**CS 590 – PARALLEL AND DISTRIBUTED COMPUTING**

**HOMEWORK 3**

**Name:** Ivan Sangines Escrig

**ID#:** 968606

**Instructor:** Dr. Mahmood

**Date:** October 2, 2018

**TABLE OF CONTENT**

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

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

[**SOURCE CODE:**](#_Toc446970374) 14

[**CONCLUSION:**](#_Toc446970375) 40

**INTRODUCTION**

The porpuse of this assignment is to get familiar with the Task Parallel Library. This will be achieved by doing the examples of the handout. After doing this examples I will have more knowledge about how to create Tasks passing lambda expressions, actions or even objects. Also, I will improve my skills with the delegates and threads.

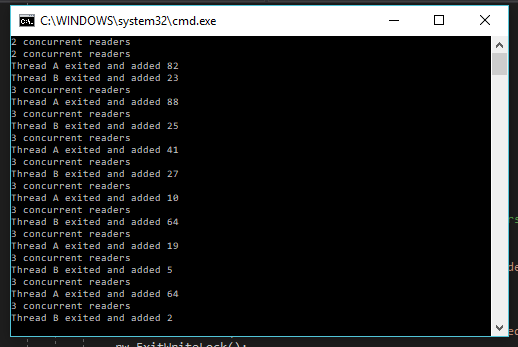
**SCREEN SHOTS:**

**ReaderWriteLock Example :**

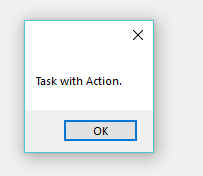
As mentioned in class, one of the delicate situations on multithreading is when we create a program where we are reading and writing data using different threads (working on same data using different threads). The problem of this situation is not reading, since it will not change the data, if all the threads are reading, there is no need to lock or protect our data. However, the problem comes whenever we are writing or updating the data and we want to read at the same time. It is in that situation when we have to protect our data using locks or synchronizing our threads.

C# provides the ReaderWriterLock class in order to handle that situation. The difference between locks and ReaderWriterLock is that the lock allows just one thread to work on the data (no matter if it is read or write). However, the ReaderWriterLock allows just one thread to write but many other threads to read the data. This is perfect for situations where we need to read more than write.

In this example we are able to see how most of the time the ReaderWriterLock allows more than one Reader all the time (3 Readers most of the time in this example). We have two infinite while loops, the reader loop simulates that three threads are reading the data the whole time. The write loop, shows how many readers we have before writing (3 most of the time), then one of the writer threads adds a random number and prints the threadID with the number added, when finished writing.

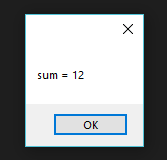
****

**Example 1:**

****

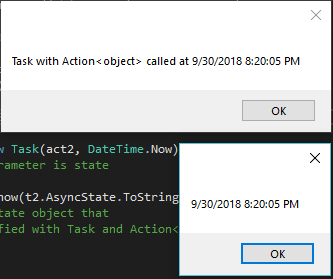
**Example 2:**

Here we can see the output of a sum using . In the load action of the form we are using Wait(), that means that the for will not show up until the task is done (we click OK or X button).

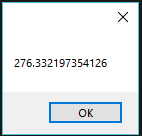
****

**Example 3:**

Here we can see two simultaneous outputs. One called through the task using Action <object> and the other one using the state of the object declared in the task.

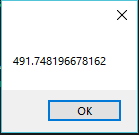
****

**Example 4:**

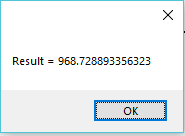


**Example 5:**

The first output is obtained by directly passing the object and state to the Task.

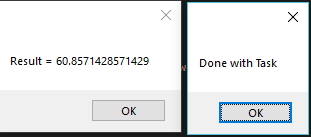
****

The second output is using a returning task double and using a lambda expression instead of passing any object.

****

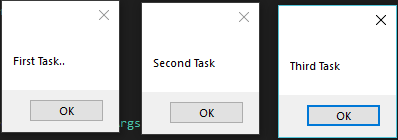
**Example 6:**

In this example we are using task factory and creating a task with startNew() and a second task to compute the result. The result is going first due the Sleep on the taskFactory

