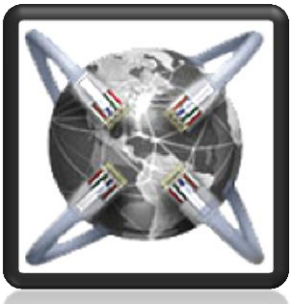
Chapter 4

TCP/UDP Asynchronous Socket Programming

المواضيع الرئيسية في هذا الفصل:

1- مدخل إلى برمجة الـ Asynchronous Socket بالدوت نت
2- معمارية البرمجة غير المتزامنة في الـ Asynchronous TCP Socket
3- معمارية البرمجة غير المتزامنة في الـ Asynchronous UDP Socket



4.1: مدخل إلى برمجة الـSocket غير المتزامن بالدوت نت :

بينا في الفصل الثاني كيف أن الفرق بين استخدام الـSocket المتزامن والأخر الغير متزامن هو في كون الأخير لا يحتاج إلى استخدام Thread منفصل عن البرنامج لتنفيذ عملية ما حيث يكتفي باستخدام معمارية شبيهة بمعمارية الـThreadPool إذ يتم مناداة الدالة من خلال Delegate خاص مع إمكانية تمرير بيانات إلى تلك الدالة من خلال الـState Object وتعتبر هذه المعمارية هي المستخدمة في الاتصال الغير متزامن وهي نفسها المستخدمة في الـThreadPool (لمزيد من المعلومات أنظر الفصل الثاني)، ففي الاتصال المتزامن لا يتم الانتقال إلى الدالة التالية إلى بعد الانتهاء من تنفيذ الدالة الحالية وفي هذه الحالة كنا نضطر إلى تنفيذ تلك الدالة على Thread منفصل وخاصة عندما تحتاج تلك الدالة إلى فترة زمنية للانتهاء ففي الدالة Receive على سبيل المثال يبقى البرنامج في وضع الانتظار إلى حين أن تصل كامل البيانات المرسلة لكن في حالة الاتصال غير المتزامن يتم تنفيذ الدالة بشكل مبدئي من خلال الدالة التي تبدأ بكلمة Begin وفي حالة ورود اتصال ما يتم إنهائها باستخدام الدالة التي تبدأ بكلمة End حيث يتم استدعائها من خلال Delegate وهو الـ AsyncCallback ويمكن أن يتم تمرير Object مع دالة الـBegin بشكل IAsyncResult Interface Object كالتالي:

Asynchronous Architecture:

