**Entendiendo hilos**

Imagine que su computadora tiene solo una unidad central de procesamiento (CPU) que es capaz de ejecutar solo una operación a la vez. Ahora, imagine lo que sucedería si la CPU tiene que trabajar duro para ejecutar una tarea que demore mucho tiempo. Mientras se ejecuta esta operación, todas las demás operaciones se detendrían. Esto significa que toda la máquina se congelaría y parecería que no responde al usuario.

Las cosas se ponen aún peor cuando esa operación de larga duración contiene un error por lo que nunca termina. Debido a que el resto de la máquina no se puede utilizar, lo único que puede hacer es reiniciar la máquina. Para remediar este problema, se utiliza el concepto de un hilo. En las versiones actuales de Windows, cada aplicación se ejecuta en su propio proceso. Un proceso aísla una aplicación de otras aplicaciones dándole su propia memoria virtual y asegurando que los diferentes procesos no puedan influirse entre sí. Cada proceso se ejecuta en su propio hilo. Un hilo es algo así como una CPU virtualizada. Si una aplicación se bloquea o llega a un bucle infinito, solo se ve afectado el proceso de la aplicación.

Windows debe administrar todos los subprocesos para garantizar que puedan hacer su trabajo. Estos gestionan cierto período de tiempo. Una vez finalizado este período, el subproceso se detiene y Windows cambia a otro subproceso. Esto se llama cambio de contexto. En la práctica, esto significa que Windows tiene que hacer algún trabajo para que esto suceda. El hilo actual está utilizando un área determinada de memoria; utiliza registros de la CPU y otros datos de estado, y Windows tiene que asegurarse de que todo el contexto del subproceso se guarda y se restaura en cada conmutador.

Pero, aunque hay ciertos éxitos de rendimiento, el uso de subprocesos garantiza que cada proceso tenga su tiempo de ejecución sin tener que esperar hasta que finalicen todas las demás operaciones. Esto mejora la capacidad de respuesta del sistema y da la ilusión de que una CPU puede ejecutar múltiples tareas a la vez. De esta manera, puede crear una aplicación que use paralelismo, lo que significa que puede ejecutar varios subprocesos en diferentes CPU en paralelo.

Casi cualquier dispositivo que compre hoy tiene una CPU con varios núcleos, que es similar a tener varias CPU. Algunos servidores no solo tienen CPU multinúcleo, sino que también tienen más de una CPU. Para hacer uso de todos estos núcleos, necesitas múltiples hilos. Windows garantiza que esos subprocesos se distribuyan a través de sus núcleos disponibles. De esta manera puede realizar múltiples tareas a la vez y mejorar la escalabilidad.

Debido a la sobrecarga asociada, debe determinar cuidadosamente si necesita multihilo. Pero si desea utilizar subprocesos para escalabilidad o capacidad de respuesta, C# y .NET Framework le ofrecen muchas posibilidades.

**Usando la clase Thread**

La clase Thread se puede encontrar en el espacio de nombres System.Threading. Esta clase le permite crear nuevas huellas, administrar su prioridad y obtener su estado.

La clase Thread no es algo que debas usar en tus aplicaciones, excepto cuando tienes necesidades especiales. Sin embargo, cuando utiliza la clase Thread, tiene control sobre todas las opciones de configuración. Por ejemplo, puede especificar la prioridad de su hilo, decirle a Windows que su hilo es de larga duración o configurar otras opciones avanzadas.

El Listado 1-1 muestra un ejemplo del uso de la clase Thread para ejecutar un método en otro hilo. La clase de consola sincroniza el uso de la secuencia de salida para que pueda escribir desde múltiples subprocesos. La sincronización es el mecanismo para garantizar que dos subprocesos no ejecuten una parte específica de su programa al mismo tiempo. En el caso de una aplicación de consola, esto significa que no hay dos hilos que puedan escribir datos en la pantalla al mismo tiempo. Si un subproceso está trabajando con la secuencia de salida, otros deberán esperar antes de que finalice.

[LISTADO 1-1 Creando un hilo con la clase Thread]

Como puede ver, ambos hilos se ejecutan e imprimen su mensaje en la consola. Se llama al método Thread.Join en el subproceso principal para que espere hasta que finalice el otro subproceso.

¿Por qué el Thread.Sleep (0)? Se utiliza para indicar a Windows que este hilo está terminado. En lugar de esperar a que finalice la totalidad del segmento de tiempo del hilo, cambiará inmediatamente a otro hilo.

Tanto tu proceso como tu hilo tienen una prioridad. Asignar una prioridad baja es útil para aplicaciones como un protector de pantalla. Tal aplicación no debe competir con otras aplicaciones por el tiempo de CPU. Un subproceso de prioridad más alta debe usarse solo cuando sea absolutamente necesario. A un nuevo hilo se le asigna una prioridad de Normal, lo cual está bien para casi todos los escenarios.

Otra cosa que es importante saber acerca de los hilos es la diferencia entre los hilos de primer plano y de fondo. Los hilos de primer plano se pueden usar para mantener viva una aplicación. Solo cuando todos los hilos de primer plano terminan, el Common Language Runtime (CLR) cierra su aplicación. Luego se terminan los hilos de fondo.

El listado 1-2 muestra esta diferencia en acción

[LISTADO 1-2 Usando un hilo de fondo]

El constructor Thread tiene otra sobrecarga que toma una instancia de un delegado ParameterizedThreadStart. Esta sobrecarga se puede usar si desea pasar algunos datos a través del método de inicio de su hilo a su método de trabajo, como muestra el Listado 1-3.

[LISTADO 1-3 Usando el ParameterizedThreadStart]