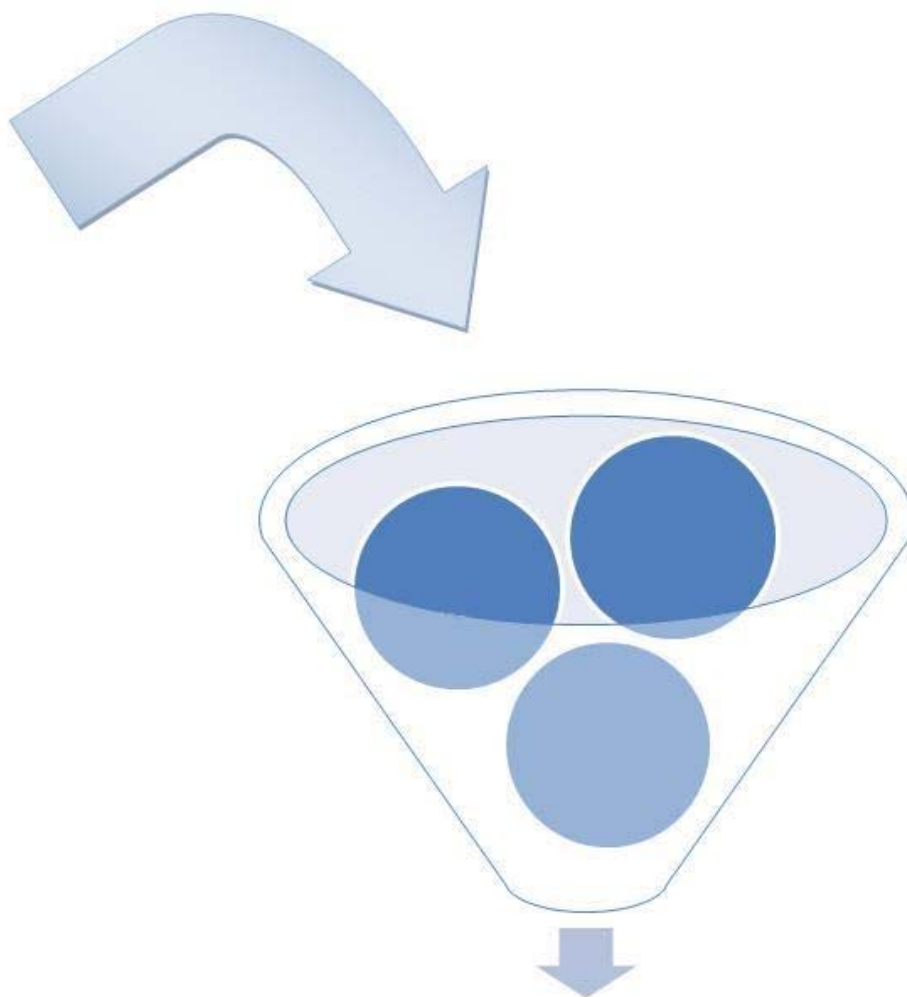


Chapter2: Introduction to Multithreading Programming

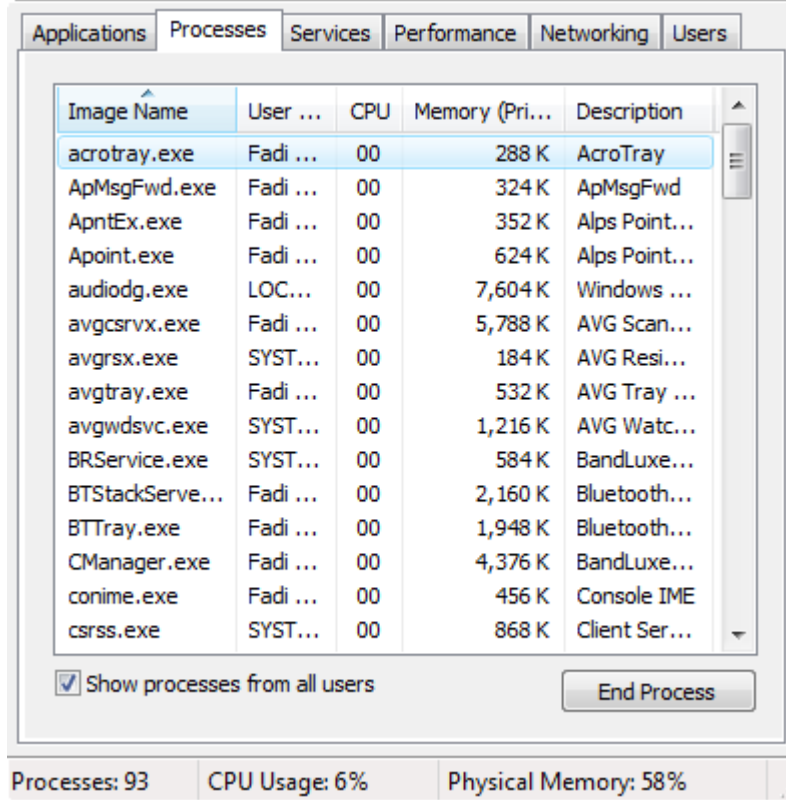


المواضيع الرئيسية في هذا الفصل:

- 1- المفهوم العام لل-Threading وال-Multithreading
- 2- مقدمة في برمجة ال-Threading في الدوت نيت.
- 3- مقدمة في استخدام ال-Multithreading في برمجة الشبكات.

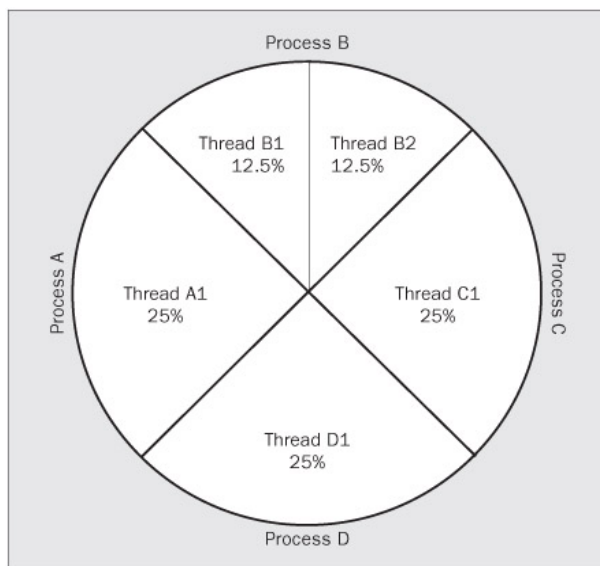
2.1: المفهوم العام للـ Threading و الـ Multithreading:

عزز مفهوم الـ Multithreading مفهوم الـ Multitasking في أنظمة التشغيل بحيث أن أي برنامج يمكنه أن يعمل على Thread منفصل عن الآخر بالإضافة إلى إمكانية أن يعمل نفس البرنامج على أكثر من Thread، أتى هذا المفهوم بعد تطوير فكرة الـ Multitasking في أنظمة التشغيل، ويتم إدارة هذه العمليات من قبل نظام التشغيل الذي يقوم بتقسيم المهام على المعالج وفق الأولويات لكل Thread وعلى فترات زمنية، ويمكن مشاهدة هذه العمليات في نظام التشغيل من خلال الـ Task Manager وكما في الشكل 2.1



الشكل 2.1 ويبين مجموعة الـ Processes التي يقوم نظام التشغيل بإدارتها أثناء Runtime

لاحظ أن كل برنامج يحجز مقدار معين من وقت المعالج بناء على حاجته ويقوم نظام التشغيل بتقسيم المهام على المعالج وفق الحاجة والأولويات ويبين الشكل 2.2 كيف يقسم المعالج دورة المعالجة على مجموعة من المهام والتي تحتوي على مجموعة من الـ Threads وإعطاء كل منها نسبة معينة من اهتمام المعالج وفق الحاجة والأولويات لكل واحدة منها.



