**Asynchronous programming and Threading in C#**

Threading and asynchronous programming are not same, even they are not related. Still if we can see them together, I think we may have better understanding of both the features.

**Topics covered in this article**

1. Introduction to asynchronous programming
2. Introduction to threading
3. Introduction to TAP
4. Introduction to TPL
5. Conclusion

**Asynchronous Programming**

Asynchronous operation means that the operation runs independent of main or other process flow. In general c# program starts executing from the Main method and ends when the Main method returns. In between all the operations runs sequentially one after another. One operation must wait until its previous operation finishes. Let’s see following code:

static void Main(string[] args)

{

DoTaskOne();

DoTaskTwo();

}

Method “DoTaskTwo” would not be started until “DoTaskOne” finishes. In other words method “DoTaskOne” blocks the execution as long it takes to finish.

In asynchronous programming a method is called that runs in the background and the calling thread is not blocked. After calling the method the execution flow immediately backs to calling thread and performs other tasks. Normally it uses Thread or Task (We will discuss Thread and Task in detail later).

In our case if we run the “DoTaskOne” asynchronously, after calling the “DoTaskOne” method, execution flow immediately backs to Main method and start “DoTaskTwo”.

We can create our own thread using Thread class or use asynchronous patterns provided by .Net to perform asynchronous programming. There are three different asynchronous patterns in .Net:

1. Asynchronous Programming Model (APM) pattern
2. Event-based Asynchronous Pattern (EAP)

Both the above models are not recommended by Microsoft so we will not discuss about them. If you are interested you can read more from following msdn link:

<https://msdn.microsoft.com/en-us/library/ms228963(v=vs.110).aspx> and

<https://msdn.microsoft.com/en-us/library/ms228969(v=vs.110).aspx>

1. Task-based Asynchronous Pattern (TAP): This model is recommended so we will discuss it in detail

**Threading**

Creating a new tread is costly, it takes time. Unless we need to control a thread, then “Task-based Asynchronous Pattern (TAP)” and “Task Parallel Library (TPL)” is good enough for asynchronous and parallel programming. TAP and TPL uses Task (we will discuss what is Task latter). Task can be run:

1. In the current thread
2. In a new thread
3. In a thread from the thread pool (A *thread pool* is a collection of threads already created and maintained by .Net framework. If we use Task, most of the cases we need not to use thread pool directly. Still if you want to know more about thread pool visit the link: ”https://msdn.microsoft.com/en-us/library/h4732ks0.aspx” )
4. Or even without any thread

But if we use Task, as a developer we need not to worry about creation or uses of the thread, .Net framework handles the inner difficulties for us.

Anyway if we need some control over the thread like,

1. We want to set a name for the tread
2. We want to set priority for the thread
3. We want to make our thread foreground or background

Then we may have to create our own thread using thread class.

**Creating Thread using Thread class**

The constructor of Thread class accepts a delegate parameter of type

1. ThreadStart: This delegate defines a method with a void return type and no parameter.
2. And ParameterizedThreadStart: This delegate defines a method with a void return type and one object type parameter.

Following is the simple example how we can start a new thread with Start method:

static void Main(string[] args)

{

Thread thread = new Thread(DoTask);

thread.Start();// Start DoTask method in a new thread

//Do other tasks in main thread

}

static public void DoTask() {

//do something in a new thread

}

We can use lamda expression instead of named method:

static void Main(string[] args)

{

Thread thread = new Thread(() => {

//do something in a new thread

});

thread.Start();// Start a new thread

//Do other tasks in main thread

}

If we don’t require the variable reference we can even start the thread directly like:

static void Main(string[] args)

{

new Thread(() => {

//do something in a new thread

}).Start();// Start a new thread

//Do other tasks in main thread

}

But if we want to control the tread object after it is created we require the variable reference. We can assign different values to the different properties of the object like:

static void Main(string[] args)

{

Thread thread = new Thread(DoTask);

thread.Name = "My new thread";// Asigning name to the thread

thread.IsBackground = false;// Made the thread forground

thread.Priority = ThreadPriority.AboveNormal;// Setting thread priority

thread.Start();// Start DoTask method in a new thread

//Do other task in main thread

}

With the reference variable we can perform some function like abort the thread or wait for the tread to complete by calling the join method. If we call join to a thread the main tread blocks until the calling thread completes.

If we want to pass some data to the method we can pass it as a parameter of Start method. As the method parameter is object type we need to cast it properly.

static void Main(string[] args)

{

Thread thread = new Thread(DoTaskWithParm);

thread.Start("Passing string");// Start DoTaskWithParm method in a new thread

//Do other task in main thread

}

static public void DoTaskWithParm(object data)

{

//we need to cast the data to appropriate object

}

**Task-based Asynchronous Pattern**

.Net framework introduced two new keywords to perform asynchronous programing: “async” and “await”. To use “await” keyword within a method we need to declare the method with “async” modifier. “await” keyword is used before calling an asynchronous method. “await” keyword suspends further execution of the method and control is return to the calling thread. See the example:

private **async** static void CallerWithAsync()// async modifier is used

{

string result = **await** GetSomethingAsync();// await is used before a method call. It suspends //execution of CallerWithAsync() method and the calling thread can perform other task.

Console.WriteLine(result);// this line would not be executed before GetSomethingAsync() //method completes

}

We cannot use await keyword before all the methods. To use “await” the method must have following criteria: