**CS 590 – PARALLEL AND DISTRIBUTED COMPUTING**

**HOMEWORK 1**

**Name:** Ivan Sangines Escrig

**ID#:** 968606

**Instructor:** Dr. Mahmood

**Date:** September 07, 2018

**TABLE OF CONTENT**

[**INTRODUCTION**](#_Toc446970371) 3

[**SCREENSHOTS:**](#_Toc446970373) 4

[**SOURCE CODE:** 8](#_Toc446970374)

[**CONCLUSION:** 15](#_Toc446970375)

**INTRODUCTION**

The porpuse of this assignment is to get used to Visual studio and understand better the concepts explained in class about threading. This will be achieved by doing the first examples of the handout.

**SCREEN SHOTS:**

**Example 1:**

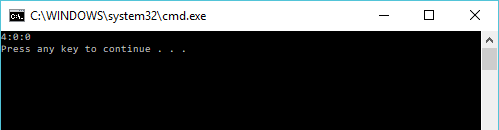
Wrong output example 1:

Here we can see the expected wrong input since there is not a good synchronization between the threads.



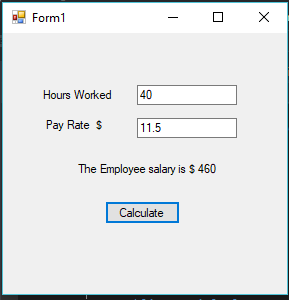
Expected output example 1:

Here we can see the right input since the threads have been synchronized properly.



**Example 2:**

Expected output since the threads are properly synchronized and the access to the different methods are correctly protected.

****

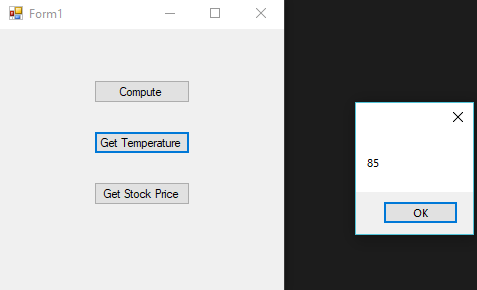
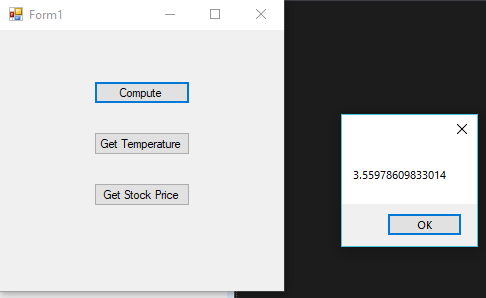
**Example 3:**

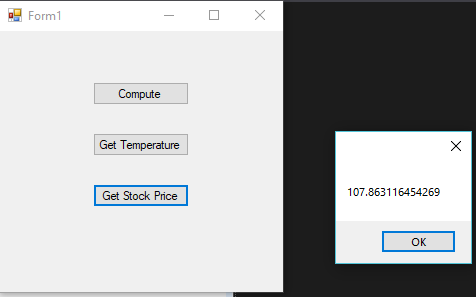
Here we can see the three different outputs for the three different operations: Compute, Get temperature and Get stock price.

Each operation has its own properties:

* Compute: It will show is the result with a delay of 5 seconds as it is supposed to be the time that it takes to get the data.
* Get Temperature: It will give the result immediately but the output will be different every two seconds.
* Get Stock price: It will show the output immediately but after three seconds, the output will be different.

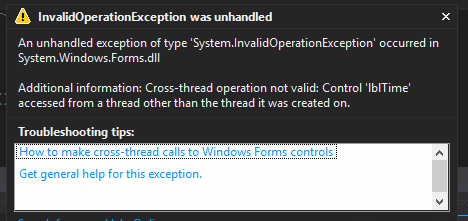
Once you close the program, all threads are stopped properly, including the one working on the background.



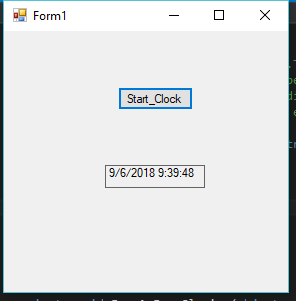


**Example 4:**

Here we can see the expected Threading Cross error since we are trying to update a label that is not properly defined in the thread that is trying to update it.



Expected output:



**SOURCE CODE:**

**Example 1:**

Main:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace MultiThreading

{

class Program

{

static void Main(string[] args)

{

MyTime mt = new MyTime();

Thread th1 = new Thread(new ThreadStart(mt.IncrementTime));

Thread th2 = new Thread(new ThreadStart(mt.ReadTime));

th1.Start();

Thread.Sleep(50);

th2.Start();

}

}

}

MyTime class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace MultiThreading

{

class MyTime

{

int hours = 3;

int mins = 59;

int secs = 59;

// any reference type can be used as a lock

object olock = new object();

public void IncrementTime()

{

lock (olock)

{

secs = secs + 1;

if (secs == 60)

{

secs = 0;

mins = mins + 1;

if (mins == 60)

{

mins = 0;

Thread.Sleep(100); // preempt

hours = hours + 1;

}

}

}

}

public void ReadTime()

{

lock (olock)

{

Console.WriteLine(hours.ToString() +

":" + mins.ToString() +

":" + secs.ToString());