الشكل 2.2 ويبين كيف يقوم المعالج بتقسيم اهتمامه على المهام

يمكن التحكم بأي Thread في نظام التشغيل إذ يمكننا من عمل مقاطعة (Interrupt) له كما يمكننا إيقافه بشكل مؤقت (Pause)، وأيضا إلغائه بعمل (Abort) له، كل هذه العمليات متاحة للمستخدم وأيضا متاح التحكم بها برمجيا.

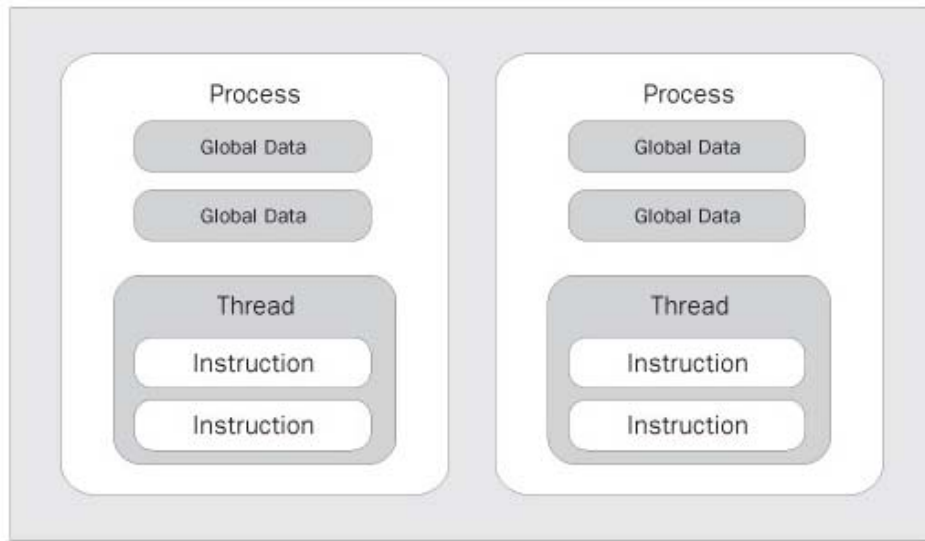
تنويه:

يهتم هذا الفصل بالجوانب المتعلقة بكيفية إدارة واستخدام Multithreading في برمجيات الشبكات ولا يهتم بالكيفية التي يقوم بها المعالج ونظام التشغيل بإدارة هذه العمليات لذلك في حالة لم تكن لديك المعرفة الكافية بكيفية قيام نظام التشغيل بإدارة هذه العمليات فيإمكانك الاستعانة بكتاب متخصص في أنظمة التشغيل وننصح بالكتابين التاليين ككتابين متخصصين في هذا الجانب:

- 1 Modern Operating Systems
- 2 Tobin Titus , A Press, C# Threading Handbook

2.2: مقدمة في برمجة الThreading في الدوت نيت:

تطور مفهوم البرنامج من كونه مجموعة من التعليمات المتتالية التي يمر عليها المعالج خطوة بخطوة إلى مجموعة من المهام التي تعمل بشكل متوازي ومتزامن مع بعضها البعض وأقرب مثال لنا هو نظام التشغيل والبرمجيات التي تعمل عليه وكمثال مبسط أكثر برنامج Microsoft Word إذ يقوم المستخدم بالكتابة ويقوم بدوره بعمل Check spelling لما يكتب بشكل متوازي مع عملية الكتابة ويسمى في هذه الحالة برنامج Microsoft Word بالProcess وتسمى العمليات التي تعمل بداخله بشكل متوازي بالThreads كعملية check spelling التي ذكرناها أنفاً لاحظ الشكل 2.3، ولا يخلو أي برنامج في الوقت الحالي من استخدام الThreading ففيه يتمكن المبرمج من تنفيذ عدة عمليات بنفس الوقت ويعتبر استخدام الThreading أيضاً غاية في الأهمية في الدوال التي تحتاج إلى وقت لتنفيذ ومنها كمثال لا الحصر تحميل بيانات من الإنترنت أو عند تنفيذ عمليات تستخدم Input/output Devices أو في الدوال التي تحتوي على جمل Infinity Looping أو Long Looping.



