Asynchronous Programming

لو اشتغلت على تطبيقات C# قبل كده، أكيد لاحظت إن بعض العمليات زي جلب البيانات من قاعدة بيانات أو الاتصال بشبكة بتأخذ وقت طويل وبتوقف البرنامج كله! ده اللي بنسميه synchronous programming، وهو مشكلة كبيرة خاصة في التطبيقات اللي فيها واجهة مستخدم (UI). بس الحل جاهز: الـ asynchronous programming! يلا نفهمها مع بعض بطريقة بسيطة، ونشوف إزاي هي بتحسن الأداء مقارنة بالطرق القديمة.

1. إيه هي البرمجة المتزامنة (Synchronous) ومشاكلها؟

في البرمجة المتزامنة، العمليات بتتنفذ خطوة بخطوة، زي طابور في البنك: اللي جاي أول يخلص أول. لو عملية واحدة أخذت وقت (مثل قراءة بيانات من الداتابيز)، البرنامج كله هيوقف لحد ما تخلص.

- **مثال بسيط في C#:**

```
private void LoadData() {
    SqlConnection conn = new SqlConnection("connection string");
    conn.Open();
    // Execute query and bind data
    // ... (كود جلب البيانات)
    conn.Close();
}
```

هنا، لو الـ Open() أو الاستعلام أخذ 5 ثواني، الـ UI هيتجمد تمامًا! ده بيؤدي لتجربة مستخدم سيئة، خاصة في تطبيقات WPF أو Web API حيث الاستجابة مهمة.

- **المشكلة الكبيرة**: في الـ UI، البرنامج بيشتغل على thread واحد، فأي تأخير بيوقف كل حاجة. تخيل مستخدم يحاول يضغط في مكان تاني ومش بيحصل حاجة عشان الويبسايت واقف!

2. إيه البرمجة الغير متزامنة (Asynchronous) وفوائدها؟

الـ asynchronous programming بتسمح بتنفيذ العمليات بشكل موازي، بدون انتظار. يعني البرنامج يبدأ العملية ويستمر في شغله التاني، ويتلقى النتيجة لما تخلص.

- **فوائد رئيسية**:

- **تحسين الاستجابة**: الـ UI يبقى responsive، المستخدم يقدر يتفاعل مع التطبيق أثناء الانتظار.
- **أداء أفضل**: خاصة في عمليات I/O زي الداتابيز أو الشبكة، بحيث الانتظار مش بيستهلك CPU.
- **قابلية التوسع**: في Web API، السيرفر يقدر يتعامل مع طلبات أكتر بدون توقف.

- **مثال بسيط**: لو عايز تجيب بيانات بدون توقيف الـ UI، استخدم threads أو async.

3. الأنماط المختلفة للـ Asynchronous في C#

في C#، فيه طرق كتير لعمل asynchronous، وهي تطورت مع الوقت. خلينا نشوفها خطوة بخطوة:

- **Asynchronous Programming Model**

- بتعتمد على زوج من الدوال: BeginMethod و EndMethod.
- **مثال:** BeginInvoke و EndInvoke في الـ delegates.
- عيب: معقدة في الكتابة.

- **Event-Based Asynchronous Pattern**

- بتعتمد على دالة Async وحدث Completed.
- **مثال:** BackgroundWorker.RunWorkerAsync و RunWorkerCompleted.
- أفضل شوية، بس لسة تحتاج plumbing كتير.

- **Task-Based Asynchronous Pattern**

- من .NET 4.0، بتعتمد على Task من System.Threading.Tasks.
- **مثال:** Task.Run() لتشغيل عملية في الخلفية.

- أسهل، وبتدعم parallelism.

🔹 **async/await في C# 5.0:**

- الثورة الحقيقية! الكود يبدو زي synchronous، بس asynchronous تحت الغطاء.
- **كيفية الاستخدام**: أضف async أمام الدالة، و await أمام العملية البطيئة.
- **مثال:**

```
private async void LoadDataAsync() {
    using (var conn = new SqlConnection("connection string")) {
        await conn.OpenAsync();
        // Execute query asynchronously
        var reader = await cmd.ExecuteReaderAsync();
        // Bind data
    }
}
```

هنا، الـ await يرجع التحكم فورًا، والـ UI يستمر responsive!

4. التحسينات على الطرق القديمة

قبل C# 5.0، كان asynchronous معقد: تحتاج تكتب callbacks أو events، وده يجعل الكود صعب القراءة والصيانة. مع async/await، الكود يبقى طبيعي ومنظم، بدون حاجة لـ plumbing يدوي.

🔹 **في ASP.NET Core Web API**

- **synchronous:** السيرفر ينتظر كل طلب، فلو طلب أخذ وقت، الآخرين يتأخروا.
- **asynchronous:** استخدم async/await لعمليات الداتابيز.
- **مثال في Repository:**

```
public async Task<User> GetUserByIdAsync(int id) {
    return await _dbContext.Users.FirstOrDefaultAsync(u => u.Id ==
id);
}
```

في الـ Controller:

```
[HttpGet("{id}")]
public async Task<IActionResult> GetUserByIdAsync(int id) {
    var user = await _userRepository.GetUserByIdAsync(id);
    return Ok(user);
}
```

ده يحسن الـ scalability، خاصة مع طلبات كتير.

- 🔹 **مقارنة سريعة**:

- قديم (Synchronous): بطيء، يوقف الـ UI.
- جديد (Async): سريع، responsive، سهل الكتابة.

5. نصائح مهمة للتعامل مع Asynchronous Programming

✔️ **استخدم async/await دائمًا لعمليات I/O**: زي الداتابيز أو HTTP calls.
✔️ **تجنب await داخل lock**: ممكن يسبب deadlocks.
✔️ **لا تستخدم async في Main أو Constructors**: مش مدعوم.
✔️ **تعامل مع الاستثناءات**: استخدم try/catch حول await.
✔️ **اختبر الأداء**: استخدم tools زي Performance Profiler في VS.