C# - TPL

Slides Roadmap

- Introduction Parallel Programming
- Task programming
- Coordinating Tasks
- Parallel Loops
- Parallel Linq

Introduction

- .NET 1.0 : classic threading
  - Namespace: System.Threading
  - Main class: Thread

- .NET 4.0: Task Parallel Library (TPL)
  - Namespace: System.Threading
  - Main class: Task

Introduction

- TPL built on top of Threads
- Number of threads calculated dynamically by the environment
- TPL uses ThreadPool to distribute the work

Introduction

- Thread

```
var thread = new Thread(start =>
{
});
thread.Start();
```

- Task

```
Task.Factory.StartNew(() =>
    {

    });
```

Task programming

- Create a task with System.Action parameter
  - Explicit Action
  - Anonymous function
  - Lambda function

- Independent unit of work
- Maximize performance of the application (ThreadPool)
- You can identify a task with Task.CurrentId

Task programming

- Add a Task state with the System.Action<object> parameter

```
string[] messages = { "task1", "task2", "task3" };
foreach (var msg in messages)
{
    Task.Factory.StartNew((obj) =>
    {
        Console.WriteLine("Message: " + obj.ToString());
    }, msg);
}

Console.ReadLine();
```

Task programming

- Return a result from a Task

```
var task = Task.Factory.StartNew<string>(() =>
    {
        return "hello!";
    });
Console.WriteLine("Result {0}", task.Result);
Console.ReadLine();
```

Demo

- Task factory
- Add task state
- Get result from task

Task programming

- Cancelling a Task
  - Create a CancelletionTokenSource
  - Call CancellationTokenSource.Token property
  - Create task with CancellationToken
  - Start the task
  - Call Cancel method on the CancellationTokenSource

Task programming

```
var tokenSource = new CancellationTokenSource();
var token = tokenSource.Token;
var task = new Task(() =>
    {
        for (var i = 0; i < int.MaxValue; i++)
        {
            if (token.IsCancellationRequested)
            {
                Console.WriteLine("Task cancel detected");
                throw new OperationCanceledException(token);
            }
            else
            {
                Console.WriteLine("value {0}", i);
            }
        }
    }, token);
task.Start();
tokenSource.Cancel();
```

Task programming

- Task sleeping with Wait Handle
  - Sleep for a specific interval or until cancellation

```
bool cancelled = token.WaitHandle.WaitOne(1000);
```

- Task sleeping with Thread.Sleep
  - Cancelling the token doesn’t cancel the task immediately

Task programming

- Waiting for tasks
  - Wait(): wait for a single task
  - WaitForAll(): wait for a set of tasks
  - WaitAny(): wait for the first of a set of tasks
  - WhenAll:() wait for a set of tasks and return a task
  - WhenAny(): wait for the first of a set of tasks and return a task

Task programming

- Tasks Exceptions:
  - System.AggregateException is a collection of exceptions triggered by Task.Wait(), Task.WailAll(), Task.WaitAny(), Task.Result

- Some task properties
  - IsCompleted
  - IsFaulted
  - IsCancelled
  - Exception

Coordinating Tasks

- Task continuation
  - **ContinueWith()**
  - **ContinueWhenAll()**
  - **ContinueWhenAny()**

Coordinating Tasks

- Task continuation

```
Task<BankAccount> task = new Task<BankAccount>(() =>
{
    BankAccount account = new BankAccount();
    for (int i = 0; i < 1000; i++)
    {
        account.Balance++;
    }
    return account;
});

Task<int> continuationTask  = task.ContinueWith<int>((Task<BankAccount> antecedent) =>
{
    Console.WriteLine("Interim Balance: {0}", antecedent.Result.Balance);
    return antecedent.Result.Balance * 2;
});

task.Start();

Console.WriteLine("Final balance: {0}", continuationTask.Result);
Console.WriteLine("Press enter to finish");
Console.ReadLine();
```

Parallel Loops

- Parallel loops available with **System.Threading.Tasks.Parallel** class

- Parallel loops and actions:
  - Invoke
  - For
  - ForEach
  - Break (complete all iterations on all threads before current iteration)
  - Stop (stop as soon as possible)

Parallel Loops

- Parallel.Invoke

```
Action[] actions = new Action[3];
actions[0] = new Action(() => Console.WriteLine("Action 1"));
actions[1] = new Action(() => Console.WriteLine("Action 2"));
actions[2] = new Action(() => Console.WriteLine("Action 3"));
Parallel.Invoke(actions);
```

```
Task parent = Task.Factory.StartNew(() =>
{
    foreach (Action action in actions)
    {
        Task.Factory.StartNew(action, TaskCreationOptions.AttachedToParent);
    }
});
parent.Wait();
```

Parallel Loops

- Parallel For loop

- **for (int i = 0; i < 10; i++)**

```
Parallel.For(0, 10, index =>
{
    Console.WriteLine("Task ID {0} processing index: {1}",
    Task.CurrentId, index);
});
```

Parallel Loops

- Parallel ForEach loop

```
Parallel.ForEach(messages, msg =>
{
    Console.WriteLine(msg);
});
```

Parallel Loops

- ParallelOptions
  - CancellationToken
  - MaxDegreeOfParallelist (-1 = no limit)
  - TaskScheduler (null = default taskScheduler)

- ParallelLoopState

```
Parallel.For(0, 10, (int index, ParallelLoopState loopState) =>
{
    if (index == 2)
    {
        //loopState.Stop();
        //loopState.Break();
    }
});
```

Parallel Linq

- Some members of ParallelEnumerable class
  - AsParallel – convert a IEnumerable to ParallelQuery
  - AsSequential – convert ParallelQuery to Ienumerable
  - AsOrdered – modify a ParallelQuery to preserve ordering
  - AsUnordered – modify a ParallelQuery to discard ordering
  - WithCancellation – monitor a cancellation token
  - WithDegreeOfParallelism – limit of tasks

Async / Await

- asynchronous code: difficult to write, debug, and maintain
- With async / await keywords:
  - The compiler does the difficult work
  - Improve responsiveness
  - Easier to write

- The method runs on the current synchronization context
- Async method typically returns a Task or Task<Result>
- Only async method can use await keyword
- Async method can be awaited (if return a Task)

Async / Await

- Exemple

```
0 references
private async Task<string> GetData()
{
    WebClient client = new WebClient();
    var data = await client.DownloadStringTaskAsync("http://www.google.com");
    if (string.IsNullOrWhiteSpace(data))
    {
        throw new NullReferenceException("data");
    }
    return data;
}
```