Chapter 2

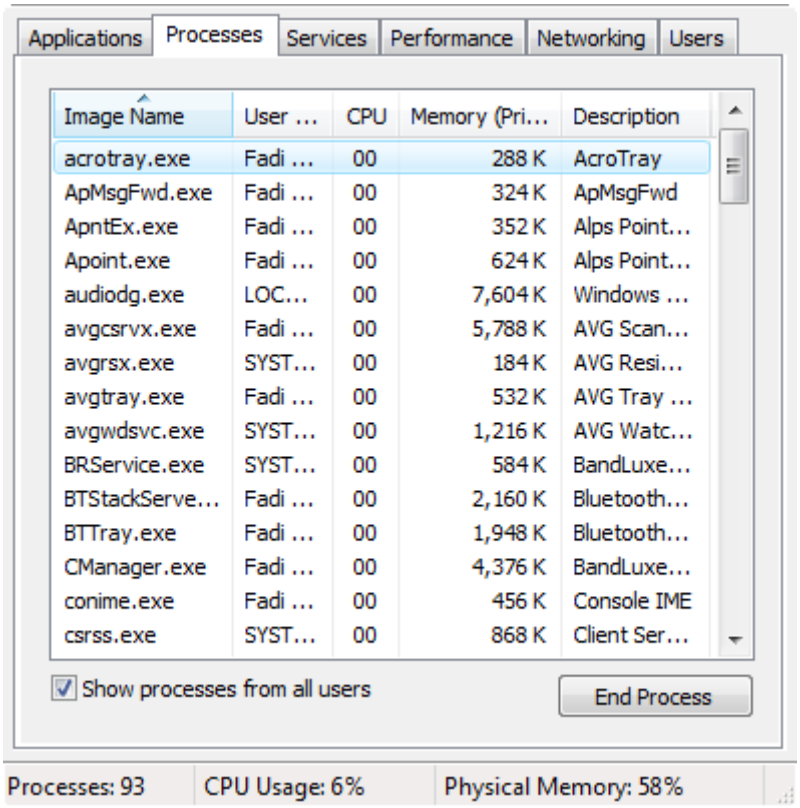
Introduction to Multithreading Programming

المواضيع الرئيسية في هذا الفصل:

1- المفهوم العام للـThreading والـMultithreading
2- مقدمة في برمجة الـThreading في الدوت نيت.
3- مقدمة في استخدام الـMultithreading في برمجة الشبكات.

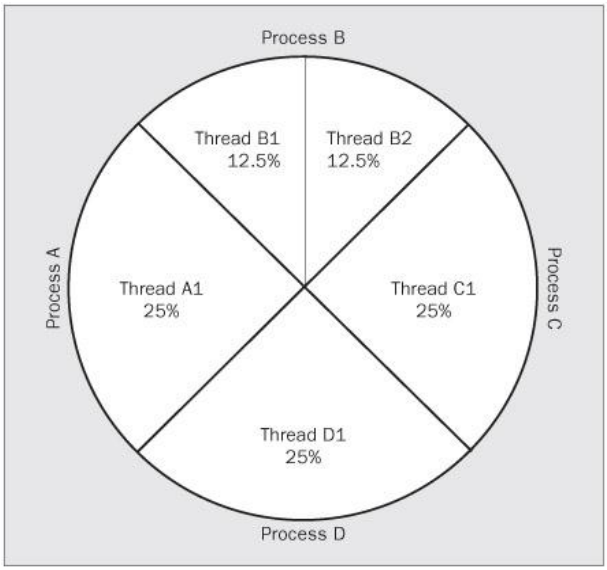
2.1: المفهوم العام للـThreading و الـMultithreading:

عزز مفهوم الـ Multithreading مفهوم الـ Multitasking في أنظمة التشغيل بحيث أن أي برنامج يمكنه أن يعمل على Thread منفصل عن الأخر 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 الـProcesses التي يقوم نظام التشغيل بإدارتها إثناء Runtime

لاحظ أن كل برنامج يحجز مقدار معين من وقت المعالج بناء على حاجته ويقوم نظام التشغيل بتقسيم المهام على المعالج وفق الحاجة و الأولويات ويبين الشكل 2.2 كيف يقسم المعالج دورة المعالجة على مجموعة من المهام والتي تحتوي على مجموعة من الـThreads 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.

