Programowanie Współbieżne

C#
Klasa Parallel

Klasa Parallel

Klasa Parallel w C# jest częścią przestrzeni nazw *System.Threading.Tasks* i służy do wykonywania operacji równoległych.

Umożliwia ona łatwe uruchamianie wielu zadań jednocześnie, co może znacznie przyspieszyć wykonywanie programów, zwłaszcza tych, które wykonują operacje niezależne od siebie i są gruboziarniste.

Klasa Parallel

W klasie Parallel kluczowe są takie metody jak:

- **Parallel.For** - równoległe wykonywanie pętli for. Każda iteracja pętli może być wykonywana na osobnym wątku.
- **Parallel.ForAsync** - odpowiednik powyżesz ale zwracający uruchomione zadanie
- **Parallel.ForEach** - równoległe przetwarzanie elementów kolekcji. Każdy element kolekcji może być przetwarzany na osobnym wątku.
- **Parallel.ForEachAsync** - asynchroniczny odpowiednik *ForEach*
- **Parallel.Invoke** - równoległe wykonywanie wielu akcji. Każda akcja może być wykonywana na osobnym wątku.

Klasa Parallel

Najprostrzy przykład

- Klasyczna pętla

```
for (int i = 0; i < max; i++)
{
     DoSomething(i);
}
```

- Wersja *Parallel*

```
Parallel.For(0, max, (i) => DoSomething(i));
```

- Uproszczona wersja *Parallel*

```
Parallel.For(0, max, DoSomething);
```

Klasa Parallel

Konkretny przykład

```
ThreadPool.SetMinThreads(64, 64);
Console.CursorVisible = false;
int max = Console.WindowHeight;
//anonymous version
Parallel.For(0, max, iter =>
 {
    for (int i = 0; i < Console.WindowWidth; i++)
    {
    lock (Console.Out)
        {
        Console.ForegroundColor = (ConsoleColor)(iter % 16);
        Console.SetCursorPosition(i, iter % Console.WindowHeight)
        Console.Write("#");
        }
    Thread.Sleep(Random.Shared.Next(50));
    }
});
```


Example: LecureParallel/SimpleHashPrinting2

Klasa Parallel

Przerywanie

- W celu przerwania zwykłej pętli użyjemy `break`.
- Wersja Parallel wymaga zatrzymania/przerwania wszystkich działających iteracji.
- Możemy przekazać do Akcji obiekt klasy `ParallelLoopState`.
- i na powyższym wykonać:
  - `Break()` W celu ustawienia statusu `ShouldExitCurrentIteration` i pozwolić dokończyć bieżącą iterację.
  - `Stop()` W celu natychmiastowego przerwania bieżącej iteracji i przekazać do pozostałych tasków informację za pomocą `ShouldExitCurrentIteration`

Example: LecureParallel/SimpleHashPrinting2

Klasa Parallel

Przerywanie przykład

```
static Action<int,ParallelLoopState> PrintHashes = (iter,state) =>
{
    for (int i = 0; i < Console.WindowWidth; i++)
    {
        if (i == 50 && iter == 10)
        {
            state.Break();
            //return; //it will work as state.Stop();
        }
        //if (i == 50 && iter == 10) state.Stop();
        if (state.ShouldExitCurrentIteration)
        {
            return;
        }
        lock (Console.Out)
        {
            Console.ForegroundColor = (ConsoleColor)(iter % 16);
            Console.SetCursorPosition(i, iter % Console.WindowHeight);
            Console.Write((i % 10).ToString());
        }
        Thread.Sleep(Random.Shared.Next(200+50* iter)); //with asymetric delay
    }
};
```