}

}

}

}

**Example 2:**

Employee Class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Threading2

{

class Employee

{

object olock1 = new object();

object olock2 = new object();

int hoursWorked;

public int HoursWorked

{

get { lock (olock1) { return hoursWorked; } }

set { lock (olock1) { hoursWorked = value; } }

}

double payRate;

public double PayRate

{

get { lock (olock2) { return payRate; } }

set { lock (olock2) { payRate = value; } }

}

public double ComputePay()

{

//lock (olock1)

//{

// return hoursWorked \* payRate;

//}

double hours = 0;

lock (olock1)

{

hours = hoursWorked;

}

double rate = 0;

lock (olock2)

{

rate = payRate;

}

return hours \* rate;

}

}

}

Form:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Threading2

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

Employee emp = new Employee();

emp.HoursWorked = Convert.ToInt32(textBox1.Text);

emp.PayRate = Convert.ToDouble(textBox2.Text);

label3.Text = "The Employee salary is $ " + emp.ComputePay();

}

}

}

**Example 3:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace Operations

{

class LongOperations

{

object olock = new object();

bool terminate = false;

public bool Terminate

{

set { lock (olock) { terminate = value; } }

}

double stockPrice;

public double StockPrice

{

get { lock (olock) { return stockPrice; } }

}

int temperature;

public int Temperature

{

get { lock (olock) { return temperature; } }

}

int data;

public int Data

{

get { lock (olock) { return data; } }

set { lock (olock) { data = value; } }

}

double result;

public double Result

{

get { lock (olock) { return result; } }

}

public void Compute()

{

double res = 2.7575 + data / 3.8686 - 5.66;

Thread.Sleep(5000); // assume it takes 5 secs

lock (olock) // to produce result

{

result = res;

}

}

public void GetWeather()

{

Random rn = new Random((int)DateTime.Now.Ticks);

while (true)

{

// suppose we are obtaining temperatur info

// from a web service and it takes 2 seconds

// to get data from it

Thread.Sleep(2000);

lock (olock) { temperature = 80 + rn.Next(2, 8); }

}

}

public void GetStockPrice()

{

Random rn = new Random((int)DateTime.Now.Ticks);

bool term = false;

lock (olock) { term = terminate; }

while (term == false)

{

Thread.Sleep(3000); // pretend it takes

// 3 seconds to get the stock price

lock (olock)

{

stockPrice = 100 + rn.NextDouble() \* 10;

}

Thread.Sleep(10); // pretend it takes

// 10 milliseconds to write to DB

lock (olock) { term = terminate; }

}

}

}

}

Form code:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Operations

{

public partial class Form1 : Form

{

Thread thCompute = null;

Thread thWeather = null;

Thread thStock = null;

LongOperations lop = null;

public Form1()

{

InitializeComponent();

//Form1\_Load(this, null);

}

private void button1\_Click(object sender, EventArgs e)

{

//Form1\_Load(this, null);

lop.Data = 25;

thCompute = new Thread(new ThreadStart(lop.Compute));

thCompute.Start();

thCompute.Join(); // wait for thread to finish

double res = lop.Result;

MessageBox.Show(res.ToString());

}

private void button2\_Click(object sender, EventArgs e)

{

//Form1\_Load(this, null);

MessageBox.Show(lop.Temperature.ToString());

}

private void Form1\_Load(object sender, EventArgs e)

{

lop = new LongOperations();

thWeather = new Thread(new ThreadStart(lop.GetWeather));

// for long running threads that can be

// stopped halfway without causing any

// harm, set them as background threads

// Background threads get terminated

// automatically when the parent process

// is terminated

thWeather.IsBackground = true;

thWeather.Start();

thStock = new Thread(new ThreadStart(lop.GetStockPrice));

thStock.Start();

}

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

lop.Terminate = true;

if (thStock != null)

{

if (thStock.IsAlive)

{

thStock.Join(); // wait for thread to finish

}

}

if (thCompute != null)

{

if (thCompute.IsAlive)

thCompute.Join();

}

}

private void button3\_Click(object sender, EventArgs e)

{

//Form1\_Load(this, null);

//Form1\_FormClosing(this, null);

MessageBox.Show(lop.StockPrice.ToString());

}

}

}

**Example 4:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace CrossThreading

{

delegate void UpdateDel(string msg);

public partial class Form1 : Form

{

Thread thClock = null;

UpdateDel udel;

bool terminate = false;

public Form1()

{

InitializeComponent();

udel = new UpdateDel(UpdateLabel);

}

private void btnClock\_Click(object sender, EventArgs e)

{

thClock = new Thread(new ThreadStart(MyClock));

thClock.Start();

}

void MyClock()

{

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

{

// lblTime.Text = DateTime.Now.ToString();

// we cannot update the label because

// that will cause cross threading

// error, use invoke on the UI element

if (terminate == false)

lblTime.Invoke(udel, new string[] { DateTime.Now.ToString() });

Thread.Sleep(1000);

}

}

void UpdateLabel(string msg)

{

lblTime.Text = msg;

}

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

terminate = true;

if (thClock != null)

{

if (thClock.IsAlive)

thClock.Join();

}

}

}

}

**Conclusion:**

Since I am not very used to work with threads, this assignment helped me to understand how threads work and the importance of protect and synchronize different threads or methods. Also, it helped me to get familiarized with Visual Studio.