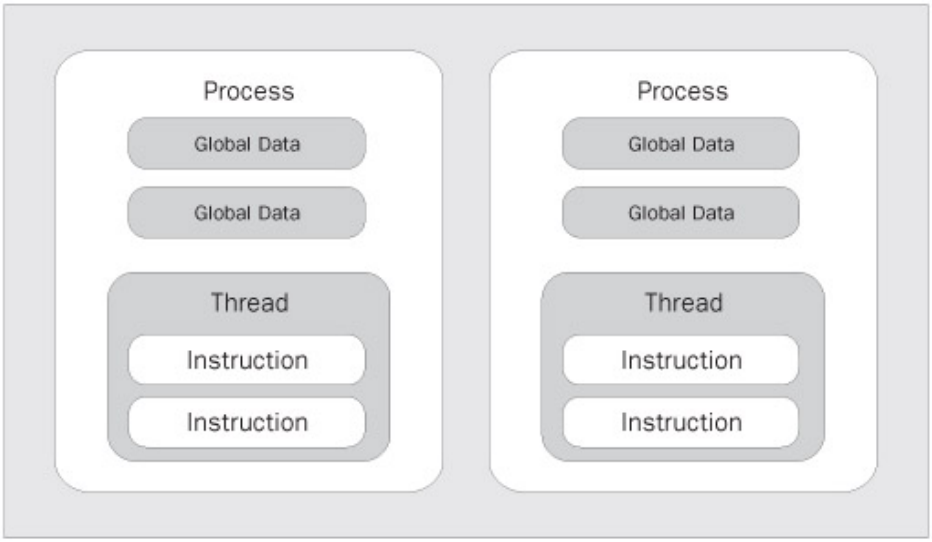
تنويه:

يهتم هذا الفصل بالجوانب المتعلقة بكيفية إدارة واستخدام الـMultithreading 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 فبإمكانك 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 الـThreading 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 بالـProcess وتسمى العمليات التي تعمل بداخله بشكل متوازي بالـThreads كعملية check spelling التي ذكرناها أنفاً لاحظ الشكل 2.3، ولا يخلو أي برنامج في الوقت الحالي من استخدام الـThreading ففيه يتمكن المبرمج من تنفيذ عدة عمليات بنفس الوقت ويعتبر استخدام الـThreading أيضا 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 الـProcess الواحد على أكثر من Thread

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

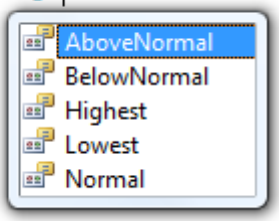
2.2.1: أهم الـClasses الموجود في System.Threading Namespace:

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 إثناء تنفيذ البرنامج بطريقة غير متزامنة على Thread متصل بالتطبيق بنفس مبدأ الـ Timer و ينتهي مع انتهاء تنفيذ البرنامج ويسمى في هذه الحالة بالـBackground Thread.

2.2.1.1: أهم الدوال Methods الموجودة في الـClass **Thread**:

Method	Description
Start	ويتم من خلاله بدأ تنفيذ الـThread وتحتوي هذه الدالة على إمكانية تمرير بارومیتر معين إلى Thread Method التي يتم مناداتها من خلال الـ Thread Delegate
Abort	ومن خلالها يمكن إيقاف عمل الـ Thread في أي وقت إثناء التشغيل Runtime Mode وتفريغ توابعه من الذاكرة.
Join	وفيها 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 بالمايكرو ثانية (كل ثانية تساوي 1000 مايكرو ثانية) .

