C++/CLI VI: Delegados e hilos

Esta no va a ser una entrada larga explicando qué es un delegado y cómo hacer *cositas* con él. Esta va ser una entrada de hombres para hombres, es decir, una entrada en la que explico algo que he leído en *C++/CLI in Action* y que considero muy interesante y muy cuca.

Vamos a utilizar un delegado para crear un hilo... pero sin hilos; voy a explicar cómo utilizar un *delegado asíncrono* para bifurcar el hilo principal de la aplicación y realizar dos tareas simultáneas.

Seguro que más de uno en sus programas utiliza hilos para realizar tareas fuera de la secuencia principal de su aplicación, como por ejemplo guardar un archivo muy grande mientras se deja al usuario que continúe con sus cosas. El procedimiento habitual es crearse un método miembro a la clase que hará en trabajo duro, seguido de una variable de tipo *Thread* que se instanciará más o menos así:

```
Thread ^hilo=gcnew Thread(gcnew ThreadStart(this,&Clase::Metodo));
hilo->Start();
```

Y luego es cosa nuestra monitorizar y esperar a que acabe el citado hilo.

Pero con delegados la cosa se puede simplificar mucho. Primero vamos a tratar la versión reducida:

```
using namespace System;

delegate void DoWorkDelegate(void);

void DoWork(void)
{
	Console::WriteLine("Start Job");
	System::Threading::Thread::Sleep(5000);
	Console::WriteLine("End Job");
}
int main(array<System::String ^> ^args)
{
	DoWorkDelegate ^dw=gcnew DoWorkDelegate(&DoWork);
	Console::WriteLine("Before Invocation");
	dw();
	Console::WriteLine("After Invocation");
	Console::ReadKey(true);

	return 0;
}
```

El código está claro. Primero definimos la firma del delegado, luego creamos un método que lo implementa y finalmente dentro de *main()* hacemos la llamada tras crearlo. La salida por pantalla debería ser similar a esta:

```
Before Invocation
Start Job
End Job
After Invocation
```

Nos damos cuenta de que primero se inicia el trabajo, luego se termina y finalmente cerramos el programa. De esta forma estamos bloqueando el hilo principal del programa. Si estuviéramos trabajando con WindowsForms, Windows nos diría que nuestra aplicación no responde y el usuario también estaría esperando a que la llamada al delegado terminara.

Pero si lo hacemos así:

```
delegate void DoWorkDelegate(void);

void DoWork(void)
{
	Console::WriteLine("Start Job");
	System::Threading::Thread::Sleep(10000);
	Console::WriteLine("End Job");
}

void WorkEnded(IAsyncResult ^)
{
	Console::WriteLine("The work has ended");
}
int main(array<System::String ^> ^args)
{
	DoWorkDelegate ^dw=gcnew DoWorkDelegate(&DoWork);
	Console::WriteLine("Before Invocation");
	dw->BeginInvoke(gcnew AsyncCallback(&WorkEnded),nullptr);
	Console::WriteLine("After Invocation");
	Console::ReadKey(true);
	return 0;
}
```

estamos realizando la llamada de forma asíncrona. El código sólo es un poco más complejo, pero a veces puede valer la pena.

Ahora añadimos un nuevo método que es una implementación del delegado *AsyncCallback* cuya firma ya trae el .NET preparada. Dicho *callback* se encargará de notificarnos de que la operación ha terminado.

Ya dentro de *main()* la invocación del delegado se realiza mediante el método *BeginInvoke()*, que toma una instancia del *callback* que nos notificará la finalización del proceso y otro objeto definido por el usuario que será pasado al parámetro de dicho *callback*.

Si ejecutáramos esto, obtendríamos la siguiente salida por pantalla:

```
Before Invocation
After Invocation
Start Job
End Job
The work has ended
```

Como podemos ver el código dentro de *main()* se ejecuta sin esperar a que lo haga el delegado, de modo que no estamos interrumpiendo el devenir del bucle principal. Deteniéndonos un poco en la salida por pantalla, primero se pintan las tres primeras líneas,

transcurren los diez segundos de rigor e inmediatamente se pintan las dos siguientes. La última es el *callback* que nos indica que el delegado ha terminado de realizar su trabajo.

Como ha comentado Augusto Ruiz, hay que tener en cuenta que el método *WorkEnded()* se está ejecutando desde el nuevo hilo y no desde el principal, así que debemos tener cuidado a la hora de hacer según qué cosas dentro de dicho método, así como en el propio delegado (como por ejemplo, acceder a la UI mediante llamadas a *Invoke()* en lugar de operar directamente con los componentes de la ficha).

Pero esto es solo el principio. La documentación explica más formas de realizar este tipo de llamadas asíncronas y aunque los ejemplos están hechos en C#, son fácilmente traducibles a C++/CLI.

Puedes obtener una copia de este artículo en PDF aquí.