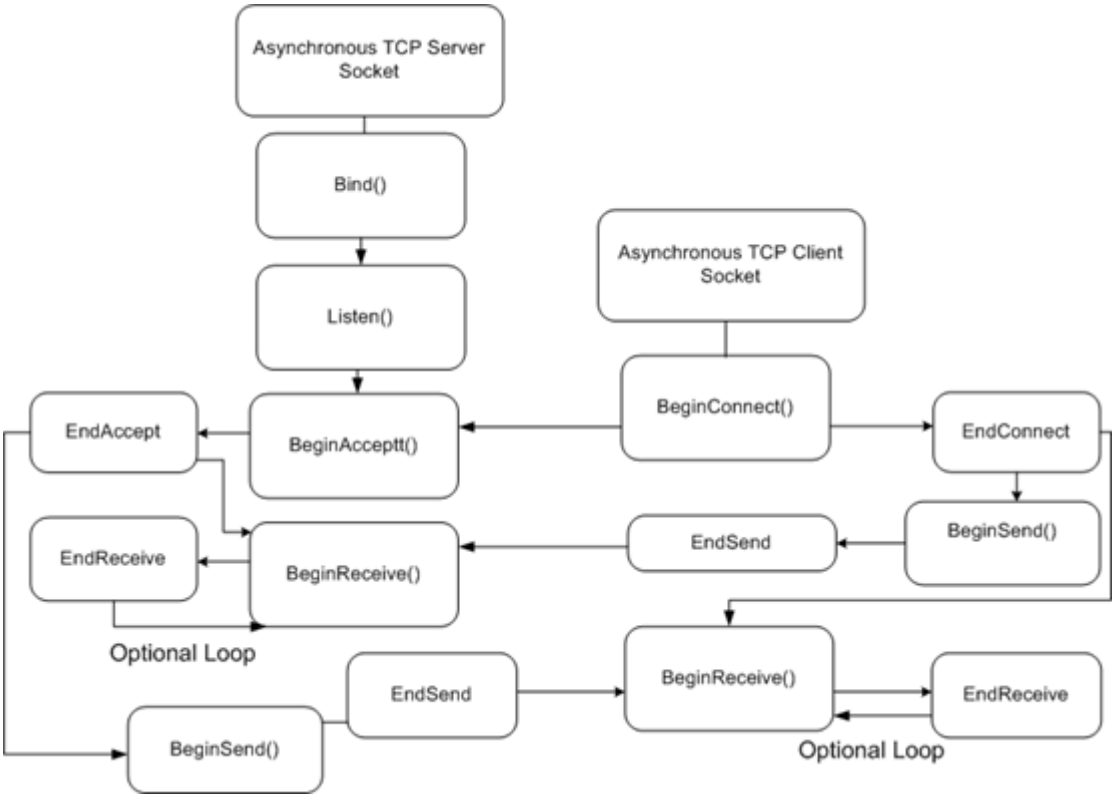
```
Socket socketobject // Socket Declaration

socketobject.Begin_ (new AsyncCallback(TheEndMethod), ObjectState);
// Call The EndMethod (should be void) By Using The AsyncCallback
Delegate And Pass the ObjectState For Example The Socket Object to
The EndMethod…
.
.
        void TheEndMethod(IAsyncResult PassedObjectState)
         {
         // Get The Passed Object (The SocketObject For Example) as
            IAsyncResult Interface object.
             NewSocketobject = PassedObjectState;
             NewSocketobject.End_(...);
         }
```

في الجزء التالي من هذا الفصل سيتم الحديث عن معمارية الـ Asynchronous TCP Socket وكيفية إنشاء تطبيق Client/Server بسيط باستخدام معمارية البرمجة غير المتزامنة.

4.2: معمارية البرمجة غير المتزامنة في الـ Asynchronous TCP Socket:

لا تختلف معمارية الـSocket في الاتصال غير المتزامن كثيرا عن الاتصال المتزامن ونستطيع أن نلخص هذه الفروق فقط بكيفية إنهاء العملية وعدم الحاجة لوجود Thread جديد لتنفيذها ويبين الشكل 4.1 معمارية الاتصال باستخدام الـTCP Asynchronous Socket :



الشكل 4.1 ويبين معمارية الـAsynchronous Client/Server TCP Socket

كما هو واضح في الشكل 4.1 فإن كل ما سنحتاج إليه لإنشاء تطبيق Client/Server باستخدام معمارية الـAsynchronous TCP Socket من جانب الـServer أولا هو إنشاء TCP Socket وعمل الـBind لـه وتفعيل الـListen كما هو في الاتصال المتزامن تماما ثم نأتي لمرحلة الـAccept وفي هذه المرحلة يتم استقبال طلب الاتصال من خلال الدالة BeginAccept وإنهاء عملية الموافقة على الانضمام من خلال الـEndAccept وبعد الموافقة على عملية الانضمام يمكن البدء بعملية الإرسال أو الاستقبال ويتم ذلك كما يلي كمثال:

Asynchronous TCP Socket Server:

```
using System.Net;
using System.Net.Sockets;
using System.Text;
.
.
.
.
```

```
Socket MyServerSocket;
Socket AcceptedSocket;
byte[] buffer = new byte[1024];
int PortNumber = 5000;

void StartTheServer()
{

MyServerSocket = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);

MyServerSocket.Bind(new IPEndPoint(IPAddress.Any, PortNumber));
MyServerSocket.Listen(-1);
MyServerSocket.BeginAccept(new AsyncCallback(End_Accept),
MyServerSocket);
}

void End_Accept(IAsyncResult ir)
{
// Cast The Passed Socket Object
Socket PassedSocketObject = (Socket)ir.AsyncState;

// End The Acceptance
AcceptedSocket = PassedSocketObject.EndAccept(ir);

// Begin Receiving ...
AcceptedSocket.BeginReceive(buffer, 0, buffer.Length,
SocketFlags.None, new AsyncCallback(End_Receive), AcceptedSocket);
}

void End_Receive(IAsyncResult ir)
{
AcceptedSocket.EndReceive(ir);

// You Can Show The ReceivedMessage On Any Control
string ReceivedMessage =
System.Text.UnicodeEncoding.Unicode.GetString(buffer);

// Begin Receiving Again (To Be As Infinity Loop)
AcceptedSocket.BeginReceive(buffer, 0, buffer.Length,
SocketFlags.None, new AsyncCallback(End_Receive), AcceptedSocket);
}
```

