Programowanie współbieżne i wielowątkowe w .NET

Paweł Biesiada
pawel.piotr.biesiada@gmail.com

Plan zajęć

- Wprowadzenie
  - Programowanie równoległe i asynchroniczne
  - Wady i zalety
  - Kiedy warto rozproszyć zadanie
- Wątki (*System.Threading.Thread*)
  - Czym są wątki
  - Praca z wątkami, sekcje krytyczne
- Synchronizacja wątków
  - Wykorzystanie klas z biblioteki Thread(+Interlocked)
  - *Lock*

Plan zajęć

- Zadania (*System.Threading.Tasks.Task*)
  - Wątek vs Zadanie
  - Fabryka zadań (*TaskFactory*)
  - Stan zadań, przerywanie działania
  - *async await*
- Klasa Parallel (*System.Threading.Tasks.Parallel*)
  - Opis metod F*or*, F*oreach*, I*nvoke*
  - *ParallelOptions*
- PLINQ (*System.Linq.ParallelEnumerable*)
  - Czym jest PLINQ
  - Wady i zalety
  - Metoda *ForAll*

Plan zajęć

- Wątki w GUI
  - WPF, UWP, Windows Forms
- Reactive Extensions
- Kolekcje bezpieczne wątkowo
- Wzorce projektowe
  - Synchronizacji, współbieżności, inicjalizacji
- Klasyczne problemy wielowątkowe
  - Biblioteka
  - Producent konsument
  - Deadlock
- Podsumowanie