الشكل 2.3 ويبين كيف يمكن أن يحتوي الProcess الواحد على أكثر من Thread

تدعم الدوت نيت استخدام الThreading في برمجياتها من خلال فضاء الأسماء System.Threading Namespace ويحتوي على مجموعة من الدوال والخواص التي تمكننا من إنشاء وإدارة الThreads في التطبيقات التي نقوم ببرمجتها.

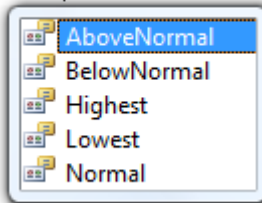
2.2.1: أهم الـClasses الموجودة في System.Threading Namespace:

Class	Description
Thread	ويحتوي على الدوال والخصائص الرئيسية لإنشاء والتحكم بالـThreading ويمكن من خلاله إنشاء والتحكم في تنفيذ عدد من الـThreads في نفس الوقت بشكل متصل أو منفصل عن التطبيق Main Thread.
ThreadPool	ويستخدم لمناداة دالة ما أثناء تنفيذ البرنامج بطريقة غير متزامنة على Thread متصل بالتطبيق بنفس مبدأ الـTimer و ينتهي مع انتهاء تنفيذ البرنامج ويسمى في هذه الحالة بالـBackground Thread.

2.2.1.1: أهم الدوال Methods الموجودة في Thread Class:

Method	Description
Start	ويتم من خلاله بدأ تنفيذ الـThread وتحتوي هذه الدالة على إمكانية تمرير باروميتر معين إلى Thread Method التي يتم مناداتها من خلال الـThread Delegate
Abort	ومن خلالها يمكن إيقاف عمل الـThread في أي وقت أثناء التشغيل Runtime Mode وتفرغ توابعه من الذاكرة.
Join	وفيه يمكن تحديد فترة زمنية معينة لتنفيذ الـThread قبل إيقافه
Suspend	إيقاف تنفيذ عمل الـThread بشكل مؤقت.
Resume	استئناف تنفيذ الـThread بعد عملية Suspend.
Sleep	وتستخدم لتجميد التنفيذ أي الانتقال إلى تنفيذ السطر التالي من البرنامج لفترة زمنية معينة بالميكرو ثانية (كل ثانية تساوي 1000 مايكرو ثانية).

2.2.1.2: أهم الخصائص Properties الموجودة في Thread Class:

Property	Description
IsBackground	وهي خاصية من نوع Boolean وتستخدم لتحديد فيما إذا كان ذلك الـThread تابع للـMain Thread أم لا وبالتالي إنهائه حال إنهاء الـMain Thread
Priority	وهي خاصية يتم من خلالها تحديد أولوية تنفيذ الـThread بالنسبة إلى نظام التشغيل وتأخذ خمسة حالات وهي كما يلي: <code>.Priority = ThreadPriority.</code> 

IsThreadPoolThread	وهي خاصية تحدد فيما إذا كان ذلك الـ Thread مدار من خلال ThreadPool أم لا.
IsAlive	وهي خاصية تحدد فيما إذا كان ذلك الـ Thread فعال أم لا.

2.2.1.3: أهم Delegates التي تستخدم مع الـ Thread Class والـ ThreadPool:

Delegate	Description
ParameterizedThreadStart	وتستخدم لمناداة الـ Method التي نود تنفيذها على Thread منفصل مع إمكانية تمرير باروميتر من نوع object إليها عند عمل Start لها.
ThreadStart	وتستخدم لمناداة الـ Method التي نود تنفيذها على Thread منفصل بشرط ألا تحتوي تلك الـ Method على أي باروميتر.
WaitCallback	وهي الوحيدة التي تستخدم مع الـ ThreadPool وتعمل على مناداة الـ Method التي نريد تنفيذها من خلال الـ ThreadPool ونستطيع من خلالها تمرير باروميتر من نوع Object للـ Method.



