Fonctions asynchrones et sockets en .NET

par Lainé Vincent

Date de publication : 04-02-2005

Dernière mise à jour :

Tutoriel sur les fonctions asynchrones et sockets en .Net/VB.NET. Ce tutoriel a pour but de donner les bases pour la programmation des sockets de façon asynchrone (non bloquante).

I - Introduction

A - But de ce tutoriel

Tout d'abord ce tutoriel n'a pas pour but une présentation exclusive des techniques réseaux au sein du Framework, mais plutôt une introduction au mécanisme des fonctions asynchrones au sein du réseau en .Net.

Qu'est ce qu'une fonction asynchrone ? Une fonction asynchrone rend la main au code appelant avant d'avoir terminé son traitement.

Comment savoir dans ce cas quand la fonction a terminé son traitement ? .NET met à notre disposition plusieurs mécanismes pour cela.

Nous allons voir comment utiliser l'un d'entre eux : les rappels (Callback).

B - Pré requis

Pour suivre ce tutoriel de manière aisée, il vous faut au minimum les connaissances de base en programmation avec le Framework .Net et en POO (Programmation Orientée Objet). Enfin une petite idée du fonctionnement d'un réseau et en particulier d'un socket peut être utile.

II - Le projet

Le projet se composera de deux applications distinctes. Une application serveur qui se mettra en attente d'une connexion et qui l'établira puis recevra des messages du client et une application cliente qui se connectera au serveur et enverra des messages au serveur.

A - L'application serveur

Voici à quoi ressemblera l'application serveur :

L'application serveur

Cette application est toute simple, elle ne sert qu'à lancer la connexion et afficher les messages. Le code du réseau et des événements seront mis dans une classe séparée de celle de la winform pour une meilleure compréhension.

B - L'application cliente

Voici à quoi ressemblera l'application cliente :

L'application cliente

Cette application se compose de 2 winforms. Une servant à spécifier le nom du serveur et l'autre servant à envoyer les messages.

III - Les fonctions asynchrones

A - Mécanisme des fonctions asynchrones

Les fonctions asynchrones utilisent le principe des Threads pour effectuer les actions de manière non bloquante. En effet quand vous faites appel à une des fonctions Begin*Fonction* de la classe socket, vous lui passez en paramètre une fonction de rappel (callback). Le programme continue son exécution sans attendre la fin de l'E/S sur le socket. Lorsque l'E/S est terminé, un thread system géré par .NET appelle le callback (rappel) passé à l'origine. Tout ce que vous avez à faire est d'implémenter les fonctions de rappels. Cette implémentation se fait grâce aux délégués coté système, par la création d'une instance de la classe AsyncCallback et la création des fonctions de rappel correspondantes coté client. Les fonctions de rappel doivent avoir la signature de AsyncCallback qui possède comme argument un type AsyncResult permettant de retrouver l'objet passé dans les fonctions Begin*Fonction*.

B - Le serveur

1 - La mise en attente d'une connexion

Pour attendre une connexion il faut utiliser la fonction BeginAccept. Avant de faire appel à BeginAccept, il faut lancer l'écoute sur le port voulu. Cela se fait grâce à la fonction Bind. Cette fonction prend comme argument un objet IPEndPoint. Pour finir le lancement de l'écoute, il faut faire appel à la fonction Listen. Cette fonction est non bloquante.

Attente d'une connexion

```
Me.SocketServer = New Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp)

Me.SocketServer.Bind(New IPEndPoint(192.168.0.1, 15))
Me.SocketServer.Listen(1)
```

Je ne reviendrai pas sur le constructeur d'un socket (ceci est développé dans un article sur développez.com) Le constructeur d' IPEndPoint prend comme argument une adresse IP et un numéro de port. La fonction Listen prend comme argument le nombre de connexions entrantes que le serveur peut mettre en file d'attente pour acceptation. C'est à dire que si vous passez 1 à Listen et que 3 clients se présentent en même temps, les deux derniers seront automatiquement refusés avant même l'examen de la demande par le socket serveur. Après avoir lancé l'écoute sur le port voulu grâce aux fonctions Bind et Listen il faut lancer l'attente de la connexion grâce à la fonction BeginAccept.

```
Sub BeginAccept(Dim CallBack As AsyncCallback, Dim obj As object)
```

Lancement de l'attente asynchrone

```
Me.SocketServer.BeginAccept(AddressOf ConnectionAcceptCallback, Me.SocketServer)
```

Le deuxième argument sera passé à la fonction connexionCallback, lorsque celle ci sera appelée, à travers la propriété AsyncState de IAsyncResult. Le premier argument est un objet de type AsyncCallback. AsyncCallback est un délégué, qui est le terme en .NET pour désigner un pointeur de fonction. Le constructeur prend comme argument une fonction ayant cette signature :

```
Sub connexionCallback(Dim asyncResult As IAsyncResult)
```

Retenez bien cette signature car elle servira pour toutes les fonctions de rappel des autres fonctions asynchrones. La fonction connexionCallback s'exécute dans un Thread séparé de celui de l'application principale (thread lancé par le système). Cette fonction prend comme argument un type IAsyncResult. L'interface IAsyncResult nous permet de retrouver l'objet que nous avons passé à la fonction BeginAccept tel qu'il était à travers sa propriété AsyncState.

```
object asyncResult.AsyncState
```

2 - Mettre fin à l'attente

Pour finir l'attente d'une connexion et pour finir d'accepter la connexion d'un client vous devez appeler la fonction EndAccept.

