**Lecture 12 - Threads**

A **thread** is defined as the execution path of a program. Each thread defines a unique flow of control. If your application involves complicated and time consuming operations then it is often helpful to set different execution paths or threads, with each thread performing a particular job.

Threads are **lightweight processes**. One common example of use of thread is implementation of concurrent programming by modern operating systems. Use of threads saves wastage of CPU cycle and increase efficiency of an application.

So far we have written programs where a single thread runs as a single process which is the running instance of the application. However, this way the application can perform one job at a time. To make it execute more than one task at a time, it could be divided into smaller threads.

## Thread Life Cycle

The life cycle of a thread starts when an object of the System.Threading.Thread class is created and ends when the thread is terminated or completes execution.

Following are the various states in the life cycle of a thread:

* **The Unstarted State**: it is the situation when the instance of the thread is created but the Start method has not been called.
* **The Ready State**: it is the situation when the thread is ready to run and waiting CPU cycle.
* **The Not Runnable State**: a thread is not runnable, when:
  + Sleep method has been called
  + Wait method has been called
  + Blocked by I/O operations
* **The Dead State**: it is the situation when the thread has completed execution or has been aborted.

## The Main Thread

In C#, the **System.Threading.Thread** class is used for working with threads. It allows creating and accessing individual threads in a multithreaded application. The first thread to be executed in a process is called the **main** thread.

When a C# program starts execution, the main thread is automatically created. The threads created using the **Thread** class are called the child threads of the main thread. You can access a thread using the **CurrentThread** property of the Thread class.

The following program demonstrates main thread execution:

using System;

using System.Threading;

namespace MultithreadingApplication

{

class MainThreadProgram

{

static void Main(string[] args)

{

Thread th = Thread.CurrentThread;

th.Name = "MainThread";

Console.WriteLine("This is {0}", th.Name);

Console.ReadKey();

}

}

}

When the above code is compiled and executed, it produces the following result:

This is MainThread

## Creating Threads

Threads are created by extending the Thread class. The extended Thread class then calls the **Start()**method to begin the child thread execution.

The following program demonstrates the concept:

using System;

using System.Threading;

namespace MultithreadingApplication

{

class ThreadCreationProgram

{

public static void CallToChildThread()

{

Console.WriteLine("Child thread starts");

}

static void Main(string[] args)

{

ThreadStart childref = new ThreadStart(CallToChildThread);

Console.WriteLine("In Main: Creating the Child thread");

Thread childThread = new Thread(childref);

childThread.Start();

Console.ReadKey();

}

}

}

When the above code is compiled and executed, it produces the following result:

In Main: Creating the Child thread

Child thread starts

## Managing Threads

The Thread class provides various methods for managing threads.

The following example demonstrates the use of the **sleep()** method for making a thread pause for a specific period of time.

using System;

using System.Threading;

namespace MultithreadingApplication

{

class ThreadCreationProgram

{

public static void CallToChildThread()

{

Console.WriteLine("Child thread starts");

// the thread is paused for 5000 milliseconds

int sleepfor = 5000;

Console.WriteLine("Child Thread Paused for {0} seconds",

sleepfor / 1000);

Thread.Sleep(sleepfor);

Console.WriteLine("Child thread resumes");

}

static void Main(string[] args)

{

ThreadStart childref = new ThreadStart(CallToChildThread);

Console.WriteLine("In Main: Creating the Child thread");

Thread childThread = new Thread(childref);

childThread.Start();

Console.ReadKey();

}

}

}

When the above code is compiled and executed, it produces the following result:

In Main: Creating the Child thread

Child thread starts

Child Thread Paused for 5 seconds

Child thread resumes

## Destroying Threads

The **Abort()** method is used for destroying threads.

The runtime aborts the thread by throwing a **ThreadAbortException**. This exception cannot be caught, the control is sent to the *finally* block, if any.

The following program illustrates this:

using System;

using System.Threading;

namespace MultithreadingApplication

{

class ThreadCreationProgram

{

public static void CallToChildThread()

{

try

{

Console.WriteLine("Child thread starts");

// do some work, like counting to 10

for (int counter = 0; counter <= 10; counter++)

{

Thread.Sleep(500);

Console.WriteLine(counter);

}

Console.WriteLine("Child Thread Completed");

}

catch (ThreadAbortException e)

{

Console.WriteLine("Thread Abort Exception");

}

finally

{

Console.WriteLine("Couldn't catch the Thread Exception");

}

}

static void Main(string[] args)

{

ThreadStart childref = new ThreadStart(CallToChildThread);

Console.WriteLine("In Main: Creating the Child thread");

Thread childThread = new Thread(childref);

childThread.Start();

//stop the main thread for some time

Thread.Sleep(2000);

//now abort the child

Console.WriteLine("In Main: Aborting the Child thread");

childThread.Abort();

Console.ReadKey();

}

}

}

When the above code is compiled and executed, it produces the following result:

In Main: Creating the Child thread

Child thread starts

0

1

2

In Main: Aborting the Child thread

Thread Abort Exception

Couldn't catch the Thread Exception

**Questions**

1. **What is the thread?**
2. **How to create the thread?**
3. **What method do you know to synchronize the threads?**
4. **What method do you know to create section of code thread safe?**
5. **What is Mutex? Semaphore?**