Multithreading Programming In C# and querying Northwind database

السلام عليكم ورحمة الله

الموضوع مهم جداً، وهناك بعض الأخوة طلبوا مني بعض الأمثلة توضيحة. لذلك سنقوم معاً بمناقشة صغيرة نفضي بعدها إلى مثال حي، يمكن أن نوظفه في أغلب تطبيقاتنا. وبالطبع نحن بحاجة له في الكثير من أجزاء أي مشروع.

مقدمة لا بد منها

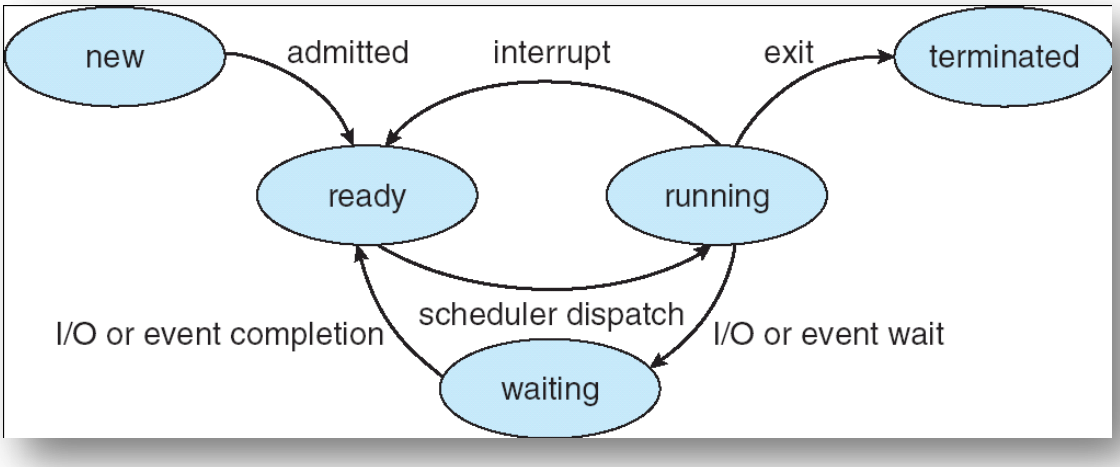
1. وحدة المعالجة المركزية تقوم بمهمة واحد فقط في نفس اللحظة. إذاً كيف يتم تشغيل أكثر من برنامج بنفس الوقت؟
2. نظام التشغيل يتعامل مع مجموعة كبيرة من البرامج ويقدم الكثير من الخدمات. إذاً كيف يتم التنسيق بين عمل هذه البرامج بحيث لا تتداخل مع بعضها أو تقوم بتعطيل عمل بعضها البعض؟
3. كيف يقوم برنامج ضخم نسبيا (كنظام التشغيل مثلاً) بعدة عمليات معقدة ومكلفة، ويبقى مع ذلك قادر على تنفيذ برامج أخرى، فيما برنامجي الصغير بمجرد قيامه بعمليه حسابية فإن الواجهات تتجمد بإنتظار انتهاء تلك العملية؟

"إذا لم تخطر ببالك الاسئلة السابقة، يجب عليك إعادة التفكير فيها"

الجواب لكل ما سبق هو بإستخدام Multithreading Programming.

ما هو الـ Thread حتى نفهم الـ Multithreading؟

أصغر وحدة تنفيذية في نظام التشغيل. كل مهمة أو عملية أو ربما برنامج، عبارة عن Thread، لذلك يكون لدينا مجموعة معينة من الـ Threads والتي تعمل بنفس الوقت ضمن بيئة نظام التشغيل، لكن يُسمح لـ Thread واحدة فقط في لحظة معينة بالدخول لـ CPU وتنفيذ كودها الخاص، وبالطبع استخدام موارد الحاسوب الأخرى مثل (الذاكرة، نظام الملفات، CPU، و...). وفي هذه الإثناء يمنع نظام التشغيل أي Thread أخرى من الدخول لـ CPU، بالتالي تعتبر في مرحلة انتظار Waiting، وبعد فترة معينة حتى وإن لم تنهي الـ Thread تنفيذها ضمن CPU يقوم نظام التشغيل بطردها، وجعلها في حالة انتظار، والسماح لأخرى بالدخول وتنفيذ جزء من كودها، وتتم هذه العملية باستمرار طوال فترة تشغيل النظام.



- New Thread: يتم وضعها بحالة جاهزية Ready حتى يسمح لها بالعمل Running

- Ready Thread: جاهزة لدخول CPU
- Waiting: بإنتظار حدث معين، إما I/O، أو مدة زمنية جديدة، كونها طويلة نسبيا.
- Terminated: تم إنهاءها، أما انتهى تنفيذها، او إلغاءها (تسمى عملية الإلغاء بالإجهاض Abort)

كيف يتم تنظيم عملية دخول وخروج الـ Threads إلى CPU؟

يقوم نظام التشغيل بهذه المهمة والتي تسمى (Scheduling) لذلك يسمى نظام التشغيل ب (Scheduler). ويختلف سلوك التعامل حسب عدة شروط مثل الأولوية، فيحدد عن طريق عدة خوارزميات مبنية ضمن نظام التشغيل منها FIFO، Round Roubin، FCFS.

"هناك المزيد من التفصيل المهم، ربما نتطرق إليه في موضوع قادم إن لزم الأمر"

ماذا نستنتج مما سبق؟

كلما زاد عدد Threads بتطبيق معين (زيادة بنسبة معينة) زادت سرعة تنفيذه، بالتالي حصلنا على سرعة أعلى، والأهم هو فصل العمليات الطويلة أو المعقدة نسبيا عن بعضها ضمن Thread خاص لكل عملية.

ماذا نسمي التطبيق في حال احتوى Thread واحدة فقط، وفي حال احتوى أكثر؟

أغلب التطبيقات تسمى Single Threaded لأنها تحتوي على Thread واحدة فقط، تكون مهمتها إدارة الواجهات، والقيام بالعمليات الحسابية الأخرى.

أما في حال أحتواء التطبيق على أكثر من Thread يسمى Multithreaded، حيث تسند لكل عملية طويلة نسبيا Thread خاصة بها.

هل من مثال واقعي للبرمجيات المتعددة المهام Multithreaded؟

نعم هناك مثال واقعي ومهم جداً، وهو الـ Visual Studio، عندما نقوم بكتابة برامجنا ضمن Visual Studio نلاحظ العديد من الخدمات قيد التنفيذ بنفس الوقت:

1. مدقق نحوي، يقوم بإكتشاف الإخطاء الإملائية الغير معروفة ضمن البرنامج حالياً (إسم متغير غير معرّف).
2. مدقق نوع البيانات، يمكنه اكتشاف إسناد قيمة نصية لمتغير عددي مثلاً.
3. قائمة الخيارات التي تظهر بمجرد الكتابة (IntelliSence)

كل عملية هي عبارة عن Thread مستقل يقوم بعمليته الخاصة، وجميعها تتشارك بنفس المصارد، مثلا الكود المصدري الذي تكتبه يعتبر متاحاً لجميعها، بالتالي تستطيع أي واحدة العمل على أساسه.

"تخيل لو كان برنامج ضخم مثل Visual Studio يقوم على أساس Thread واحدة، كم ستضطر للإنتظار عند كل حرف تكتبه! "