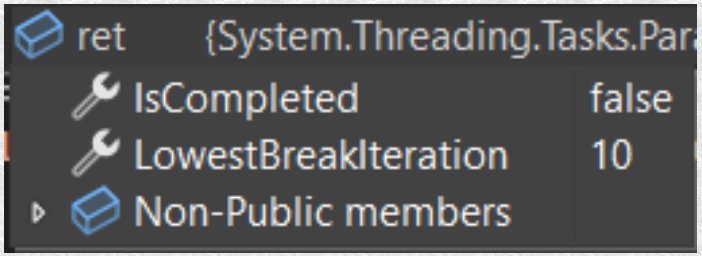
Zakładamy, że przerwiemy, gdy będzie 10 iteracja na 50 znaku

Przerwanie także bieżącej iteracji

Jeżeli nastąpiło żądanie przerwania przez inny task możemy tu na to zareagować

Klasa Parallel

- Parallel.For zwraca obiekt klasy `ParallelLoopResult`
- Można odczytać
  - czy pętla wykonała się w całości `IsCompleted`
  - i jaka była najniższa przerwana iteracja `LowestBreakIteration`



Example: LecureParallel/SimpleHashPrinting3

Klasa Parallel

Stopień równoległości

- Do pętli można przekazać kilka dodatkowych parametrów za pomocą obiektu klasy `ParallelOptions`
- Jedna z opcji pozwala ustawić maksymalny stopień równoległości `MaxDegreeOfParallelism`

```
int iterations = Console.WindowHeight;
ParallelOptions options = new ParallelOptions();
options.MaxDegreeOfParallelism = iterations;

 ParallelLoopResult ret = Parallel.For(1, iterations,
options, (i, state) =>
{ PrintHashes(i, state); });
```


Example: LecureParallel/SimpleHashPrinting3

Klasa Parallel

Wersja Asynchroniczna

- Wersja asynchronicza pętli równoległej to ForAsync
- Zwraca ona uruchomionego Taska na którego wynik można poczekać.
- Mamy trzy przeciążenia ForAsync:

- ```
  public static Task ForAsync<T>(T fromInclusive, T toExclusive, Func<T, CancellationToken, ValueTask> body)
   where T : notnull, IBinaryInteger<T>
  ```

- ```
  public static Task ForAsync<T>(T fromInclusive, T toExclusive, CancellationToken cancellationToken, Func<T, CancellationToken, ValueTask> body)
    where T : notnull, IBinaryInteger<T>
  ```

- ```
  public static Task ForAsync<T>(T fromInclusive, T toExclusive, ParallelOptions parallelOptions, Func<T, CancellationToken, ValueTask> body)
    where T : notnull, IBinaryInteger<T>
  ```
  ```

Klasa Parallel

Wersja Asynchroniczna

- W wersji asynchronicznej Parallel.ForAsync zwraca uruchomione zadanie.
- Może ono działać w tle lub możemy poczekać na zakończenie w dogodnym momencie.
- Możemy też przekazać `CancellationToken` lub `ParallelOptions`
- Wszystkie 3 przeciążenia otrzymują zakres iteracji od do oraz Funkcję której przekazujemy numer iteracji oraz instancję cancellationToken.
- Funkcja dodatkowo zwraca `ValueTask`.

```
static  ValueTask PrintHashes(int iter, CancellationToken ctoken)
```

Klasa Parallel

ValueTask vs Task

- **Task** jest typem referencyjnym, alokowany na stercie. Każda instancja wymaga alokacji pamięci, co może prowadzić do większego zużycia pamięci i częstszego uruchamiania garbage collectora. Jest bardziej uniwersalny i może być używany w większości scenariuszy asynchronicznych. Ma wbudowaną obsługę wyjątków i może przechowywać wyjątki, które wystąpiły podczas wykonywania operacji asynchronicznej.
- **ValueTask** jest typem wartościowym, alokowany na stosie. Może być bardziej efektywny pod względem zarządzania pamięcią, ponieważ unika alokacji na stercie w przypadku zakończonych operacji asynchronicznych. Jest używany głównie w scenariuszach, gdzie operacje asynchroniczne często kończą się natychmiast, co pozwala uniknąć kosztów alokacji pamięci.ma pewne ograniczenia i nie powinien być używany w przypadkach, gdy operacja asynchroniczna może być oczekiwana wielokrotnie, ponieważ może to prowadzić do nieoczekiwanych zachowań.

Klasa Parallel

Task vs ValueTask, kiedy używać

Task

- W większości scenariuszy asynchronicznych, gdzie operacje mogą trwać dłużej.
- Gdy potrzebujesz pełnej obsługi wyjątków i bardziej uniwersalnego podejścia.
- Gdy operacja asynchroniczna może być oczekiwana wielokrotnie.

ValueTask

- Gdy operacja asynchroniczna często kończy się natychmiast.
- Gdy chcesz zminimalizować alokacje pamięci i zmniejszyć obciążenie garbage collectora.
- Gdy jesteś pewien, że operacja asynchroniczna nie będzie oczekiwana wielokrotnie.

Klasa Parallel

ValueTask przykład

