Think Async

Sunday June 9, 2024

Usually, when writing asynchronous code, benefits are incremental. It can be difficult to see significant results at the micro level, unless you're willing to dig in with profiling tools. Sometimes, though, benefits are obvious. I love these, because they provide real motivation for me to take the time to write better code.

A while back, I wrote a simple application to get unread mail counts for a list of accounts. I originally wrote it in Python, but I had some issues with it not always playing nice when scheduled in Conky. I rewrote it as a console application in C#, and got much better stability.

Originally, it was structured as follows. Information about an account is populated in a MailEntry class:

```
public class MailEntry
{
    public MailEntry(string hostUrl, string userName, string password)
    {
        HostUrl = hostUrl;
        UserName = userName;
        Password = password;
    }

    public string? HostUrl { get; set; }
    public string? UserName { get; set; }
    public string? Password { get; set; }
}
```

I store my account info in appsettings, and populate a list:

```
List<MailEntry> mailEntries = new();

// code to retrieve from appsettings goes here.
```

Then, I iterate through the list and use my MailHandler code to do the actual checking and reporting of unread counts:

```
foreach (MailEntry mailEntry in mailEntries)
{
    mailHandler.CheckMail(
        mailEntry.HostUrl ?? "",
        mailEntry.UserName ?? "",
        mailEntry.Password ?? ""
    );
}
```

But, you'll notice that this code runs in a linear fashion. As in:

1. Make a call to check an account.
2. Wait for that call to complete.
3. Move to the next item in the list.
4. Repeat.

I have 10 accounts that I'm checking, so this is not efficient at all. It wasn't a big deal, but it started to bother me that it was taking 10 - 15 seconds to complete on each run. I decided to do some refactoring to support asynchronicity. First, I changed Main():

```
private static async Task Main(string[] args)
```

Then, I created a list to hold information about the state of each running task:

```
List<Task> tasks = new();
```

I changed the CheckMail method in MailHandler:

```
public async Task CheckMail(string hostUrl, string userName, string
password)
```

I checked each method I'm calling in the IMAP library I'm using to see which ones have an asynch version, and updated them accordingly:

```
await client.AuthenticateAsync(userName, password);

await client.Inbox.OpenAsync(FolderAccess.ReadOnly);
```

Back in Main(), the check loop is updated to add each async task to the list:

```
foreach (MailEntry mailEntry in mailEntries)
{
    tasks.Add(
        mailHandler.CheckMail(
            mailEntry.HostUrl ?? "",
            mailEntry.UserName ?? "",
            mailEntry.Password ?? ""
        )
    );
}
```

And then I wait for all the tasks to complete before proceeding:

```
await Task.WhenAll(tasks);
```

My application now runs in 2 - 4 seconds.  The takeaway here is that it's absolutely worth your time to look carefully at repeating logic in your code and optimize it to run asynchronously, if you can.

Full code is here: https://github.com/jfcarr/unread-mail-count