Programmazione asincrona

Programmazione asincrona

Il modello di programmazione asincrona (TAP) fornisce un'astrazione sul codice asincrono.

Il codice sembrerà scritto in maniera sequenziale ma il compilatore apporrà le dovute trasformazioni per farlo diventare asincrono.

L'obiettivo di questa sintassi consiste nell'abilitare un codice che viene letto come una sequenza di istruzioni ma viene eseguito in un ordine più complesso in base all'allocazione delle risorse esterne e al completamento dell'attività.

Programmazione asincrona VS concorrente

Sono due cose diverse:

- la programmazione concorrente (o parallela) richiede più thread al lavoro nello stesso momento
- nella programmazione asincrona il thread è uno solo, che divide il suo operato tra più attività che stanno andando avanti nello stesso momento.

Esempio della colazione (asincrona)

La preparazione della colazione è un buon esempio di lavoro asincrono non parallelo. Tutte le attività possono essere gestite da una sola persona (o thread).
Una sola persona può preparare la colazione in modo asincrono iniziando l'attività successiva prima che l'attività precedente venga completata. La preparazione procede indipendentemente dal fatto che venga controllata da qualcuno.
Non appena si inizia a scaldare la padella per le uova, è possibile iniziare a friggere la pancetta. Dopo aver iniziato a cuocere la pancetta, è possibile inserire il pane nel tostapane.

Esempio della colazione (concorrente)

In un algoritmo parallelo sarebbero necessari più cuochi (o thread). Un cuoco cucinerebbe le uova, un cuoco cucinerebbe la pancetta e così via. Ogni cuoco si dedicherebbe a una singola attività.

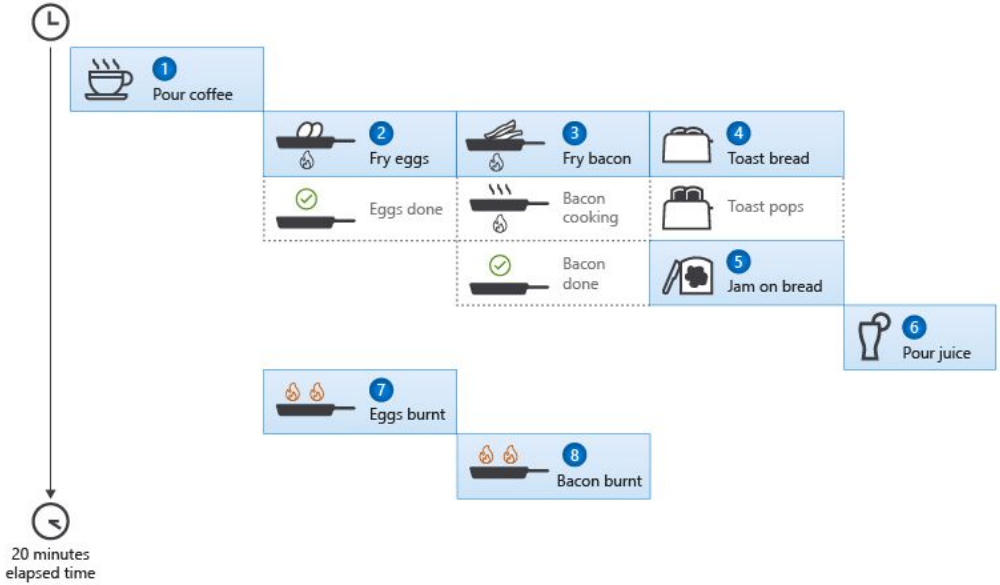
Ogni cuoco (o thread) verrebbe bloccato in modo sincrono in attesa che la pancetta sia pronta per essere girata o che la tostatura del pane venga completata.

Algoritmo sincrono sequenziale

La colazione preparata in modo sincrono richiede circa 30 minuti perché il totale è la somma di ogni singola attività.
Il computer si **bloccherà** in corrispondenza di ogni **istruzione** fino a quando non verrà completata prima di passare all'istruzione successiva. In questo modo non verrà preparata una colazione soddisfacente. Le attività successive non verranno iniziate prima del completamento delle attività precedenti. La preparazione della colazione richiederà **più tempo** e alcuni alimenti si raffredderanno prima di essere serviti.

```
Coffee cup = PourCoffee();
Console.WriteLine("coffee is ready");

Egg eggs = FryEggs(2);
Console.WriteLine("eggs are ready");

Bacon bacon = FryBacon(3);
Console.WriteLine("bacon is ready");

Toast toast = ToastBread(2);
ApplyButter(toast);
ApplyJam(toast);
Console.WriteLine("toast is ready");

Juice oj = PourOJ();
Console.WriteLine("oj is ready");
Console.WriteLine("Breakfast is ready!");
```

Non bloccare, ma attendere




```
static async Task Main(string[] args)
{
    Coffee cup = PourCoffee();
    Console.WriteLine("coffee is ready");

    Egg eggs = await FryEggsAsync(2);
    Console.WriteLine("eggs are ready");

    Bacon bacon = await FryBaconAsync(3);
    Console.WriteLine("bacon is ready");

    Toast toast = await ToastBreadAsync(2);
    ApplyButter(toast);
    ApplyJam(toast);
    Console.WriteLine("toast is ready");

    Juice oj = PourOJ();
    Console.WriteLine("oj is ready");
    Console.WriteLine("Breakfast is ready!");
}
```

Metodo asincrono

```
static async Task<Toast> MakeToastWithButterAndJamAsync(int number)
{
    var toast = await ToastBreadAsync(number);
    ApplyButter(toast);
    ApplyJam(toast);

    return toast;
}
```