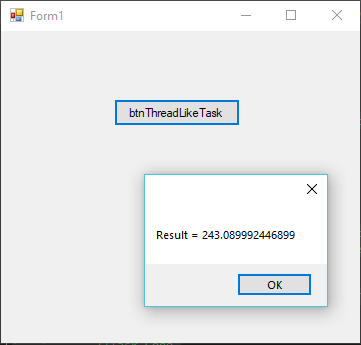
****

**Example 7:**

Running multiple tasks using lambda expressions

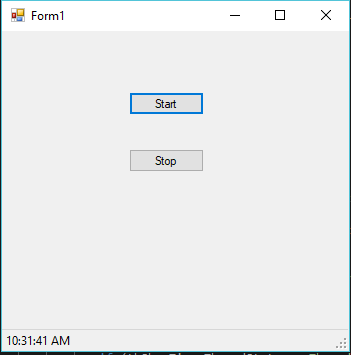
****

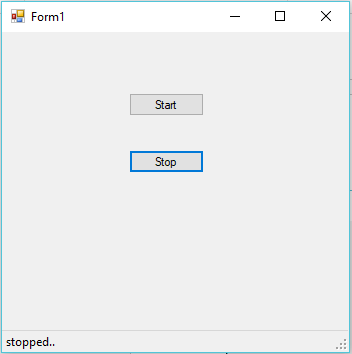
**Example TPL:**

****

**Example 9:**

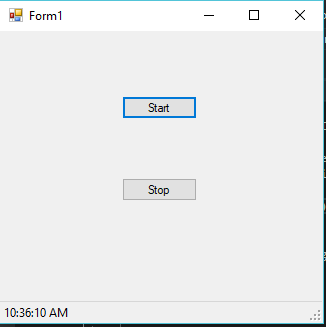
Using delegates and Threads we are able to start s StopWatch whenever we click on the button start. The time is displayed on the status label. Once we click stop, the time is stopped and the lable Text updated with the stopped msg.

****

****

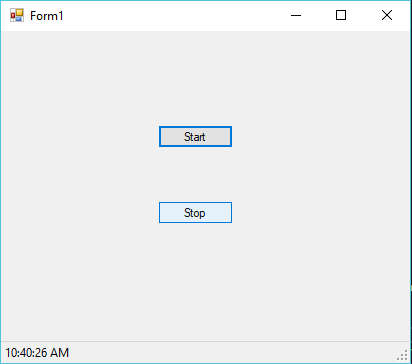
**Example 10:**

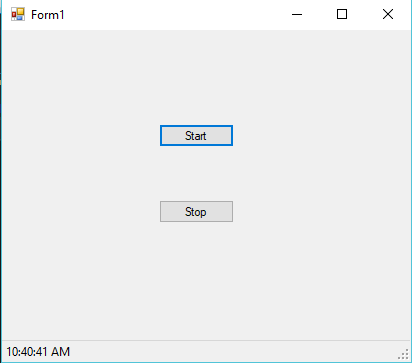
In this example we are doing the same as example 9 but using Tasks instead of threads.

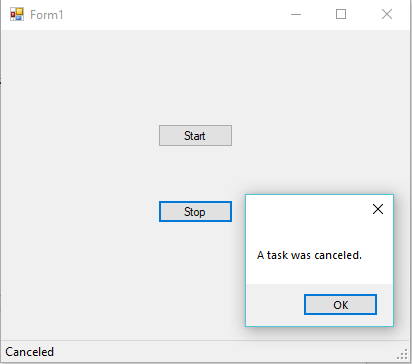
****

****

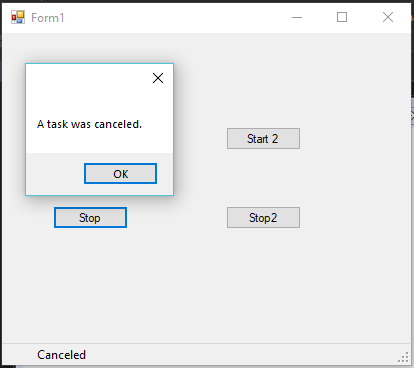
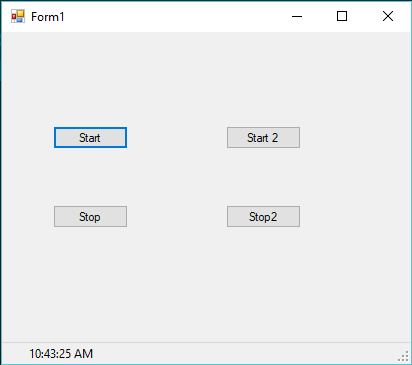
**Example 11:**