For More Details In Threading Classes/Methods/Properties Please Check MSDN.Com

2.2.2: التركيب العام للـ Threading Classes بالدوت نيت وكيفية استخدامه:

وأما فيما يخص طريقة استخدام الـ Thread والـ ThreadPool فلهما تركيب متشابه نسبيا غير أن الأخيرة لا تحتاج لعمل Start لتنفيذ الـ Thread كما أنها تعتبر Background Thread بشكل افتراضي بحيث تنتهي مع انتهاء عمل الـ Main Thread وتتسابقان بكونهما يشترطان بأن تكون دالة الـ Thread من نوع void وبالتالي لا ترجع قيم. وأما والتركيب العام لهما فهو كالتالي:

أولا: Thread Class:

C#: Thread Class Without Parameters:

```
using System;
using System.Threading;
public class My_First_Sample_In_Threading
{
    void Start_The_Method_as_New_Thread()
    {
        Thread th = new Thread(
            new ThreadStart(Method_Needs_Threading) );
        th.Start();
    }
    void Method_Needs_Threading()
    {
        bool infinity = true;
        int Count = 0;
        while (infinity)
        {
            Console.WriteLine(Count);
        }
    }
}
```

```

        Count++;
        Thread.Sleep(1000);
    }
}

```

C#: Thread Class With Passing Parameter:

```

using System;
using System.Threading;
public class My_First_Sample_In_Threading
{
    void Start_The_Method_as_New_Thread()
    {
        int startingvalue = 10;
        Thread th = new Thread(
            new ParameterizedThreadStart (Method_Needs_Threading) );
        th.Start(startingvalue);
    }
    void Method_Needs_Threading(object PassedValue)
    {
        bool infinity = true;
        int Count = (int)PassedValue;

        while (infinity)
        {
            Console.WriteLine(Count);
            Count++;
            Thread.Sleep(1000);
        }
    }
}

```

ثانياً: ThreadPool Class

C#: ThreadPool Class Without Passing Parameter

```

using System;
using System.Threading;
public class My_First_Sample_In_ThreadPool
{
    void Start_The_Method_as_New_ThreadPool()
    {
        ThreadPool.QueueUserWorkItem(new
            WaitCallback(Method_Needs_Threading));
    }
    void Method_Needs_Threading(object state)
    {
        bool infinity = true;
    }
}

```

```

    int Count = 0;

    while (infinity)
    {
        Console.WriteLine(Count);
        Count++;
        Thread.Sleep(1000);
    }
}

```

C#: ThreadPool Class With Passing Parameter

```

using System;
using System.Threading;
public class My_First_Sample_In_ThreadPool
{
    void Start_The_Method_as_New_ThreadPool()
    {
        int startingvalue = 10;
        ThreadPool.QueueUserWorkItem(new
            WaitCallback(Method_Needs_Threading), startingvalue);
    }

    void Method_Needs_Threading(object PassedValue)
    {
        bool infinity = true;
        int Count = (int) PassedValue;

        while (infinity)
        {
            Console.WriteLine(Count);
            Count++;
            Thread.Sleep(1000);
        }
    }
}