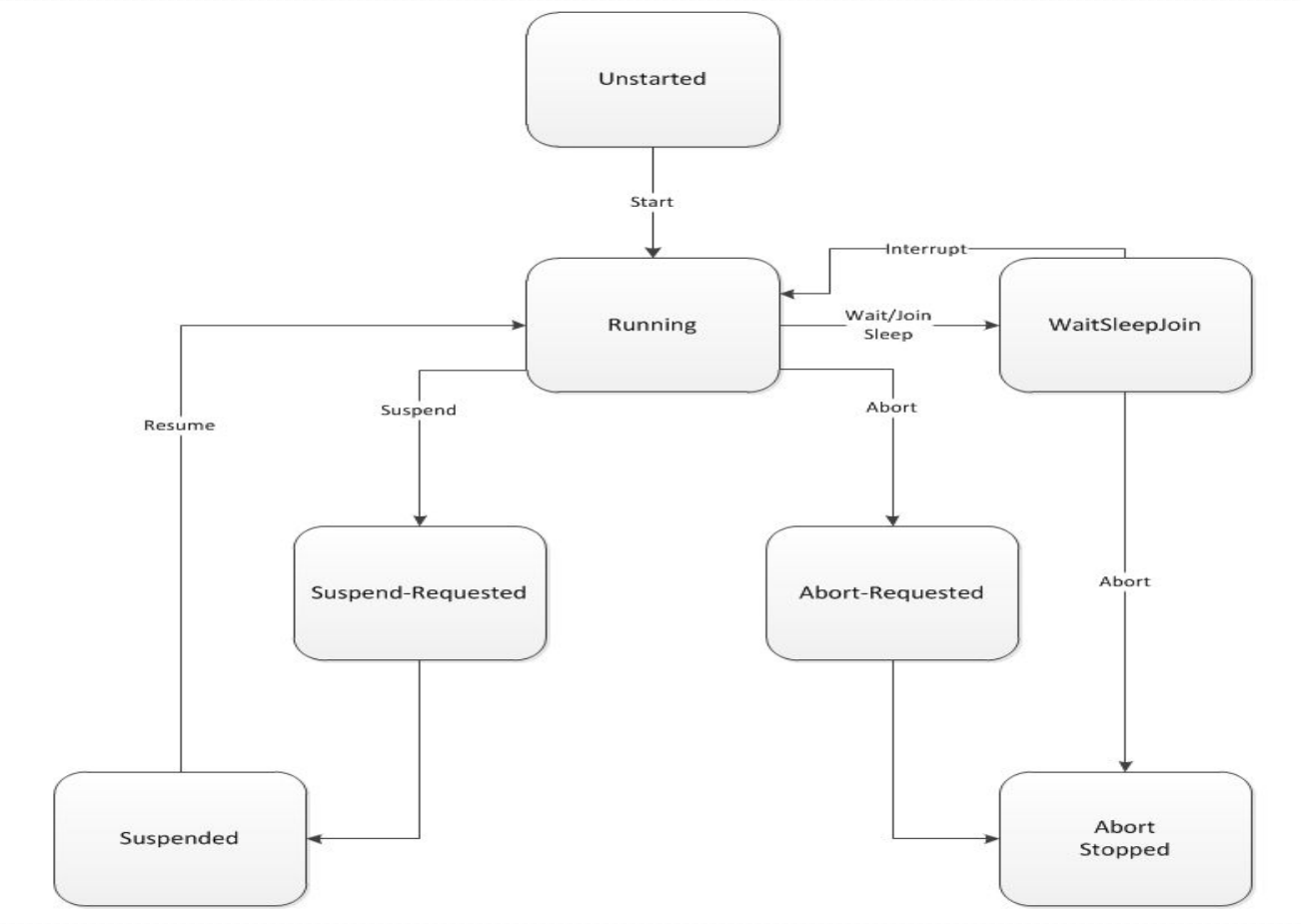
Wprowadzenie

- Programowanie równoległe
  - Rozdzielenie zadań *(robienie kilku rzeczy naraz na różnych wątkach)*
  - Wielowątkowość *(forma współbieżności)*
- Programowanie asynchroniczne
  - Obiekty typu *future (obietnica wykonania zadania w przyszłości – Task : z void, Task<> z wartością zwrotną)*
- *Programowanie reaktywne*
  - Reactive Extensions (Rx) *(programowanie na zdarzeniach)*
- Wady i zalety
- Kiedy NIE stosować

Wątki

- Czym są wątki *(wątek vs proces, stare)*
- Przestrzeń nazw *System.Threading*
- Klasa *System.Threading.Thread*
  - Utworzenie
  - Uśpienie
  - Przerwanie
- sekcje krytyczne

Stany wątków



Synchronizacja wątków

- *Join/Abort/Sleep* – zmiana stanów wątku
- *ThreadPool* - Klasa do zarządzania wątkami
- *Interlocked* – synchronizacja operacji dodaj/odejmij
- *Monitor* – Metody są bezpieczne
  - *lock* – dekorator syntaktyczny metod Enter i Exit
  - *Pulse/Wait* – usypia lub sygnalizuje do wątku
- *Semafor* – ogranicza maksymalną liczbę wątków w SK
  - *Mutex* – Semafor binarny
- *Bariera* – Synchronizacja wątków podczas

Zadania

- Wątek vs Zadanie
- Klasy *Task* i *Task<>*
  - *Wykorzystanie*
  - *Synchronizacja*
  - *async await*
- Fabryka zadań (*TaskFactory*)
- Stan zadań, przerywanie działania

Stany zadań

Status	Description
Canceled	The task acknowledged cancellation by throwing an OperationCanceledException with its own CancellationToken while the token was in signaled state, or the task's CancellationToken was already signaled before the task started executing.
Created	The task has been initialized but has not yet been scheduled.
Faulted	The task completed due to an unhandled exception.
RanToCompletion	The task completed execution successfully.
Running	The task is running but has not yet completed.
WaitingForActivation	The task is waiting to be activated and scheduled internally by the .NET Framework infrastructure.
WaitingForChildrenToComplete	The task has finished executing and is implicitly waiting for attached child tasks to complete.
WaitingToRun	The task has been scheduled for execution but has not yet begun executing.