```
static  ValueTask PrintHashes(int iter, CancellationToken ctoken)
{
    return new ValueTask(Task.Run(() =>
    {
        for (int i = 0; i < Console.WindowWidth; i++)
        {
            if (ctoken.IsCancellationRequested) break;
            lock (Console.Out)
            {
               Console.ForegroundColor = (ConsoleColor)(iter % 16);
               Console.SetCursorPosition(i, iter %
Console.WindowHeight);
                Console.Write("#");
            }
            Thread.Sleep(Random.Shared.Next(100));
        }
    }));
}
```


Example: LecureParallel/SimpleHashPrinting4

Klasa Parallel

ValueTask przykład

```
CancellationTokenSource cancellationSource = new CancellationTokenSource();
int iterations = 32;
ParallelOptions options = new ParallelOptions()
    {
    CancellationToken = cancellationSource.Token ,
    MaxDegreeOfParallelism = iterations
    };
Task ret = Parallel.ForAsync(0, iterations, options, PrintHashes);
cancellationSource.CancelAfter(2000);
try
{
    ret.Wait();
}
catch (Exception ex)
{
    Console.SetCursorPosition(0, iterations);
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine($"We have exception {ex.Message}");
}
```


Example: LecureParallel/SimpleHashPrinting4

Klasa Parallel

TaskScheduler

```
 sealed class TwoThreadTaskScheduler : TaskScheduler,
IDisposable
   {
   ...
   }
```

```
ParallelOptions options = new ParallelOptions()
  {   //custom taskscheduler
      TaskScheduler = new TwoThreadTaskScheduler(6),
      MaxDegreeOfParallelism = iterations
  };
  Parallel.For(0, iterations,options, PrintHashes);
```


Example: LecureParallel/SimpleHashPrinting7

Klasa Parallel

- Invoke wywołuje jednocześnie kilka akcji naraz

```
static void PrintAsterisk()
{
    for (int i = 0; i < Console.WindowWidth; i++)
    {
        lock (Console.Out)
        {
            Console.ForegroundColor = (ConsoleColor)(Random.Shared.Next(1, 16));
            Console.SetCursorPosition(Random.Shared.Next(0, Console.WindowWidth),
Random.Shared.Next(0, Console.WindowHeight));
            Console.Write("*");
        }
        Thread.Sleep(Random.Shared.Next(50));
    }
}
static void Main(string[] args)
{
    ThreadPool.SetMinThreads(64, 64);
    Console.CursorVisible = false;
    Parallel.Invoke(PrintAsterisk, PrintAsterisk, PrintAsterisk, PrintAsterisk);
    Console.SetCursorPosition(0, Console.WindowHeight);
    Console.ForegroundColor = ConsoleColor.White;
}
```

Wywołanie w sposób współbieżny podanych akcji

Example: LecureParallel/SimpleAsteriskPrinting1

Klasa Parallel

Parallel.Invoke

- Wywołanie całej tablicy akcji

```
static Action<int> PrintHashes = (iter) =>
{
  ...
}

int threadsNo = Console.WindowHeight - 1; ;