```

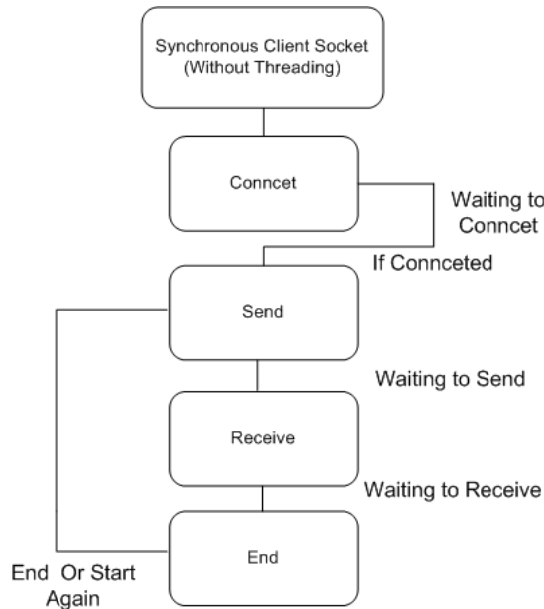
يوضح في كلا المثالين كيف انه يمكن تنفيذ دالة بطريقة غير متزامنة مع تنفيذ البرنامج بحيث أن تلك الدالة تحتوي على Infinity Loop وبالتالي فإنها ستبقى تعمل إلى الأبد ولو أننا لم نستخدم الـ Threading لكان من الاستحالة عمل دالة شبيهة بتلك الدالة في برامجنا إذ أن تنفيذ تلك الدالة بالطريقة المعتادة سيمنع المعالج من إكمال تنفيذ بقية البرنامج.

2.3: مقدمة في استخدام Multithreading في برمجة الشبكات:

ما يهمنا في هذا الفصل هو معرفة متى سنحتاج إلى استخدام Threading في تطبيقات الشبكات والفرق بين استخدام البرمجة المتزامنة وغير المتزامنة بمدى التأثير على أدائية النظام من جهة وسرعة وطريقة تنفيذ المهام المطلوبة من جهة أخرى.

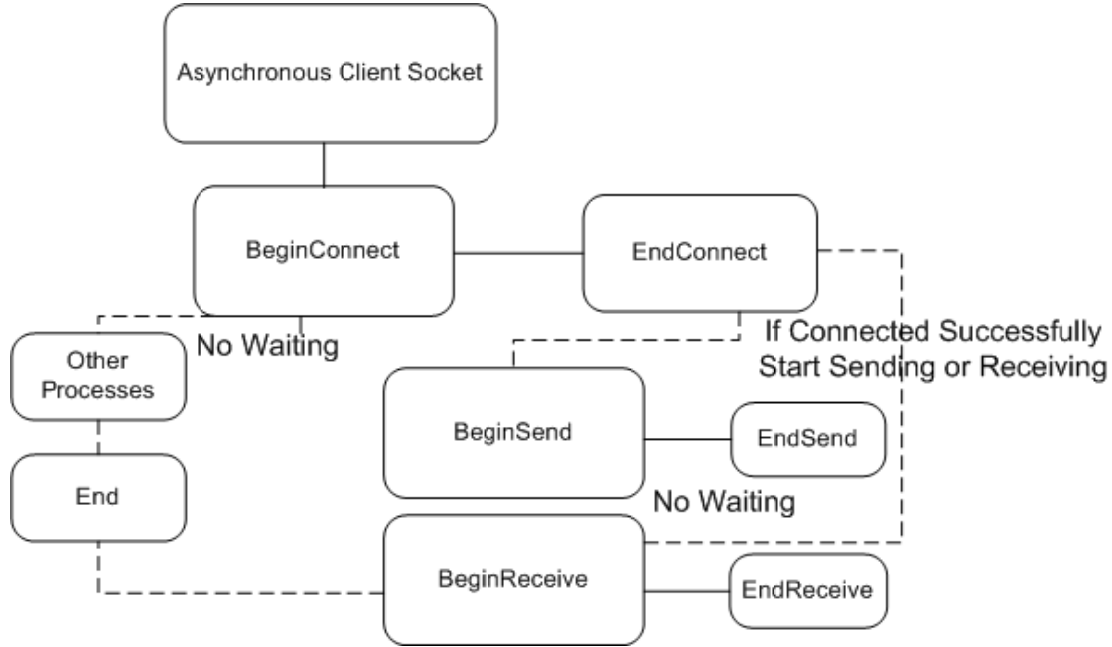
لاحظنا في الجزء السابق من هذا الفصل كيف أننا نستطيع مناداة دالة ما من خلال Thread والـ ThreadPool وبالتالي تنفيذ تلك الدالة بطريقة غير متزامنة مع Main Thread (والمقصود به التطبيق نفسه) وبالتالي فإن مناداة أي دالة بطريقة غير متزامنة تسمح لتطبيق بإكمال تنفيذ بقية البرنامج مع استمرار تنفيذ تلك الدالة. ويعتبر استخدام الـ ThreadPool أكثر أماناً من استخدام الـ Thread Class والسبب أن الأخير قد يبقى في وضع التنفيذ حتى بعد إنهاء عمل البرنامج ولاستخدامه بشكل أكثر أماناً يفضل تفعيل الخاصية IsBackground وبالتالي جعله ينتهي مع إغلاق البرنامج ولكن استخدام الـ Thread Class يسمح لنا بالتحكم بشكل أكبر بتنفيذ دالة ما من حيث إمكانية تنفيذها أو إيقاف تنفيذها والتحكم بأولويتها للمعالجة أثناء تشغيل التطبيق كما أن تنفيذ مجموعة كبيرة من الـ ThreadPool على نفس التطبيق قد يؤدي إلى جعل التطبيق بطيء في تنفيذ مهامه على عكس تنفيذ مجموعة كبيرة من الدوال من خلال الـ Thread Class إذ لا يؤثر تنفيذها على التطبيق نفسه. ومن الأمثلة على الدوال التي تنفذ بطريقة الـ ThreadPool البرمجة غير المتزامنة بالـ Socket والتي تبدأ بالعادة بكلمة Begin و End كمثال BeginConnect و BeginSend وغيرها وبالتالي جعل البرنامج ينفذ عملية ما بمصاداتها بـ Thread مؤقت وإكمال تنفيذ البرنامج بينما يقوم ذلك الـ Thread بإنهاء عملية التنفيذ وينتهي بانتهاء تنفيذ تلك العملية (انظر الفصل الرابع لمزيد من المعلومات).