كما هو واضح في المثال السابق فإن عملية بناء الـ Asynchronous TCP Socket Server مرت في المراحل التالية:

- تعريف الـSocket وعمل الـBind ثم تفعيل الـListen على الـPort المحدد.
- تفعيل دالة الـBeginAccept حيث يتم من خلالها مناداة دالة الـEnd_Accept من خلال AsyncCallback Delegate ومررنا معها الـSocket Object والذي يحتوي على تعريف الـSocket الأساسي وليتم استخدامه في إنهاء عملية الموافقة.
- يجب أن تكون دالة End_Accept من النوع void حيث لا ترجع قيم ويجب أن تحتوي في تعريفها على بارومتر واحد من نوع IAsyncResult Interface وينطبق هذا الشكل على كافة دوال الـEnd في برمجة الـSocket غير المتزامن.
- في دالة End_Accept قمنا بتعريف Socket جديد للمتصل والهدف من هذه العملية هو أن يتم وضع كل متصل على Socket منفصل.
- في نفس الدالة يتم بدء عملية الاستقبال بمناداة BeginReceive Method حيث مررنا لها دالة End_Receive والـ Socket الخاص بالمتصل الجديد حيث ستبدأ عملية الاستقبال من ذلك المتصل ويمكننا من خلالها تكرار عملية الاستقبال بمناداة الدالة BeginReceive مرة أخرى في نهاية الدالة End_Receive حيث عند انتهاء عملية الاستقبال يتم استئنافها من جديد.

أما في جانب الـClient فكل ما سنحتاجه هو إنشاء TCP Socket وإجراء عملية الاتصال من خلال الدالة BeginConncet حيث يتم مناداة الدالة End_Conncet من خلال AsyncCallback Delegate ومن ثم البدء بعملية الإرسال والاستقبال وكما يلي كمثال:

Asynchronous TCP Socket Client:

```
Socket ClientSocket;

string ServerIP = "127.0.0.1";
int PortNumber = 5000;
byte[] buffer = new byte[1024];

void Begin_Conncet ()
{
ClientSocket = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);

IPEndPoint IPEnd = new IPEndPoint(IPAddress.Parse(ServerIP),
PortNumber);

ClientSocket.BeginConnect(IPEnd, new AsyncCallback(End_Conncet),
ClientSocket);
}

void End_Conncet(IAsyncResult ir)
{
ClientSocket.EndConnect(ir);
ClientSocket.BeginReceive(buffer, 0, buffer.Length, SocketFlags.None,
new AsyncCallback(End_Receive), ClientSocket);
}
```

```
void End_Receive(IAsyncResult ir)
{
ClientSocket.EndReceive(ir);

// You Can Show The ReceivedMessage On Any Control
string ReceivedMessage =
System.Text.UnicodeEncoding.Unicode.GetString(buffer);
ClientSocket.BeginReceive(buffer, 0, buffer.Length, SocketFlags.None,
new AsyncCallback(End_Receive), ClientSocket);
}

void Begin_Send(string Message)
{
byte[] SendingBuffer =
System.Text.UnicodeEncoding.Unicode.GetBytes(Message);
ClientSocket.BeginSend(SendingBuffer, 0, SendingBuffer.Length,
SocketFlags.None, new AsyncCallback(End_Send), ClientSocket);
}

void End_Send(IAsyncResult ir)
{
ClientSocket.EndSend(ir);
}
```

