# Department of Software Engineering

**CS 332: Distributed Computing**

**BSCS 9AB**

**Lab 01: Multi Threading**

**CLO-1: Explain how existing distributed systems work**

**Date: January 1, 2022**

**Instructors : Dr. Farzana Jabeen**

**Submitted By:**

|  |
| --- |
| Fatima Seemab  291310  CS-9B |

**Lab 01: Multithreaded Programming**

**Introduction**

This lab is design to revise the concepts of multithreaded programming.

**Objectives**

Analyze the execution time of single thread and multi-thread program.

**Tools/Software Requirement**

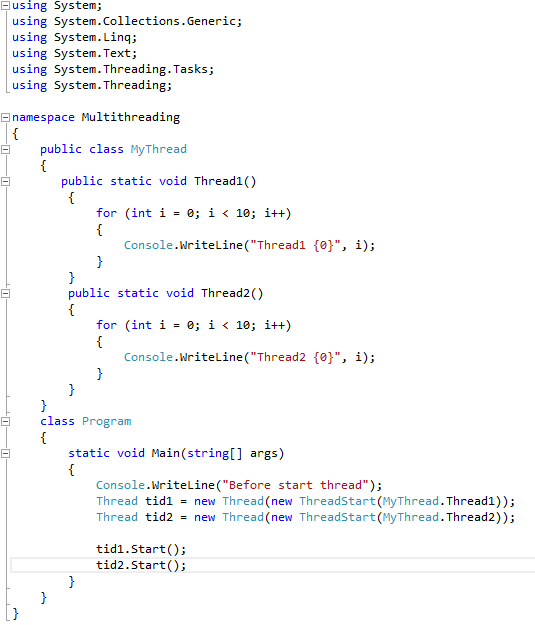
MS Visual Studio 2013 Or any other programming language

**Description**

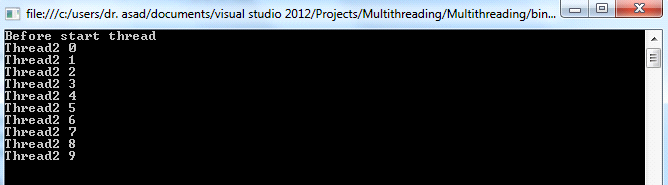
***About Threading***

Threading is a lightweight process. With the help of threads, we can decrease the processing time of the application. To use multithreading, we have to use the **Threading namespace** which is included in System. The *System.Threading* namespace includes everything we need for multi-threading.

***Example 1:***



**Output:**



Let's explore the whole program: This program has a class MyThread which has two static functions Thread1 and Thread2. To make a thread you have to make an object of class Thread. The constructor of this class takes a reference of a ThreadStart class. This constructor can send two types of exceptions; ArgumentNullException when the parameter is a null reference or a Security Exception when program does not have permission to create thread.

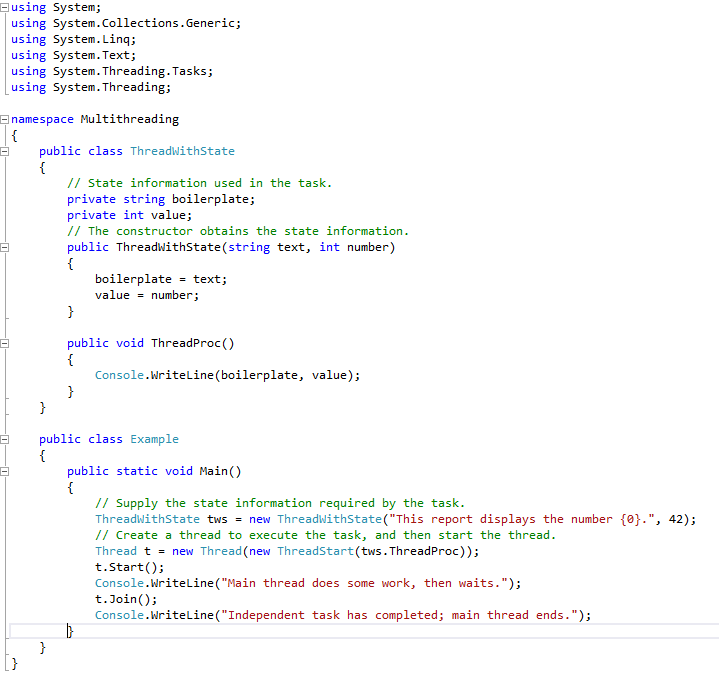
The parameter of the Thread class is reference to a ThreadStart class. ThreadStart class points to the method that should be executed first when a thread is started. The parameter is the name of the function, which is considered as a thread function. Thread1 is a static function so we give it with the name of class name without making an object of the class. The thread starts execution with Start() method of the Thread class. The output of this program is

