o bloqueadas por estar esperando a una lectura de disco, por ejemplo. En estos casos, esta tarea se bloquea y otra entra en acción, aprovechando de esta forma la programación multihilo.

Ejemplos

Para entender mejor lo que hemos comentado, pondremos dos ejemplos en los que puede resultar beneficioso el uso de la programación multihilo.

Ejemplo comparativo con programación secuencial

Uno de los casos de uso más comunes es cuando una aplicación precisa acceder a diferentes orígenes, como servidores o bases de datos, para procesar la información recogida de cada origen.

En programación secuencial, los accesos a estos orígenes serían secuenciales y hasta que no tuviésemos toda la información, no podríamos procesarla. Si por el contrario usamos programación multihilo, podríamos lanzar en paralelo el acceso a los diferentes orígenes provocando una mejora sensible del rendimiento.

Programación secuencial

1) Acceder a la primera base de datos
2) Realizar consulta
3) Esperar datos
4) Operar
5) Acceder a la segunda db
6) Realizar consulta
7) Esperar datos
8) Operar

Programación multihilo

- (Tarea 1) Acceder a la primer base de datos
- (Tarea 1) Realizar consulta (Tarea 2) Acceder a la segunda base de datos
- (Tarea 2) Realizar consulta (Tarea 1) esperar datos
- (Tarea 1) esperar (Tarea 2) esperar datos
- (Tarea 1) Operar
- (Tarea 2) Operar

Otro ejemplo aplicativo

Un segundo caso de uso tradicional de la programación multihilo es cuando dentro de una aplicación queremos priorizar alguno de los procesos que comporta sobre el resto. Por ejemplo, Es muy habitual usar este tipo de programación en videojuegos, priorizando la parte de renderización de la imagen sobre otros procesos paralelos que se ejecutan a la vez.

Programación Asíncrona

Antes de poder hablar de programación asíncrona debemos entender la diferencia entre ejecución síncrona y asíncrona.
En un modelo de programación sincrónica, las cosas suceden una a la vez. Cuando se llama a una función que realiza una acción de larga duración, sólo vuelve cuando la acción ha finalizado y puede devolver el resultado. Esto detiene el programa durante el tiempo que dure la acción.
Un modelo asincrónico permite que sucedan múltiples cosas al mismo tiempo. Al iniciar una acción, el programa continúa ejecutándose. Cuando la acción termina, el programa es informado y tiene acceso al resultado.
Podemos comparar la programación síncrona y asíncrona usando un pequeño ejemplo: un programa que obtiene dos recursos de la red y luego combina resultados.

En un entorno sincrónico, donde la función de petición sólo vuelve después de haber hecho su trabajo, la forma más fácil de realizar esta tarea es hacer las peticiones una tras otra. Esto tiene el inconveniente de que la segunda solicitud se iniciará sólo cuando la primera haya terminado. El tiempo total necesario será como mínimo la suma de los dos tiempos de respuesta.
La solución a este problema, en un sistema asíncrono, es poner en marcha hilos de control adicionales. Un hilo, como ya fue explicado anteriormente, es otro programa en ejecución cuya ejecución puede ser entrelazada con otros programas por el sistema operativo; ya que la mayoría de los ordenadores modernos contienen múltiples procesadores, múltiples hilos pueden incluso ejecutarse al mismo tiempo, en diferentes procesadores. Un segundo hilo podría iniciar la segunda petición, y luego ambos hilos esperan a que sus resultados vuelvan, después de lo cual se re-sincronizan para combinar sus resultados.

En el modelo sincrónico, el tiempo que toma la red es parte de la línea de tiempo para un hilo de control dado. En el modelo asíncrono, el inicio de una acción de red causa conceptualmente una división en la línea de tiempo. El programa que inició la acción continúa ejecutándose, y la acción ocurre junto a él, notificando al programa cuando finaliza.

Ventajas y desventajas

La programación asíncrona establece la posibilidad de hacer que algunas operaciones devuelvan el control al programa llamante antes de que hayan terminado mientras siguen operando en segundo plano.
Esto agiliza el proceso de ejecución y en general permite aumentar la escalabilidad, pero complica el razonamiento sobre el programa.

Ventaja: La programación asíncrona resulta ventajosa ya que aumenta el throughput soportado. Esta ventaja logra que tenga mucha tracción dada la demanda de sistemas de alta escalabilidad que se requieren en Internet.

Desventaja: El principal problema de la programación asíncrona se refiere a cómo dar continuidad a las operaciones no bloqueantes del algoritmo una vez que éstas han terminado su ejecución.

Modelos de programación asíncrona

Para dar respuesta al problema anterior –cómo dar tratamiento de continuidad al resultado de las operaciones no bloqueantes una vez que éstas han terminado– se han establecido diferentes modelos de programación asíncrona. Las bondades de dichos modelos se valoran en términos de cuánto permiten aproximar la programación asíncrona a un esquema lo más parecido al secuencial.

Modelo de paso de continuadores
Es el modelo de asincronía más utilizado dentro Node JS. Cada función recibe información acerca de cómo debe tratar el resultado –de éxito o error– de cada operación. Requiere orden superior.

Modelo de eventos
Se utiliza una arquitectura dirigida por eventos que permite a las operaciones no bloqueantes informar de su terminación mediante señales de éxito o fracaso. Requiere correlación para sincronizar.

Modelo de promesas
Se razona con los valores de retorno de las operaciones no bloqueantes de manera independiente del momento del tiempo en que dichos valores –de éxito o fallo– se obtengan.

Modelo de generadores
Se utilizan generadores para devolver temporalmente el control al programa llamante y retornar en un momento posterior a la rutina restaurando el estado en el punto que se abandonó su ejecución.

Bibliografía

- Programación asíncrona: paso de continuadores, eventos, promesas y generadores
- Breve introducción a la programación en paralelo en .NET
- Asynchronous programming in C
- Conceptos básicos de la programación asíncrona
- Qué es la programación multihilo y qué ventajas tiene
- Introducción a la programación multihilo
- Hilo (informática)