****

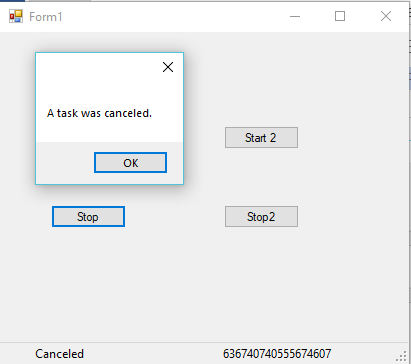
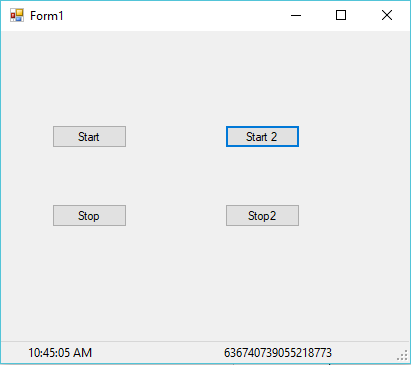
****

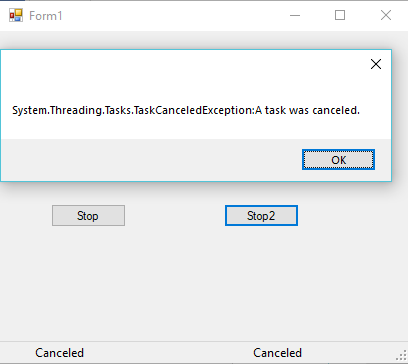
****

**Example 12:**

****

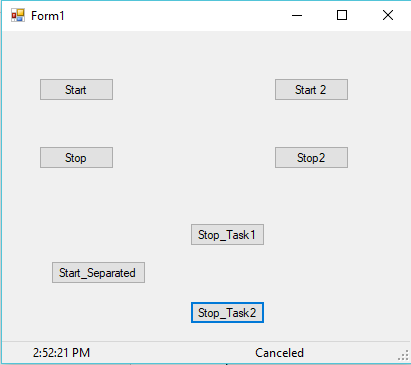
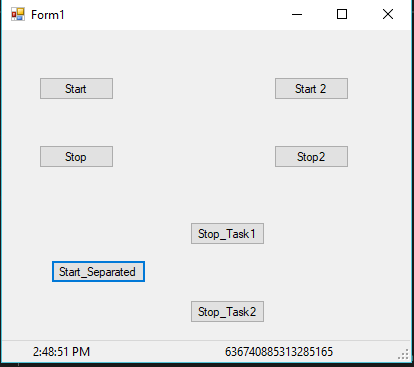
I have changed the second part of this example since there was some problem while trying to fetch data from the webpage provided. I changed the second task, which is giving the actual time in ticks instead of fetching data from the stocks page. Clicking the button start2, two tasks start at the same time task1 returns the actual time in a proper format, task2 returns the actual time in ticks. Once both tasks are running, we can either just stop the first one with the Stop button or stop both of them with the Stop2 button.

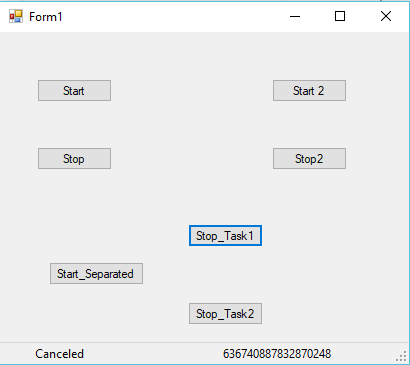
****

****

**Example 13:**

This example incorporates the last two, where we have a Start and Stop button to start first task (update the clock every 5 seconds) and finish it. Then we have the Start2 and Stop2 which start and stop two Tasks at the same time. The third part of this example is the start separated button, it will start two task but they are not depending on each other. That means we can stop one at a time. This can be achieve by starting each task separately and using different tokens to stop them.





**SOURCE CODE:**

**Example ReaderWriteLock :**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace ReaderWriterLock

{

class Program

{

static ReaderWriterLockSlim \_rw = new ReaderWriterLockSlim();

static List<int> \_items = new List<int>();

static Random \_rand = new Random();

static void Main(string[] args)

{

new Thread(Read).Start();

new Thread(Read).Start();

new Thread(Read).Start();

new Thread(Write).Start("A");

new Thread(Write).Start("B");

}

static void Read()

{

while (true)

{

\_rw.EnterReadLock();

foreach (int i in \_items) Thread.Sleep(10);

\_rw.ExitReadLock();

}

}

static void Write(object threadID)

{

//Console.WriteLine(\_rw.CurrentReadCount + " concurrent readers");

while (true)

{

Console.WriteLine(\_rw.CurrentReadCount + " concurrent readers");

int newNumber = GetRandNum(100);

\_rw.EnterWriteLock();

\_items.Add(newNumber);

Console.WriteLine("Thread " + threadID + " exited and added " + newNumber);

\_rw.ExitWriteLock();