Źródło: https://msdn.microsoft.com/en-us/library/system.threading.tasks.taskstatus(v=vs.110).aspx

Wzorce programowania asynchronicznego

- APM - .NET 1.1
  - *BeginInvoke/EndInvoke*
- EAP - .NET 2.0
  - Events & *BackgroundWorker*
- TAP - .NET 4.5
  - *async/await Task, Task<>* & *CancellationToken*

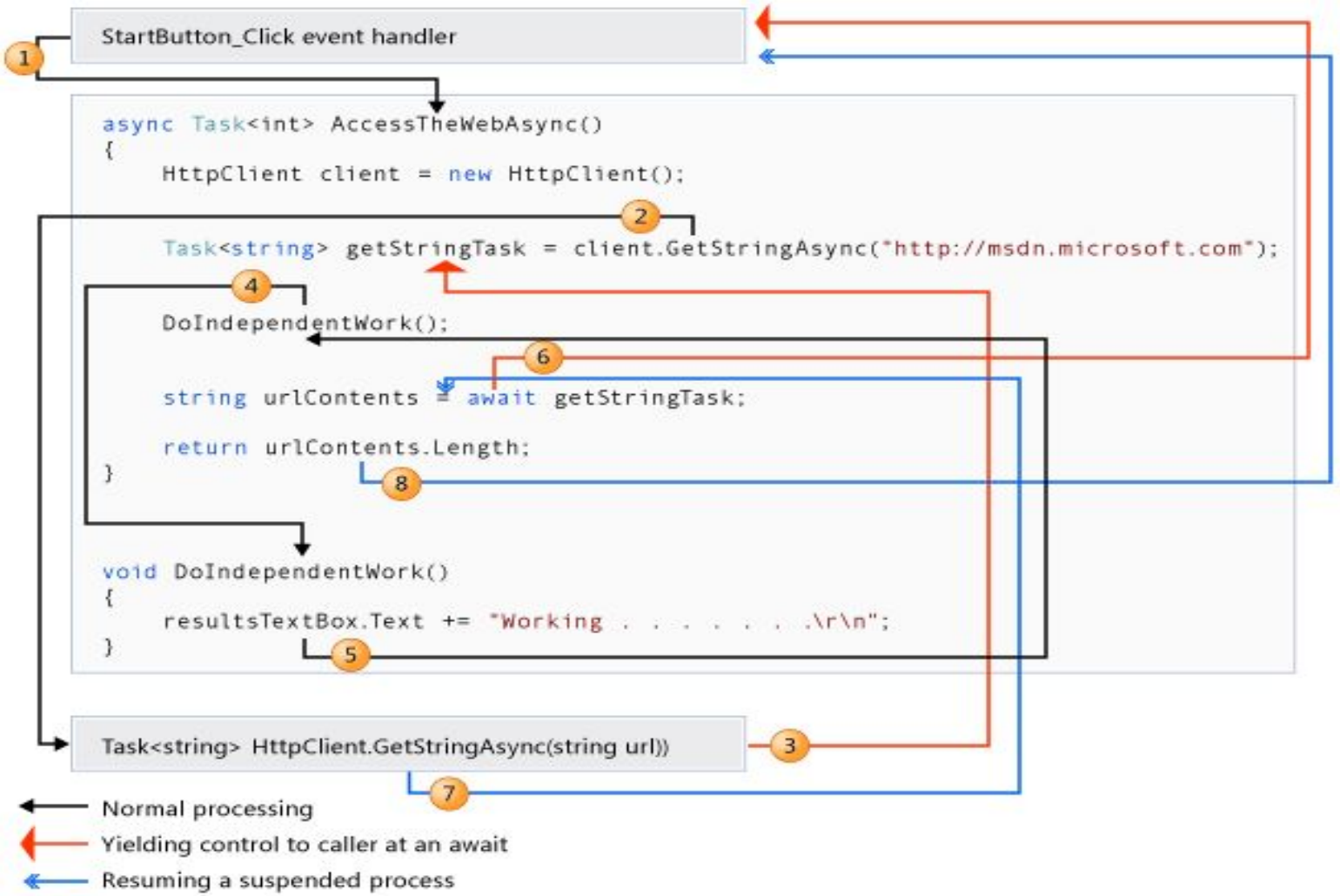
BackgroundWorker vs async await

```
private void LoadData()
{
  var bw = new BackgroundWorker();
  bw.DoWork += (sender, e) =>
  {
     SomeList = GetSomeListFromService();
     QuickCpuCalculationsOnList();
     LongCpuCalculations();
  };
  bw.RunWorkerCompleted += (sender, e) =>
  {
      SomeCollection =
                  new ListCollectionView(SomeList);
  };
  bw.RunWorkerAsync();
}
```

```
private async Task LoadDataAsync()
{
 SomeList = await GetSomeListFromServiceAsync();
 QuickCpuCalculationsOnList();
 await Task.Factory.StartNew(() =>
                                  LongCpuCalculations());

 // jeżeli mamy context, inaczej w metodzie wyżej
 SomeCollection =
                          new ListCollectionView(SomeList);
}
```