Action[] actions = new Action[threadsNo];
for (int i = 0; i < threadsNo; i++)
{
    int iter = i;
    actions[i] = () => PrintHashes(iter);
}
Parallel.Invoke(actions);
```

Wywołanie w sposób współbieżny całego zestawu Akcji

Example: LecureParallel/SimpleHashPrinting5

Klasa Parallel

Parallel.Invoke asynchroniczne

```
  async Task InvokeAsync(params Action[] actions)
{
    var tasks = actions.Select(a => Task.Run(a));
    await Task.WhenAll(tasks);
}

static void Main(string[] args)
{
 ThreadPool.SetMinThreads(64, 64);
 Console.CursorVisible = false;
 int threadsNo = Console.WindowHeight;
 Action[] actions = new Action[threadsNo];
 for (int i = 0; i < threadsNo; i++)
    {
        int iter = i;
        actions[i] = () => PrintHashes(iter);
    }
Task runningActions = InvokeAsync(actions);
runningActions.Wait(); //Prevent from ending too early
}
```

Do każdej akcji utwórz zadanie następnie zwróć task czekający na wszystkie zadania

Przygotuj zestaw akcji

Wywołanie w sposób asynchroniczny całego zestawu akcji

Example: LecureParallel/SimpleHashPrinting6

Parallel For i inne

```
int[] numbers = GenerateRandomArray(sizeOfProblem);
int sum = 0;

foreach (int number in numbers) sum += number;
Parallel.ForEach(numbers, number => sum += number);
Parallel.ForEach(numbers, number => { lock (sw) { sum += number; } });
sum = numbers.Sum(number => number);
numbers.AsParallel().ForAll(number => sum += number);
numbers.AsParallel().ForAll(number => { lock (sw) { sum += number; } });

sum = numbers.AsParallel().Sum(number => number);
sum = numbers.AsParallel().Sum();
```


Example: LecureParallel/VerySimpleSum

Parallel For i inne

```
 Parallel.For<double>(0, numbers.Length, new ParallelOptions
{ MaxDegreeOfParallelism = 8 },
    // Initialize the local states
    () => 0,
    // Accumulate the thread-local computations in the loop body
    (i, loop, localState) =>
    {
        return localState + numbers[i];
    },
    // Combine all local states
    localState => { lock (numbers) { sum += localState; } }
 );
```


Example: LecureParallel/VerySimpleSum

Parallel For i inne

```
var tasks = new List<Task<double>>();
int processorCount = Environment.ProcessorCount;
int end = numbers.Length - 1;
int range = (end + 1) / processorCount;
//   Console.WriteLine($"I found {processorCount} processors");
for (int i = 0; i < processorCount; i++)
{
    int rangeStart = i * range;
    int rangeEnd = (i + 1) * range;
    tasks.Add(Task.Run(() =>
    {
        double localSum = 0;
        for (int j = rangeStart; j < rangeEnd; j++)
        {
            localSum += numbers[j];
        }
        return localSum;

    }));
}
double[] partialResults = Task.WhenAll(tasks).Result;
sum = partialResults.Sum();
```


Example: LecureParallel/VerySimpleSum

Parallel For i inne

```
Simple sum sumed by foreach in:                                271     with result     450001866
Simple sum sumed by unsafe Parallel.Foreach in:                1038    with result     60346552
Simple sum sumed by Parallel.Foreach with lock in:             4814    with result     450001866
Simple sum sumed by one thread Linq in:                        529     with result     450001866
Simple sum by Linq as Parallel ForAll without lock in:         1217    with result     63061203
Simple sum by Linq as Parallel ForAll  with lock in:           4548    with result     450001866
Simple sum by Linq as Parallel without selector in:            74      with result     450001866
Simple sum  by Parallel.For and LocalState without lock in:    190     with result     176834245
Simple sum by Parallel.For and LocalState with lock in:        173     with result     450001866
Simple sum by 8 tasks in:                                      93      with result     450001866
```


Example: LecureParallel/VerySimpleSum

Parallel For i inne