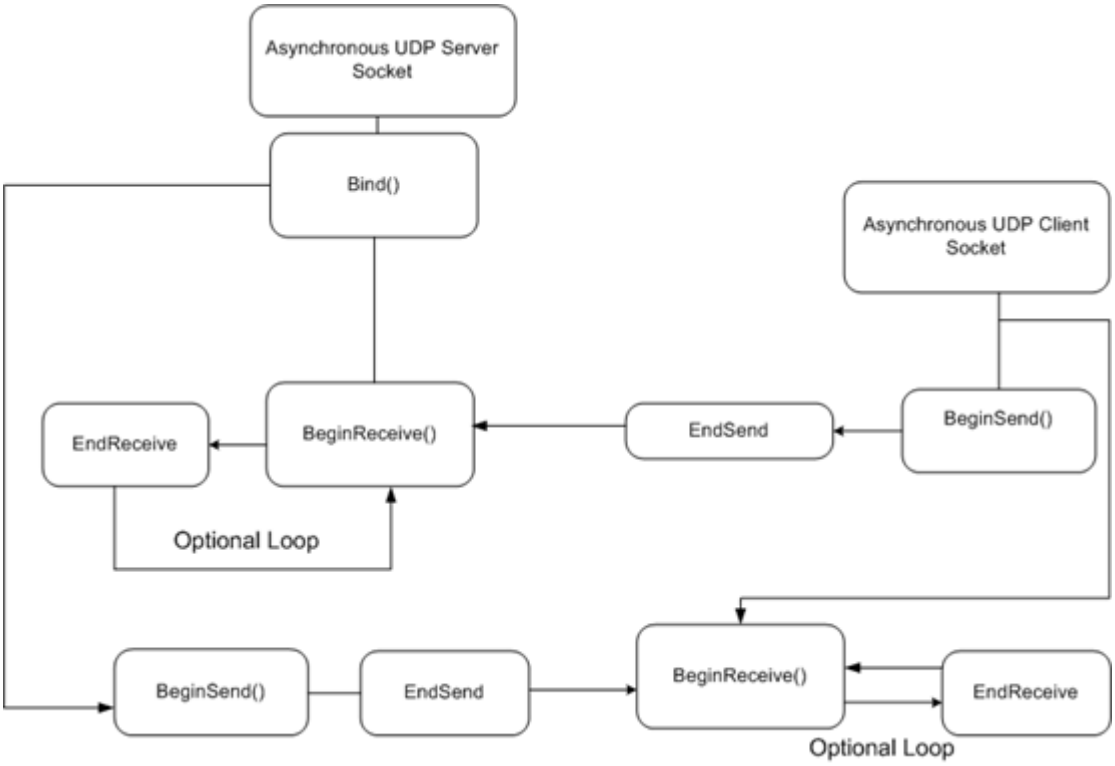
كما هو واضح في المثال السابق فإن عملية بناء الـ Asynchronous TCP Socket Client مرت في المراحل التالية:

- تعريف الـSocket وإنشاء عملية الاتصال من خلال الدالة BeginConncet حيث يتم من خلالها إجراء الاتصال مع الـServer ويمرر لها الـIPEndPoint Object والذي يحتوي على الـ IP Address والـPort Number الخاص بالسيرفر وفي الباروميتر الثاني يتم مناداة الدالة End_Conncet من خلال AsyncCallback Delegate ومررنا في الباروميتر الثالث الـSocket Object والذي يحتوي على تعريف الـSocket الأساسي وليتم استخدامه في إنهاء عملية الاتصال مع السيرفر.
- بعد الانتهاء من إجراء عملية الاتصال ونجاحها يتم مناداة الدالة BeginReceive حيث تبدأ عملية الاستقبال.
- تتم عملية الإرسال من خلال تمرير الرسالة المراد إرسالها إلى الدالة Begin_Send حيث تحول إلى Bytes وترسل باستخدام الدالة BeginSend حيث يتم إنهاء عملية الإرسال باستدعاء دالة الـEnd_Send من خلال الـ AsyncCallback Delegate.