async/await - przetwarzanie



Źródło: https://docs.microsoft.com/pl-pl/dotnet/csharp/programming-guide/concepts/async/

Task.ConfigureAwait(false)

- Zatrzymanie przełączania kontekstu synchronizacji
  - *System.ThreadingSynchronizationContext.Current*
- Przyspieszenie całości operacji
- Ochrona przed zakleszczeniem (nie zawsze)
- Konsekwencje użycia
- Stosować czy nie?

Wątki w GUI

- Asynchroniczność w aplikacjach okienkowych
- Zmiana stanu interfejsu z poziomu wątku
  - WPF
  - UWP
  - Windows Forms
- Priorytety wątków
- Eventy
  - Asynchroniczne czekanie na sygnał

Testy metod asynchronicznych

- public async Task Test() (jeżeli framework pozwala)
- do testowania ścieżki rzucającej wyjątkiem:
  - Assert.ThrowsAsync<Exception>(async () => await method())
  - await Assert.ThrowsExceptionAsync<Exception>
    (async () => await method())

Programowanie równoległe

- Parallel
  - *For*
  - *Foreach*
  - *Invoke*
- PLINQ
  - Czym jest Parallel LINQ
  - Wady i zalety
  - *AsParallel*
  - *ForAll*

Reactive Extensions (Rx)

- Model *push-based*
- Implementacja
  - *IObservable<T>*
  - *IObserver<T>*
- Wykorzystanie w WPF

Kolekcje bezpieczne wątkowo

- Kolekcje asynchroniczne
- Kolekcje niemutowalne
- Kolekcje blokujące

Kolekcje bezpieczne wątkowo

- Concurrent collections
  - ConcurrentQueue
  - ConcurrentStack
  - ConcurrentDictionary
  - ConcurrentBag

Kolekcje bezpieczne wątkowo

- BlockingCollection
- BufferBlock

Kolekcje bezpieczne wątkowo

- Immutable collections
  - ImmutableQueue
  - ImmutableStack
  - ImmutableArray
  - ImmutableList
  - ImmutableDictionary, ImmutableSortedDictionary
  - ImmutableSet, ImmutableSortedSet

Wzorce projektowe

- Synchronizacji
- Współbieżności
- Inicjalizacji
- Wzorce obsługi zdarzeń
  - Reactive Extensions
  - Exceptions

Dobre praktyki

- Zagnieżdżone blokady
- volatile
- async Task/Task<>
- ConfigureAwait(false)
- Ustawianie timeout'u

Optymalizacja

- lock vs SpinLock vs no locks
- klasy Slim (SemaphoreSlim, ReaderWriterLockSlim)
- Task.Run vs Task.Factory.StartNew vs new Task
- PLINQ - AsParallel
- Task vs ValueTask - tylko w .NET Core & .NET
- Asynchroniczne inicjowanie obiektów

Podsumowanie