Single Tasking : We perform on task at a time

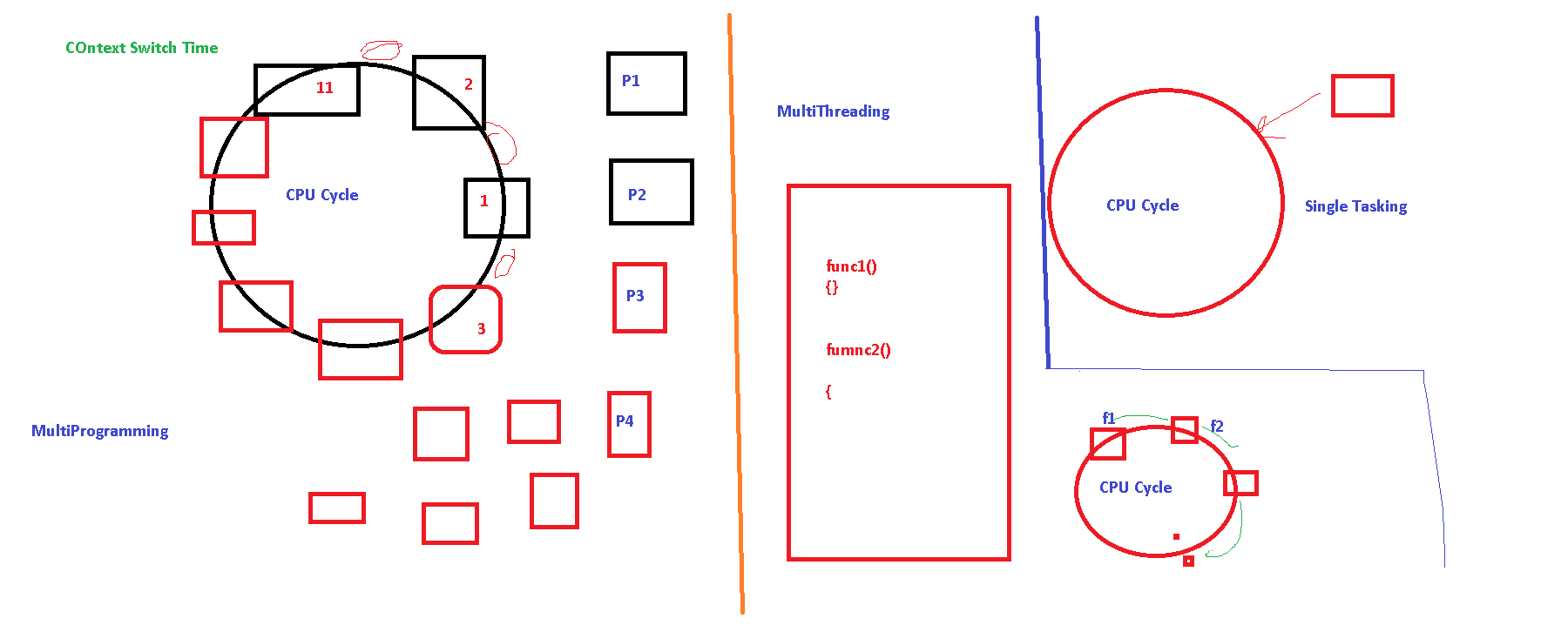
Multi Tasking : We perform more than one task at a time

Which one is more efficient

Multitasking can be achieved ??

Using Multiprogramming OR MultiThreading

Out of these which is more efficient



Context Switch Time : Time taken to move from one program to other

In case of MultiThreading, here will be one thread.

Task > Threads will be there and thread will come from thread pool

And differ threads will run parallely on different cores

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace Threading\_TPLDemo

{

class ThreadDemo

{

static void Func1()

{

for(int i=1;i<=10;i++)

{

Console.WriteLine(i);

// Thread.Sleep(10);

}

}

static void Func2()

{

for (int i = 11; i <= 20; i++)

{

Console.WriteLine(i);

}

}

static void Main()

{

ThreadStart ts1 = new ThreadStart(Func1);

ThreadStart ts2 = new ThreadStart(Func2);

Thread t1 = new Thread(ts1);

Thread t2 = new Thread(ts2);

t1.Start();

t2.Start();

t1.Join();