مثالنا

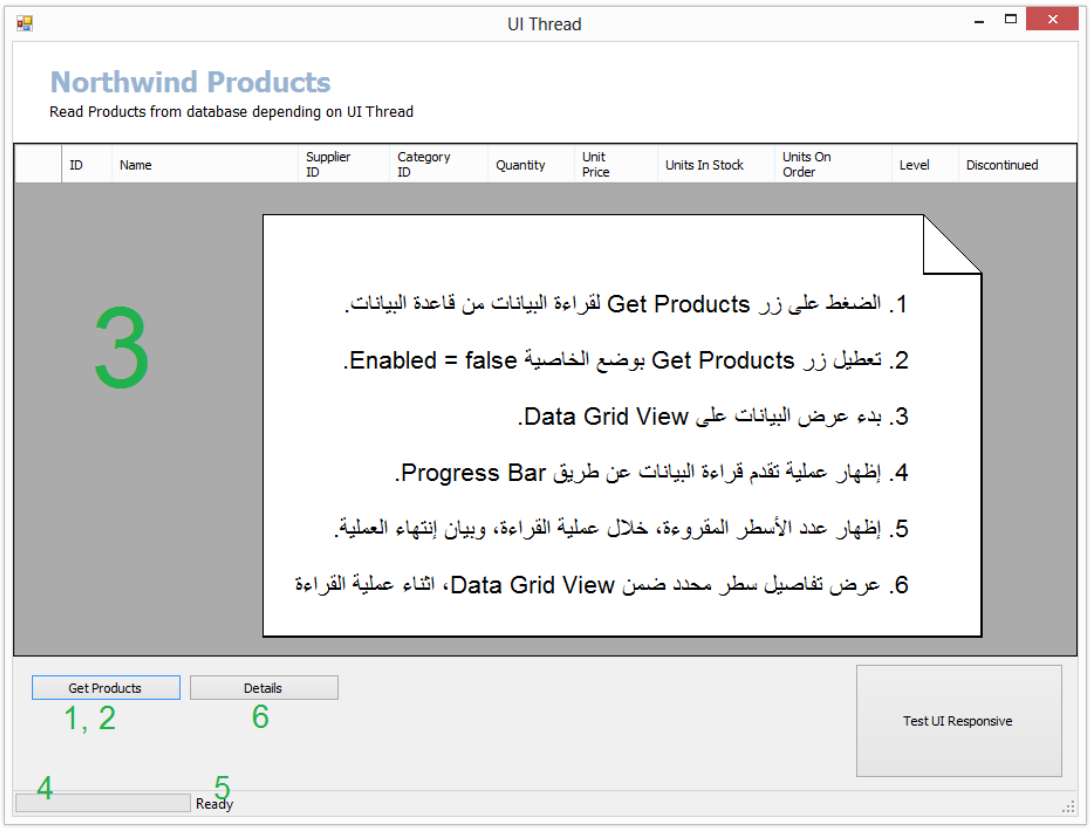
سنقوم بعمل برنامج يقوم بقراءة بيانات جدول Products ضمن قاعدة بيانات Northwind العالمية.

1. سنقوم بعمل قسم Single Threaded يقوم بقراءة البيانات، وعرضها على الواجهة، هنا ستتم عمليه القراءة من قاعدة البيانات، والعرض على الواجهة عن طريق Thread واحدة فقط، هي UI Thread.
2. سنقوم بعمل قسم Multithreaded يقوم بقراءة البيانات عن طريق Thread خاصة، ويقوم بعرض البيانات على الواجهة عن طريق Thread أخرى، هي UI Thread.

في كلتا الحالتين، سنقوم بعمل سلوك معين لإختبار فعالية الواجهات خلال مرحلة قراءة البيانات من قاعدة البيانات، حيث سنقوم بمحاولة عرض تفاصيل Product تم قراءته حديثاً وما زالت عملية القراءة مستمرة.

ما هو شكل الواجهة والسلوك المفترض تنفيذه في مثالنا؟

الآن لنشاهد الشكل التالي ونحاول فهم سلوك البرنامج، خلال فترة التنفيذ.



ما هي متطلبات هذا العمل؟

- نحن بحاجة لتعريف Product Class ليسهل علينا التخاطب مع قاعدة البيانات، لذلك سنقوم بكتابة هذا الـ Class بالشكل التالي، والذي يعكس حقول الجدول ضمن قاعدة البيانات.

```
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int SupplierId { get; set; }
    public int CategoryId { get; set; }
    public string Quantity { get; set; }
    public decimal UnitPrice { get; set; }
    public int UnitsInStock { get; set; }
    public int UnitsOnOrder { get; set; }
    public int Level { get; set; }
    public bool Discontinued { get; set; }

    public Product() { }

    public Product(int id, string name, int supplierId,
        int categoryId, string quantity, decimal unitPrice,
```

```
        int unitsInStock, int unitsOnOrder, int level, bool discontinued)
    {
        Id = id;
        Name = name;
        SupplierId = supplierId;
        CategoryId = categoryId;
        Quantity = quantity;
        UnitPrice = unitPrice;
        UnitsInStock = unitsInStock;
        UnitsOnOrder = unitsOnOrder;
        Level = level;
        Discontinued = discontinued;
    }
}
```