2.2.1.2: أهم الخصائص Properties الموجودة في الـClass **Thread**:

Prosperity	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: `.Priority = ThreadPriority.` 

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

2.2.1.3: أهم Delegates التي تستخدم مع الـClass **Thread** والـ **ThreadPool**:

Delegate	Description
ParameterizedThreadStart	وتستخدم لمناداة الـMethod التي نود تنفيذها على Thread منفصل مع إمكانية تمرير بارومتر من نوع object إليها عند عمل Start لها.
ThreadStart	وتستخدم لمناداة الـMethod التي نود تنفيذه على Thread منفصل بشرط ألا تحتوي تلك الـMethod على أي بارومتر.
WaitCallback	وهي الوحيدة التي تستخدم مع الـThreadPool وتعمل على مناداة الـMethod التي نريد تنفيذها من خلال الـThreadPoolونستطيع من خلالها تمرير بارومتر من نوع Object للـMethod.

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

2.2.2: التركيب العام للـThreading Classes بالدوت نيت وكيفية استخدامه:

وأما فيما يخص طريقة استخدام الـThread والـThreadPool فلهما تركيب متشابه نسبيا غير أن الأخيرة لا تحتاج لعمل Start لتنفيذ الـThread كما أنها تعتبر Background Thread بشكل افتراضي بحيث تنتهي مع انتهاء عمل الـMain Thread وتتشابهان بكونهما يشترطان بأن تكون دالة الـThread 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_Thrading) );
        th.Start();
    }
    void Method_Needs_Thrading()
    {
        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_Thrading) );
        th.Start(startingvalue);
    }
    void Method_Needs_Thrading(object PassedValue)
    {
        bool infinity = true;
        int Count = (int)PassedValue;

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

ثانيا: 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_Thrading));
    }

    void Method_Needs_Thrading(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_Thrading),startingvalue);
    }

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

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

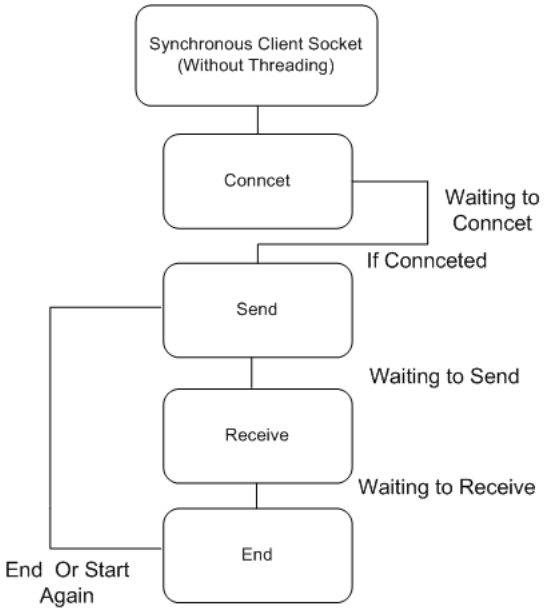
يوضح في كلا المثالين كيف انه يمكن تنفيذ دالة بطريقة غير متزامنة مع تنفيذ البرنامج بحيث أن تلك الدالة تحتوي على Infinity Loop وبالتالي فإنها ستبقى تعمل إلى الأبد ولو أننا لم نستخدم الـThreading لكان من الاستحالة عمل دالة شبيهة بتلك الدالة في برامجنا إذ أن تنفيذ تلك الدالة بالطريقة المعتادة سيمنع المعالج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

2.3: مقدمة في استخدام الـMultithreading في برمجة الشبكات:

ما يهمنا في هذا الفصل هو معرفة متى سنحتاج إلى استخدام الـThreading 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.