Fonction de rappel pour la connexion d'un client

```
Private Sub connexionAcceptCallback(ByVal asyncResult As IAsyncResult)
    SocketClient = SocketServer.EndAccept(asyncResult)
End Sub
```

3 - Accepter une connexion

EndAccept renvoie le socket avec lequel nous allons pouvoir envoyer les données vers le client. A partir de ce moment la connexion est établie entre le serveur et le client.

C - Le client

1 - Se connecter à un serveur

Pour le client, le but va être de se connecter au serveur et d'envoyer des messages à celui-ci. Pour se connecter au serveur nous allons utiliser la fonction BeginConnect

connexion au serveur

```
SocketClient.BeginConnect(New IPEndPoint("192.168.0.2", 15), AddressOf connexionConnectCallback, SocketClient)
```

Par rapport au serveur il n'y a rien de nouveau dans cette partie. Pour que la connexion soit complète il faut faire appel à la fonction EndConnect

Fin de la connexion au serveur

```
Private Sub connexionConnectCallback(ByVal asyncResult As IAsyncResult)
    SocketClient.EndConnect(asyncResult)
End Sub
```

D - Les fonctions communes

1 - Réception d'un message

Pour se mettre en attente d'un message nous allons utiliser la fonction BeginReceive

```
Sub BeginReceive(ByVal buffer As Byte(), ByVal offset As Integer, ByVal size As Integer,
```

```
        ByVal socketFlags As SocketFlags, ByVal asyncCallback As AsyncCallback, ByVal state As Object)
```

Mise en attente d'un message

```
    SocketClient.BeginReceive(buf, 0, buf.Length, SocketFlags.none, AddressOf ReceiveCallback, SocketClient)
```

Voici la fonction de rappel pour l'attente d'un message :

Fin de l'attente d'un message

```
    Private Sub ReceiveCallback(ByVal asyncResult As IAsyncResult)
        Dim read As Integer = SocketClient.EndReceive(asyncResult)
    End Sub
```

EndReceive renvoie le nombre d'octets reçus.

2 - Envoyer un message

Pour envoyer un message il faut utiliser la fonction BeginSend. Voici sa signature :

```
    Sub BeginSend(ByVal buffer As Byte(), ByVal offset As Integer, ByVal BufferLenght As Integer,
        ByVal socketFlags As SocketFlags, ByVal asyncCallBack As AsyncCallback, ByVal state As Object)
```

et voici comment l'utiliser :

Envoyer un message

```
    SocketClient.BeginSend(buf, 0, buf.Length, SocketFlags.none, AddressOf SendCallback, SocketClient)
```

Pour finir d'envoyer des données de façon asynchrone vous devez dans la méthode de rappel appeller la fonction EndSend.

Fonction de rappel pour l'envoie d'un message

```
    Private Sub SendCallback(ByVal asyncResult As IAsyncResult)
        Dim send As Integer = SocketClient.EndSend(asyncResult)
    End Sub
```

EndSend renvoie le nombre d'octets envoyés.

3 - Fermer la connexion

Le socket client qui souhaite fermer la connexion doit faire un Shutdown puis un Close.

Fermeture de la connexion coté client

```
    SocketClient.Shutdown(SocketShutdow.Both)
    SocketClient.Close
```

Notez que l'appel de shutdown ou close met fin à toute les E/S en attente en cours, et les fonctions de rappel correspondantes sont appellées. Pour le serveur il suffit de faire à appel à la fonction Close des sockets

Fermeture de la connexion coté serveur

```
    SocketServer.Close
```

IV - En résumé

Pour conclure cette article je tiens a préciser que cet article ne se veux pas complet mais donne les bases pour comprendre le fonctionnement des appels asynchrones des sockets en .Net. Pour plus de précision référez vous à la documentation du Framework .Net.

Pour résumé nous pouvons dire que la programmation avec les fonctions asynchrones se décompose en deux étapes :

- 1 : l'appel à la fonction asynchrone elle même. (les fonctions Begin*Fonction*)

- 2 : l'appel par la fonction asynchrone de la fonction de rappel qui doit elle même faire appel à la fonction de type : End*Fonction*

V - Bibliographie

Les articles de développez.com sont de bonnes bases :

Initiation au réseau

Utilisation des sockets en C#

Initiation aux délégués en C#

VI - Remerciements

Je tiens à remercier en particulier neo.51 et Abelman pour leur aide dans la rédaction de cet article ainsi que dans le debugage de l'application de démonstration.

Je remercie également Ukyuu pour son aide à la correction orthographique.

Plus généralement je remercie toute l'équipe de développez.com pour leur site vraiment très agréable.

VII - Téléchargement

Télécharger le projet complet en VB.NET: Le projet VB.NET (Projet SharpDevelop)