***Observation***

* It is not a requirement that thread function must be static. We can make it a non-static function but in this case we have to create object of that class.
* It is not necessary to make two functions for making two threads. You may have only one thread function and create two threads by creating two objects of the thread class. Take a look at this program.

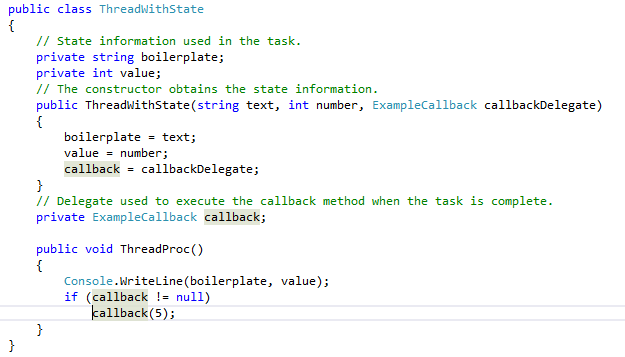
***Example 2:***

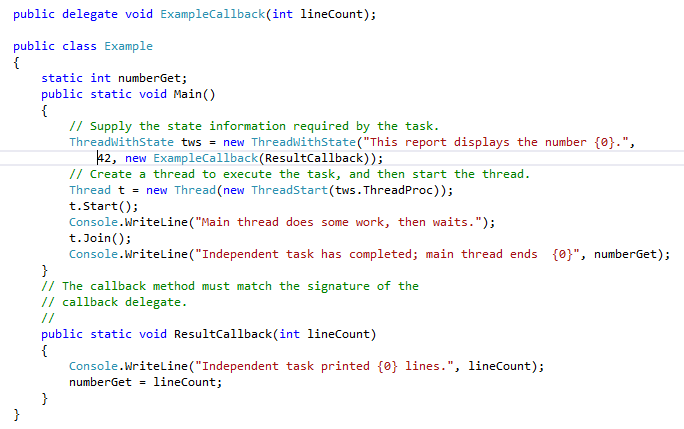
**Passing data to Thread**



**Retrieving Data with Callback Methods**

The following example demonstrates a callback method that retrieves data from a thread. The constructor for the class that contains the data and the thread method also accepts a delegate representing the callback method; before the thread method ends, it invokes the callback delegate.



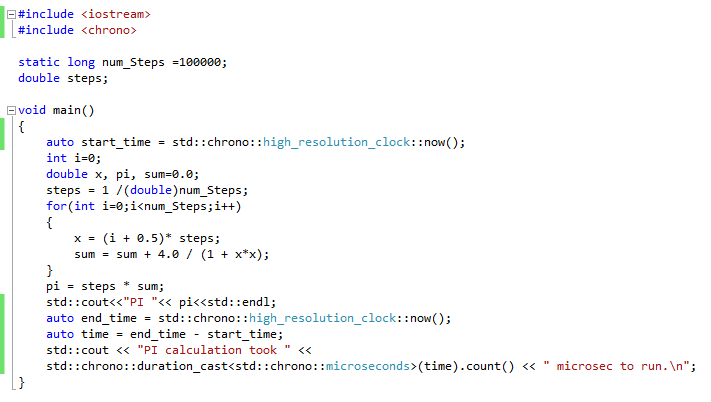


This section will contain material (e.g. examples) to help students to successfully conduct the experiment. It may contain the list of experimental steps that the students will have to go through for successful experimentation. It can guide students on common pitfalls and points to note.