```
sum = numbers.AsParallel().Sum(number => number);
sum = numbers.AsParallel().Sum();
```

```
Simple sum sumed by foreach in:                          265    with result    450012306
Simple sum sumed by unsafe Parallel.Foreach in:          1044   with result    62565649
Simple sum sumed by Parallel.Foreach with lock in:       5996   with result    450012306
Simple sum sumed by one thread Linq in:                  578    with result    450012306
Simple sum by Linq as Parallel ForAll without lock in:   1582   with result    61263259
Simple sum by Linq as Parallel ForAll  with lock in:     6608   with result    450012306
Simple sum by Linq as Parallel with selector in:         431    with result    450012306
Simple sum by Linq as Parallel without selector in:      337    with result    450012306
Simple sum  by Parallel.For and LocalState without lock in:  233  with result  265437820
Simple sum by Parallel.For and LocalState with lock in:  216    with result    450012306
Simple sum by 8 tasks in:                                101    with result    450012306
```


Example: LecureParallel/VerySimpleSum

Parallel For i inne

```
static double calculatedValue(double value) //always returns 1
    {
    return (Math.Sin(Math.Sin(value * value) * Math.Cos(value * value)
     / Math.PI) * Math.Sin(Math.Sin(value * value) * Math.Cos(value * value)
     / Math.PI)) > 1 ? 0 : 1;
    }

foreach (int number in numbers)
     sum += calculatedValue(number);
Parallel.ForEach(numbers, number => sum += calculatedValue(number));
Parallel.ForEach(numbers, number => { lock (sw)
                                       { sum += calculatedValue(number); }
                                   });
Parallel.For(0, numbers.Length, i => sum += calculatedValue(numbers[i]));
Parallel.For(0,numbers.Length, i => { lock (sw) { sum +=
calculatedValue(numbers[i]); } });
sum = numbers.Sum(number => calculatedValue(number));
sum = numbers.AsParallel().Sum(number => calculatedValue(number));
```


Example: LecureParallel/SumWithCalculation

Parallel For i inne

```
 Parallel.For<double>(0, numbers.Length, //new ParallelOptions
{ MaxDegreeOfParallelism = 8 },
    // Initialize the local states
    () => 0,
    // Accumulate the thread-local computations in the loop body
    (i, loop, localState) =>
    {
        return localState + calculatedValue(numbers[i]);
    },
    // Combine all local states
    localState => { lock (numbers) { sum += localState; } }
 );
```


Example: LecureParallel/SumWithCalculation

Parallel For i inne

```
var tasks = new List<Task<double>>();
int processorCount = Environment.ProcessorCount;
int end = numbers.Length - 1;
int range = (end + 1) / processorCount;
for (int i = 0; i < processorCount; i++)
{
    int rangeStart = i * range;
    int rangeEnd = (i + 1) * range;
    tasks.Add(Task.Run(() =>
    {
        double localSum = 0;
        for (int j = rangeStart; j < rangeEnd; j++)
        {
            localSum += calculatedValue(numbers[j]);
        }
        return localSum;

    }));
}
double[] partialResults = Task.WhenAll(tasks).Result;
sum = partialResults.Sum();
```


Example: LecureParallel/SumWithCalculation

Parallel For i inne

```
One thread foreach in:                              755     with result 10000000
Unsafe Parallel.Foreach in:                         224     with result 1247965
Parallel.Foreach with lock in:                      1229    with result 10000000
Unsafe Parallel.For in:                             215     with result 1180724
Parallel.For with lock in:                          1217    with result 10000000
Simple one thread Linq in:                          779     with result 10000000
Linq as Parallel in:                                220     with result 10000000
Parallel.For and LocalState without lock in:        184     with result 8709343
Parallel.For and LocalState with lock in:           176     with result 10000000
Simple 8 tasks in:                                  171     with result 10000000
```


Example: LecureParallel/SumWithCalculation