- نحن بحاجة أيضاً لكتابة الكود المسؤول عن الإتصال بقاعدة البيانات، وقراءة بيانات الجدول، وتشكيل Product Object من كل سطر مقروء.

```
SqlConnection con = new SqlConnection(@"Server = localhost; Database =
Northwind; Integrated Security = true;");

    SqlCommand cmd = new SqlCommand("Select * from Products", con);
    con.Open();
    SqlDataReader rdr = cmd.ExecuteReader();
    while (rdr.Read())
    {
        int id = rdr.GetInt32(0);
        string name = rdr.GetString(1);
        int supplierId = rdr.GetInt32(2);
        int categoryId = rdr.GetInt32(3);
        string quantity = rdr.GetString(4);
        decimal unitPrice = rdr.GetDecimal(5);
        int unitsInStock = rdr.GetInt16(6);
        int unitsOnOrder = rdr.GetInt16(7);
        int level = rdr.GetInt16(8);
        bool discontinued = rdr.GetBoolean(9);

        Product p = new Product(id, name, supplierId, categoryId, quantity,
unitPrice, unitsInStock, unitsOnOrder, level, discontinued);

        // 1. Display the row in Data Grid View
        // 2. Make the Progress Bar walk a step
        // 3. Display the row count
    }
    con.Close();
```

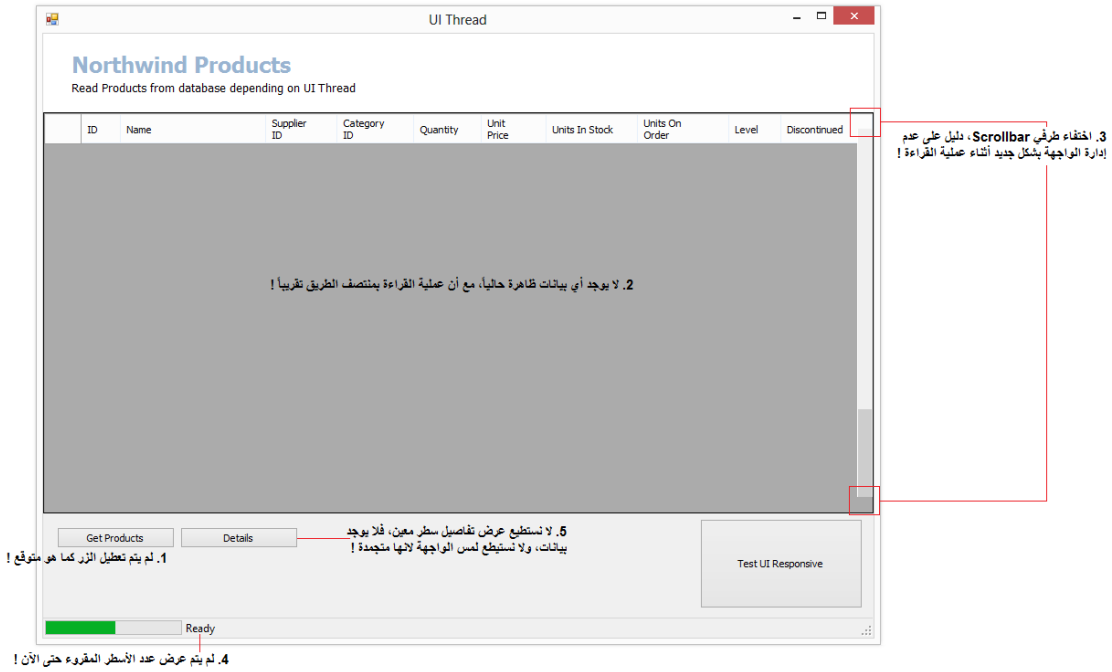
الآن عند الضغط على زر Get Products يجب حصول السيناريو التالي:

1. تعطيل الزر
2. قراء البيانات من قاعدة البيانات (مع عرض تفاصيل عملية القراءة)
3. إعادة تفعيل الزر
4. عرض كلمة Done والتي تشير لإنتهاء عملية القراءة.

```
private void btnGetProducts_Click(object sender, EventArgs e)
{
    btnGetProducts.Enabled = false;
    DoWork();
    btnGetProducts.Enabled = true;
    lblStatus.Text = "Done...";
}
```

ماذا يحصل في حالة قراءة البيانات والتعديل على الواجهات، بالطريقة المعروفة، والتي تستخدم Thread واحدة فقط، هي UI Thread؟

سنقوم ضمن القسمين بتبطيء العمل قليلاً، لكي يتسنى لنا ملاحظة الأداء، سيتم ذلك عن طريق استخدام.