4.3: معمارية البرمجة غير المتزامنة في الـ Asynchronous UDP Socket:

كما في الاتصال المتزامن في الـUDP فإن عملية البرمجة من طرف الـServer تبدأ بتعريف الـSocket ثم عمل الـBind والبدء بعملية الإرسال باستخدام الدالة BeginSendTo وإنهائها من خلال الدالة EndSendTo وكذلك الاستقبال من خلال الدالة BeginReceive وإنهاء عملية الاستقبال من خلال الدالة

EndReceive وفي طرف الـClient نكتفي بتعريف الـSocket والبدء بعملية الإرسال والاستقبال كما في الـServer ويبين الشكل 4.2 معمارية الاتصال غير المتزامن من خلال بروتوكول الـUDP.



الشكل 4.2 ويبين معمارية الـAsynchronous Client/Server UDP Socket

بناء على الشكل 4.2 يتضح أن معمارية البرمجة غير المتزامنة في الـUDP هي نفسها كما في البرمجة المتزامنة غير أن الاختلاف هو فقط في كيفية تطبيق دوال الإرسال والاستقبال، وكمثال على ذلك سنقوم في البداية بإنشاء جزء الـServer وتنفيذ عملية الاستقبال والإرسال عليه وكما يلي كمثال:

Asynchronous UDP Socket Server:

```
using System.Net;
using System.Net.Sockets;
using System.Text;
.
.
.
int ServerPort = 5000;
byte[] buffer = new byte[1024];
Socket ServerSocket;

void StartUDPServer()
{
ServerSocket = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp);
```

```
ServerSocket.Bind(new IPEndPoint(IPAddress.Any, ServerPort));

ServerSocket.BeginReceive(buffer, 0, buffer.Length, SocketFlags.None,
new AsyncCallback(End_Receive), ServerSocket);
}

void End_Receive(IAsyncResult ir)
{
ServerSocket.EndReceive(ir);

string ReceivedMessage =
System.Text.UnicodeEncoding.Unicode.GetString(buffer);

// Looping
ServerSocket.BeginReceive(buffer, 0, buffer.Length, SocketFlags.None,
new AsyncCallback(End_Receive), ServerSocket);
}
```

وفي طرف الـClient سنحتاج فقط إلى تعريف الـSocket ومن ثم البدء بعملية الإرسال أو الاستقبال وكما يلي كمثال:

Asynchronous UDP Socket Client:

```
using System.Net;
using System.Net.Sockets;
using System.Text;
.
.
.

Socket ClientSocket;

string ServerIP = "127.0.0.1";
int ServerPort = 5000;

void UdpClientSend(string Message)
{

ClientSocket = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp);

IPEndPoint IpEnd = new IPEndPoint(IPAddress.Parse(ServerIP),
ServerPort);

byte[] TempBufferToSend =
System.Text.UnicodeEncoding.Unicode.GetBytes(Message);

ClientSocket.BeginSendTo(TempBufferToSend, 0,
TempBufferToSend.Length,
SocketFlags.None, IpEnd, new AsyncCallback(End_Send), ClientSocket);

}
```

```
void End_Send(IAsyncResult ir)
{
ClientSocket.EndSendTo(ir);
}
```

أما بنسبة لعملية الإرسال كـMulticast و Broadcast فلا تختلف بشيء فقط بتطبيق قواعد الإرسال والاستقبال كتفعيل الـBroadcast على الـSocket من خلال الـSocketOption في حالة الإرسال كـBroadcast وكذلك عملية الانضمام إلى المجموعة في حالة الإرسال أو الاستقبال من Multicast Group ويمكن الرجوع إلى الفصل الثالث لمزيد من المعلومات عن كيفية تطبيق ذلك.

الخلاصة:

بينا في هذا الفصل كيفية تطبيق قواعد البرمجة غير المتزامنة على الـSocket في كلا البروتوكولين الـTCP والـUDP وبينا كيف أن معمارية البرمجة غير المتزامنة شبيهة إلى حد بعيد بعملية الـThreadPool حيث يتم مناداة الدالة التي تحتوي على إنهاء العملية من خلال Delegate خاص ثم تمرير الـSocket Object من خلال دالة الـBegin إلى دالة الـEnd حيث تحتوي على تعريف الـSocket الذي تم استخدامه في عملية الاتصال. في الفصل التالي سيتم الحديث عن استخدام دوال الـStreaming في الـNET. وكيفية إرسال بيانات Binary وكذلك تطبيق مفهوم الـSerialization لإرسال Objects عبر الـSocket.