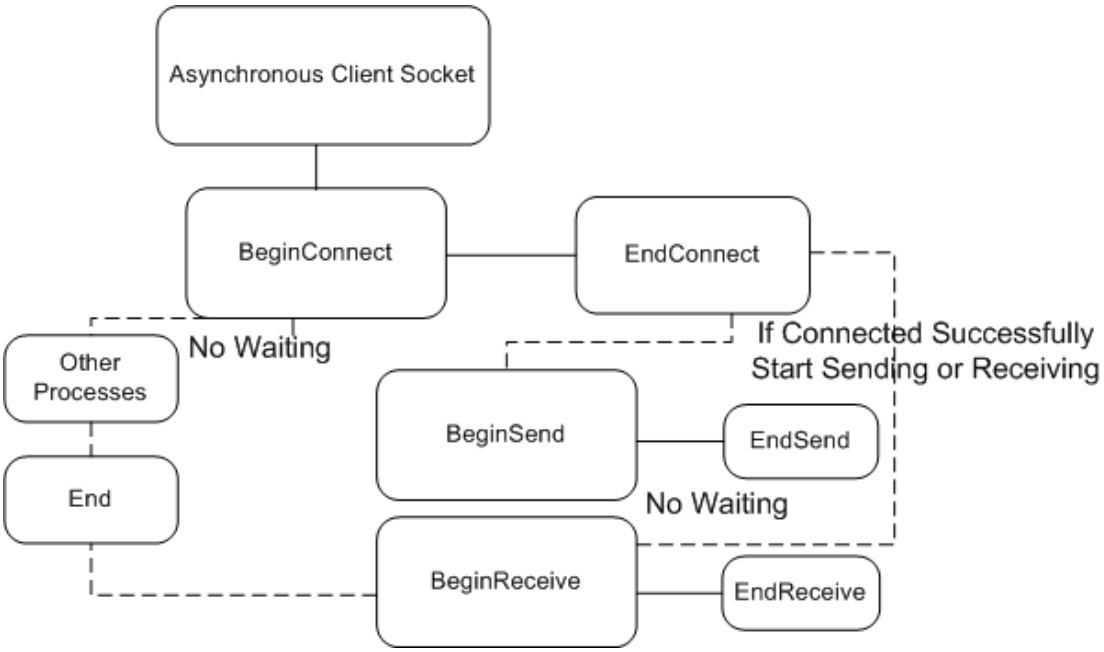
لاحظنا في الجزء السابق من هذا الفصل كيف أننا نستطيع مناداة دالة ما من خلال الـThread والـThreadPool وبالتالي تنفيذ تلك الدالة بطريقة غير متزامنة مع الـMain Thread (والمقصود به التطبيق نفسه) وبالتالي فإن مناداة أي دالة بطريقة غير متزامنة تسمح لتطبيق بإكمال تنفيذ بقية البرنامج مع استمرار تنفيذ تلك الدالة. ويعتبر استخدام الـThreadPool أكثر أمانا من استخدام الـThread Class والسبب أن الأخير قد يبقى في وضع التنفيذ حتى بعد إنهاء عمل البرنامج ولاستخدامه بشكل أكثر أمانا يفضل تفعيل الخاصية IsBackground وبالتالي جعله ينتهي مع إغلاق البرنامج ولكن استخدام الـThread Class يسمح لنا بالتحكم بشكل أكبر بتنفيذ دالة ما من حيث إمكانية تنفيذها أو إيقاف تنفيذها والتحكم بأولويتها للمعالجة إثناء تشغيل التطبيق كما أن تنفيذ مجموعة كبيرة من الـThreadPool على نفس التطبيق قد يؤدي إلى جعل التطبيق بطئ في تنفيذ مهامه على عكس تنفيذ مجموعة كبيرة من الدوال من خلال الـThread Class إذ لا يؤثر تنفيذها على التطبيق نفسه. ومن الأمثلة على الدوال التي تنفذ بطريقة الـThreadPool البرمجة غير المتزامنة بالـSocket والتي تبدأ بالعادة بكلمة Begin و End كمثال BeginConnect و BeginSend وغيرها وبالتالي جعل البرنامج ينفذ عملية ما بمناداتها بـ Thread مؤقت وإكمال تنفيذ البرنامج بينما يقوم ذلك الـThread 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 الـThreading 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 الـThread Class والـThreadPool وكيف أن الأخير يستخدم في الـ Asynchronous Socket Programming فيما يمكن استخدام الـThread Class مع الـSynchronous Socket Methods لاستخدامها بدون أن يتأثر أداء النظام بحيث تنفذ كل عملية على Thread منفصل وبينا انه يمكننا من خلال الـThreadPool مناداة دالة ما بطريقة غير متزامنة ولكن مع تحكم قليل بكيفية تنفيذ الدالة في حين يمكننا الـThread Class من التحكم بشكل كامل بالـThread سواء بفترة تشغيله أو إنهائه أو أولوية معالجته.

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