يمكنك قراءة الكود البسيط لهذه الواجهة، وملاحظة عدم تطابق التنفيذ مع الشيء المكتوب.

ما هو الحل للحصول على الأداء المطلوب؟

الحل طبعاً بإستخدام Multithreading Programming.

تقدم .NET Framework مجموعة خدمات متنوعة لعمل Multithreading Application، وقد طورت حديثاً مجموعة جديدة من المفاهيم.

سنقوم بإستخدام مفهوم بسيط نسبياً هو Background Worker، وهو عبارة عن Thread جديدة، يسند لها مهمة خاصة، تقوم بتنفيذها بدون التأثير على عمل أجزاء البرنامج الأخرى، كالواجهات مثلاً.

ما هي مراحل العمل مع Background Worker؟

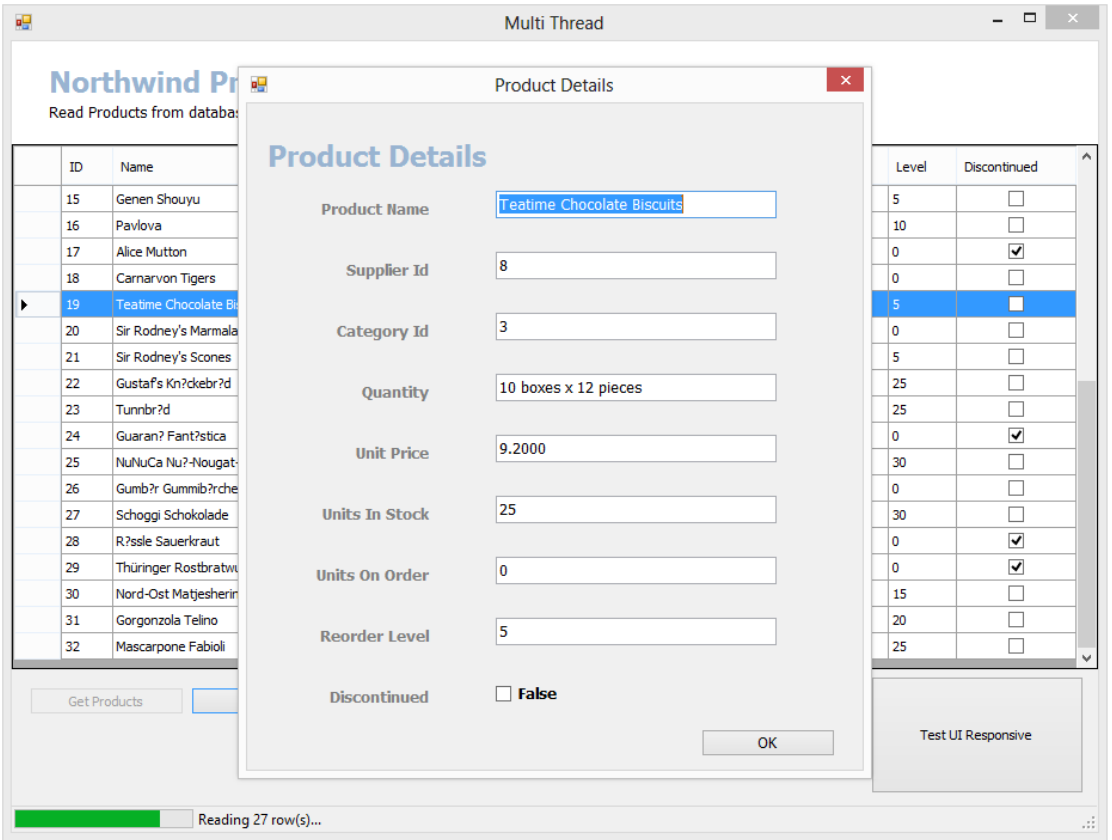
لدينا ثلاث مفاهيم أساسية ضمن Background Wroker تعطينا القدرة على التحكم بأداء البرنامج أثناء التنفيذ:

