Parallel

What is the difference between Parallel.For and Parallel.ForEach

Answer: Parallel.For represent asynchronous implementation of for loop, Parallel.ForEach represent asynchronous implementation of foreach loop, so the difference between them similar with difference of for and foreach.

Why are Parallel loops need?

Answer: Parallel loops need for processing data asynchronously.

Tasks

Difference between Thread and Task?

- Answer: Task represents an asynchronous operation (in .NET)
- Thread is one of the many possible workers which performs that task

When we use Thread.Sleep? When Task.Delay?

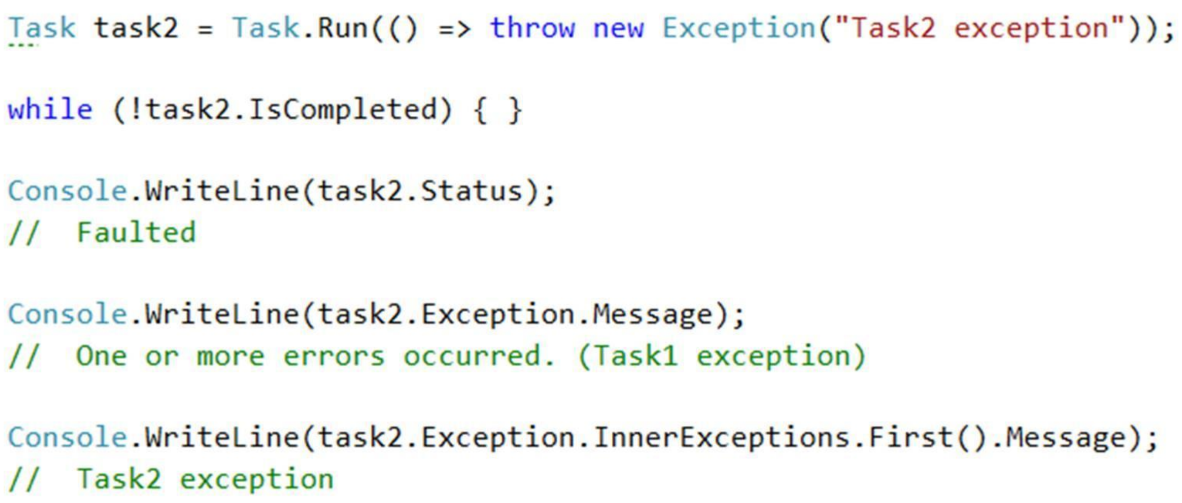
Answer: When we need to block current thread we use Thread.Sleep. When we need to wait without blocking current thread we use Task.Delay with async/await.

How to handle exception in Task?

Answer: we can catch exception in 2 ways. 1st invoke method Wait of Task and cover it with try/catch (AggregateException)

```
Task task = Task.Run(() => throw new Exception("Task1 exception"));

try
{
    task.Wait(); // Throws exception
}
catch (AggregateException e)
{
    Console.WriteLine(task.Status);
    //  Faulted

    Console.WriteLine(e.Message);
    //  One or more errors occurred. (Task1 exception)

    Console.WriteLine(e.InnerExceptions.First().Message);
    //  Task1 exception
}
```

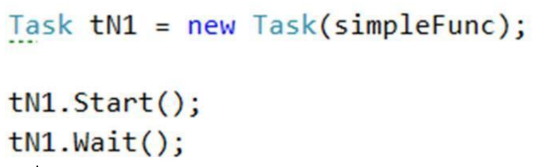
Second way is use property Exception of task. It will contain Aggregate exception and our exception in InnerException property.



```
Task task2 = Task.Run(() => throw new Exception("Task2 exception"));

while (!task2.IsCompleted) { }

Console.WriteLine(task2.Status);
//  Faulted

Console.WriteLine(task2.Exception.Message);
//  One or more errors occurred. (Task1 exception)

Console.WriteLine(task2.Exception.InnerExceptions.First().Message);
//  Task2 exception
```

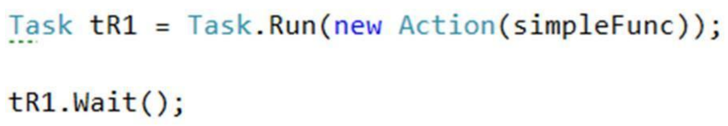
Ways to create Task

Answer: We have 3 ways to create task. 1st use constructor. In this case we need to start our Task instance before wait.



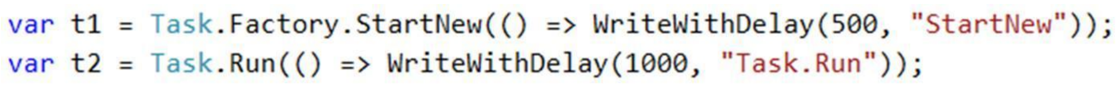
```
Task tN1 = new Task(simpleFunc);

tN1.Start();
tN1.Wait();
```

2nd way: use static method Task.Run, that's create task and started it.



```
Task tR1 = Task.Run(new Action(simpleFunc));

tR1.Wait();
```

3rd way: use static method Task.Factory.StartNew. It's similar previous way, but have more overloads and ways to customize our task.



```
var t1 = Task.Factory.StartNew(() => WriteWithDelay(500, "StartNew"));
var t2 = Task.Run(() => WriteWithDelay(1000, "Task.Run"));
```

What is Task scheduler?

Answer: An instance of the TaskScheduler class represents a task scheduler. A task scheduler ensures that the work of a task is eventually executed.

> The default task scheduler is based on the .NET Framework 4 thread pool, which provides work-stealing for load-balancing, thread injection/retirement for maximum throughput, and overall good performance. It should be sufficient for most scenarios.
>
> The TaskScheduler class also serves as the extension point for all customizable scheduling logic. This includes mechanisms such as how to schedule a task for execution, and how scheduled tasks should be exposed to debuggers. If you require special functionality, you can create a custom scheduler and enable it for specific tasks or queries.

PLINQ

Difference PLINQ and LINQ? When we should use PLINQ?

Answer: it resolves similar issues and difference in how it process data. LINQ process data synchronously, PLINQ makes asynchronously. When we process large count of data we prefer use PLINQ, because it will process it in few threads and will make it faster. But if we process small count of data we prefer LINQ, because parallel processing might give worth performance in this case.

Difference between Parallel.ForEach and PLINQ ForAll?

Answer: 1st of all, method ForAll need when we have complex PLINQ query and we need to process the result. But when we have some collection and we need make simple actions with it Parallel.ForEach makes it better. The difference between them in count of threads that they use. PLINQ.ForAll – uses optimal count of thread usually it's equal count of processor cores. Parallel.ForEach – uses threadpool, and can be use more thread than processor core.

When we should use AsOrdered and AsUnordered?

Answer: AsOrdered – need when sequence of data is important for us. AsUnordered – need when sequence of data is not important for us.