يبين الشكل 2.4 كيف يتم تنفيذ برنامج للاتصال بـ Server Socket بالطريقة المتزامنة بحيث يبقى تنفيذ الدالة التالية معلق إلى حين الانتهاء من تنفيذ الدالة السابقة:



الشكل 2.4 ويبين كيفية عمل الاتصال المتزامن

ويبين الشكل 2.5 كيفية قيام الاتصال غير المتزامن بتنفيذ دوال الاتصال والإرسال و الاستقبال حيث يستمر تنفيذ الدوال التالية بدون الانتظار لإنهاء الدالة السابقة:



الشكل 2.5 ويبين كيفية عمل الاتصال غير المتزامن

الخلاصة:

يتبين لنا في هذا الفصل كيف انه يمكننا استخدام الـ Threading والبرمجة غير المتزامنة لتحسين أداء البرنامج وتنفيذ أكثر من عملية بنفس الوقت بدون أن تتأثر أي منها بتنفيذ الأخرى، كما وبينا الفرق بين استخدام كل من الـ Thread Class والـ ThreadPool وكيف أن الأخير يستخدم في الـ Asynchronous Socket Programming فيما يمكن استخدام الـ Thread Class مع الـ Synchronous Socket Methods لاستخدامها بدون أن يتأثر أداء النظام بحيث تنفذ كل عملية على Thread منفصل وبينا انه يمكننا من خلال الـ ThreadPool مناداة دالة ما بطريقة غير متزامنة ولكن مع تحكم قليل بكيفية تنفيذ الدالة في حين يمكننا الـ Thread Class من التحكم بشكل كامل بالـ Thread سواء بفترة تشغيله أو إنهائه أو أولوية معالجته.

سيتم الحديث في الفصل التالي عن كيفية برمجة الـ Synchronous Socket وباستخدام الـ Threading Class ثم في الفصل ألاحق سيتم شرح الـ Asynchronous Socket وكيفية تنفيذه من خلال الدوت نيت.