Console.WriteLine("Main thread has ended");

//Func1();

//Func2();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Threading\_TPLDemo

{

class Program

{

**static void Main(string[] args)**

**{**

**Task task1 = new Task(new Action(GetTheTime));**

**task1.Start();**

**task1.Wait();**

**}**

**private static void GetTheTime()**

**{**

**Console.WriteLine("Started");**

**Console.WriteLine("The time now is {0}", DateTime.Now);**

**}**

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Threading\_TPLDemo

{

class Program

{

static void Main(string[] args)

{

Task task1 = new Task(new Action(GetTheTime));

Task task2 = new Task(delegate {

Console.WriteLine("The time now by is {0}", DateTime.Now);

});

task1.Start();

task1.Wait();

task2.Start();

task2.Wait();

}

private static void GetTheTime()

{

Console.WriteLine("Started");

Console.WriteLine("The time now is {0}", DateTime.Now);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Threading\_TPLDemo

{

class Program

{

static void Main(string[] args)

{

Task task1 = new Task(new Action(GetTheTime));

Task task2 = new Task(delegate {

Console.WriteLine("The time now by is {0}", DateTime.Now);

});

Task task3 = new Task(() => MyMethod());

// This is equivalent to: Task task1 = new Task( delegate(MyMethod) );

task1.Start();

task1.Wait();

task2.Start();

task2.Wait();

task3.Start();

task3.Wait();

}

private static void GetTheTime()

{

Console.WriteLine("Started");

Console.WriteLine("The time now is {0}", DateTime.Now);

}

public static void MyMethod()

{

Console.WriteLine("Called this method");

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Threading\_TPLDemo

{

class Program

{

static void Main(string[] args)

{

Task task1 = new Task(new Action(GetTheTime));

Task task2 = new Task(delegate {

Console.WriteLine("The time now by is {0}", DateTime.Now);

});

Task task3 = new Task(() => MyMethod());

// This is equivalent to: Task task1 = new Task( delegate(MyMethod) );

// Create and queue a task that returns the day of the week as a string.

Task<string> task4 = Task.Run<string>(() => DateTime.Now.DayOfWeek.ToString());

// Retrieve and display the task result.

Console.WriteLine(task4.Result);

task1.Start();

task1.Wait();

task2.Start();

task2.Wait();

task3.Start();

task3.Wait();

}

private static void GetTheTime()

{

Console.WriteLine("Started");

Console.WriteLine("The time now is {0}", DateTime.Now);

}

public static void MyMethod()

{

Console.WriteLine("Called this method");

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Threading\_TPLDemo

{

class Program

{

static void Main(string[] args)

{

int from = 0;

int to = 500000;

double[] array = new double[500000];

// This is a sequential implementation:

for (int index = 0; index < 500000; index++)

{

array[index] = Math.Sqrt(index);

}

//for (int index = 0; index < array.Length-1; index++)

//{

// Console.WriteLine(array[index]);

//}

// This is the equivalent parallel implementation:

Parallel.For(from, to, index =>

{

array[index] = Math.Sqrt(index);

});

for (int index = 0; index < array.Length - 1; index++)

{

Console.Write(array[index]);

}

Task task1 = new Task(new Action(GetTheTime));

Task task2 = new Task(delegate {

Console.WriteLine("The time now by is {0}", DateTime.Now);

});

Task task3 = new Task(() => MyMethod());

// This is equivalent to: Task task1 = new Task( delegate(MyMethod) );

// Create and queue a task that returns the day of the week as a string.

Task<string> task4 = Task.Run<string>(() => DateTime.Now.DayOfWeek.ToString());

// Retrieve and display the task result.

Console.WriteLine(task4.Result);

task1.Start();

task1.Wait();

task2.Start();

task2.Wait();

task3.Start();

task3.Wait();

}

private static void GetTheTime()

{

Console.WriteLine("Started");

Console.WriteLine("The time now is {0}", DateTime.Now);

}

public static void MyMethod()

{

Console.WriteLine("Called this method");

}

}

}

Synchronous Programming > When you call a method , you wait for its response, Until then u are blocked

Asynchronous Programming > When you call a method , you don’t wait for its response, you can do other tasks