1. DoWork وهي خاصة بتنفيذ الكود المسند إليها، غالباً يكون عبارة عن عمليات طويلة أو معقدة نسبياً تحتاج إلى وقت معين.
2. Progress Changed وهي خاصة بعمل تغييرات مرحلية خلال مرحلة التنفيذ، مثلاً عرض السطر الذي تم قراءته من قاعدة البيانات.
3. RunWorkerCompleted وهي خاصة بمرحلة إنتهاء التنفيذ، ويمكننا خلال هذه المرحلة عمل ما يناسب إنتهاء التنفيذ، كأن نظهر نص أنه تم الإنتهاء.

ملاحظة مهمة جداً جداً.

كل Thread تستطيع مشاركة مصادر البرنامج ككل، ولكن لا تستطيع أي Thread التعامل مع مصادر Thread أخرى، مثلاً لا يمكن لـ Thread القراءة من قاعدة البيانات التعديل على الواجهة بأي شكل، لأن ذلك محصور على UI Thread، ولكن تقوم بإرسال بيانات معينة وبتنبيه UI Thread لكي تقوم الأخيرة بإجراء التعديلات المطلوبة.

ماذا يحصل في حالة قراءة البيانات عن طريق Thread خاصة، وتعديل الواجهة عن طريق UI Thread؟



كل تفاصيل السلوك المطلوبة والتي لا تتم بالشكل المطلوب تمت هنا تماماً كما نريد.

ملاحظات في هذه المرحلة

1. عند الضغط على زر Get Products، نقوم بتفعيل عمل الـ Thread الجديدة، والتي بدورها ستقوم ببدء الاتصال بقاعدة البيانات والقراءة منها، بشكل منفصل عن الواجهة، لذلك لا تزال الواجهة تعمل بشكل جيد، وعند قراءة كل سطر تقوم بتنبيه الواجهة لتقوم بالتعديل المطلوب على عناصر الواجهة.

2. نستطيع التعديل على الواجهات (أو أي Thread أخرى) ضمن Thread اخرى بأكثر من طريقة: مثلاً مع Background Worker نستطيع التعديل بطريقتين
   - عن طريق الميثود ProgressChanged، حيث نقوم بإرسال البيانات المطلوبة ضمن ProgressChangedEventArgs، مع النسبة المؤية لتقدم العمل Progress Percentage.
   - عن طريق استدعاء مباشر لUI Trehad . بواسطة الميثود Invoke أو BeginInvoke. بالنسبة لـ WPF هناك مفهوم جديد هو Dispatcher، على العموم هناك عدة طرق لتطبيق مفهوم Multithreading ضمن تطبيقاتنا، ربما نقوم بعرضه في مقالات قادمة إذا اقتضى الأمر.

3. عملية تحديث Progress Bar. عملية التحديث التقليدية هي بتعريف مجال عمل Progress Bar عن طريق القيمتين Minimum, Maximum وتحديد مقدار كل خطوة Step، وعند تنفيذ عملية معينة نقوم بالمشي خطوة للأمام باستدعاء الميثود ()PerformStep. لكن عندما نقوم بقراءة بيانات من قاعدة البيانات، أو حتى بشكل عام، فإن عدد العمليات غير معروف بالضبط، مما يسبب مشكلة في حال تجاوزنا قيمة Maximum. لذلك ومن وجهة نظر شخصية في مثل هذه الحالات، كما في مثالنا قمت بمعالجة المشكلة بالشكل التالي:

افترضت قيمة Maximum مبدئيا صفر، وعند تنفيذ عملية معينة نقوم بزيادة قيمة Maximum وبتنفيذ خطوة للأمام progressBar.Value +=1.
هناك طرق أخرى تقوم بحساب النسبة المؤية حسب قيم محددة وتحديث الـ Progress Bar، ربما نتطرق اليها في مقالات قادمة بإذن الله تعالى.

طارق جهاد الجباوي
tareqjehad@yahoo.com
2014