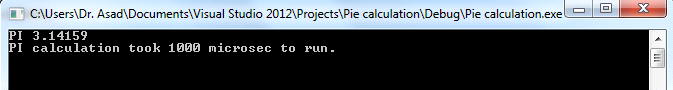
**Lab Tasks**

**Q1:** Below is the C code to calculate pi, it took 1000 microseconds to calculate and display the calculated value. Your task is as follows:

* Convert the following C code into C# or Java and calculate the execution time.
* Convert your C# /Java code into multithreaded version and analyze the execution time.
* **Is it better to use multithreaded version compared to single threaded in PI calculation?**



**Output:**



**Code:**

|  |
| --- |
| **import java.time.LocalTime;**  **class HelloWorld {**  **public static int num\_Steps=1000;**  **public static double steps;**  public static void main(String[] args)  { long start = System.nanoTime();  double x, pi;  double sum=0.0;  steps=1/(double)num\_Steps;  for(int i=0;i<num\_Steps;i++)  {  x=(i+0.5)\*steps;  sum=sum+4.0/(1+x\*x);    }  pi=steps\*sum;  System.out.println("The value of pin is "+pi);  long end = System.nanoTime();  long time=end-start;  System.out.println("The execution time is "+time);      }  } |

**Output**

|  |
| --- |
| **Graphical user interface, text, application  Description automatically generated** |

Convert your C# /Java code into multithreaded version and analyze the execution time.

**CODE:**

|  |
| --- |
| **public** **class** Main{  **public** **static** **int** *num\_Steps*=1000;  **public** **static** **double** *steps*=1/(**double**)*num\_Steps*;    **public** **static** **int** *THREADS\_COUNT*=4;  **public** **static** **double** *pi*=0.0;      **private** **static** **class** Multithreading **extends** Thread{    **public** **double** x;  **double** sum=0.0;  **int** start=0;  **int** end=0;    **public** Multithreading(**int** start,**int** end){  **this**.start=start;  **this**.end=end;  }  **public** **void** run() {  **try** {  **for**(**int** i=start;i<end;i++)  {  x=(i+0.5)\**steps*;  sum=sum+4.0/(1+x\*x);    }  *pi*=*pi*+*steps*\*sum;    } **catch** (IndexOutOfBoundsException e) {    }  }      }  **public** **static** **void** main(String[] args)  { **long** start = System.*nanoTime*();  Thread[] threads = **new** Thread[*THREADS\_COUNT*];    threads[0] = **new** Multithreading(0,*num\_Steps*/4);  threads[1] = **new** Multithreading(*num\_Steps*/4+1,*num\_Steps*/2);  threads[2] = **new** Multithreading(*num\_Steps*/2+1,3\**num\_Steps*/4);  threads[3] = **new** Multithreading(3\**num\_Steps*/4+1,*num\_Steps*);    **for** (**int** i = 0; i < *THREADS\_COUNT*; i++) {  threads[i].start();  }    **for** (**int** i = 0; i < *THREADS\_COUNT*; i++) {  **try** {  threads[i].join();  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }    System.***out***.println("The value of pi is "+*pi*);  **long** end = System.*nanoTime*();  **long** time=end-start;  System.***out***.println("The execution time is "+time);    }  } |

**Screenshot**

|  |
| --- |
| **A picture containing timeline  Description automatically generated** |

|  |
| --- |
| **Text  Description automatically generated** |

**OUTPUT: With the execution of two threads**

|  |
| --- |
| **Graphical user interface, text, application  Description automatically generated**  **Graphical user interface, text, application, email  Description automatically generated** |

**With 4 threads**

* **Is it better to use multithreaded version compared to single threaded in PI calculation?**

|  |
| --- |
| According to my code the value of pi calculated using multithreading takes more time then single thread because creation of different threads and then joining them and after that we have to perform sum on all the values calculated by different threads takes more time than simply calculating the results in a single thread. With 2 threads the time is in range of 6000000 ns whereas in case of 4 threads this time is reduced in the range of 3000000 ns but with one thread the time is small and is almost 1300000 ns |

**Note:**

To calculate Time, you can use following code or google any other library.

using System.Diagnostics;

Stopwatch stopWatch = new Stopwatch();

stopWatch.Start();

stopWatch.Stop();

TimeSpan ts = stopWatch.Elapsed;

string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:00}",

ts.Hours, ts.Minutes, ts.Seconds,

ts.Milliseconds / 10);

Console.WriteLine("RunTime " + elapsedTime);

**Deliverables**

Compile a single word file and upload on LMS.