Thread.Sleep(1000);

}

}

static int GetRandNum(int max) { lock (\_rand) return \_rand.Next(max); }

}

}

**Example 1:**

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 Task1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

//USING REGULAR EXPRESSIONS

/\*

Action act = new Action(DoSomething);

Task t = new Task(DoSomething);

t.Start();

void DoSomething()

{

// some task, e.g., update DB

MessageBox.Show("Task done..");

}

\*/

//USING LAMBDA EXPRESSION

Action act1 = new Action(() => MessageBox.Show("Task with Action."));

Task t1 = new Task(act1);

t1.Start();

}

}

}

**Example 2:**

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 Example2

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

void DoAdd(int a, int b)

{

int sum = a + b;

MessageBox.Show("sum = " + sum.ToString());

}

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

{

Action<int, int> actAdd = new Action<int, int>(this.DoAdd);

Task t1a = new Task(() => actAdd(5, 7));

t1a.Start();

t1a.Wait(); //shows the result of the sum and not the form until task is done

}

}

}

**Example 3:**

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 Example3

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

Action<Object> act2 = new Action<Object>(

(obj) => {

MessageBox.Show("Task with Action<object> called at " +

obj.ToString());

//DateTime dt = (DateTime)obj;

//MessageBox.Show(dt.Ticks.ToString());

});

Task t2 = new Task(act2, DateTime.Now);

// second parameter is state

t2.Start();

MessageBox.Show(t2.AsyncState.ToString());

// get the state object that

// was specified with Task and Action<object>

}

}

}

**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.Tasks;

using System.Windows.Forms;

namespace Example4

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

MyCompute mc = new MyCompute();

Func<float, float, double> fptr = new Func<float, float, double>(mc.Compute2);

// also OK, We would be using Func deligate:

// var task = new Task<double>(() => fptr(23.67f,12.66f));

var task = new Task<double>(() => mc.Compute2(23.67f, 12.66f));

task.Start();

task.Wait(); // wait for task to complete

MessageBox.Show(task.Result.ToString());

}

}

}

**CLASS:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

namespace Example4

{

class MyCompute

{

public double Compute(object p)

{

double result = (float)p \* 21.77 - 23.33;

Thread.Sleep(5000);

return result;

}

public double Compute2(float a, float b)

{

double result = a \* b - 23.33;

Thread.Sleep(1000);

return result;

}

}

}

**Example 5:**

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 Example5

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

MyCompute mc = new MyCompute();

Func<object, double> fptr = new Func<object, double>(mc.Compute);

Task<double> tCompute = new Task<double>(fptr, 23.66f);

// second parameter is the state object to pass

tCompute.Start();

MessageBox.Show(tCompute.Result.ToString()); // blocking call

// you can instead create a lambda that takes no parameter and

// calls your function

Task<double> tCompute2 = new Task<double>(() => mc.Compute(45.57f));

// <double> in above line indicates that the return value from

// Task is double

tCompute2.Start();

double res = tCompute2.Result;

MessageBox.Show("Result = " + res.ToString());

}

}

}

**CLASS:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

namespace Example5

{

class MyCompute

{

public double Compute(object p)

{

double result = (float)p \* 21.77 - 23.33;

Thread.Sleep(5000);

return result;

}

public double Compute2(float a, float b)

{

double result = a \* b - 23.33;

Thread.Sleep(1000);

return result;

}

}

}

**Example 6:**

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.Threading;

using System.Windows.Forms;

namespace Example6

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

Task t1 = Task.Factory.StartNew(() => { Thread.Sleep(3000); MessageBox.Show("Done with Task"); });

// Getting a result from a task that is created via StartNew

Task<double> t2 = Task.Factory.StartNew<double>(() =>

{

double avg = 0;

int[] data = { 25, 78, 32, 91, 88, 67, 45 };

avg = data.Average();

return avg;

});

double res = t2.Result;

MessageBox.Show("Result = " + res.ToString());

// this is output before t1 task because of Sleep call in t1

}

}

}

**Example 7:**

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.Threading;

using System.Windows.Forms;

namespace Example7

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

Parallel.Invoke(

() => MessageBox.Show("First Task.."),

() => { Thread.Sleep(3000); MessageBox.Show("Second Task"); },

() => { Thread.Sleep(4000); MessageBox.Show("Third Task"); }

);

}

}

}

**Example TPL:**

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 TPLTest

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

/\*

// MultiThreading code

MyCompute mc = new MyCompute();

mc.Data1 = 24.22f;

mc.Data2 = 11;

Action fCompute = new Action(mc.Compute3);

Thread thCompute = new Thread(new ThreadStart(fCompute));

thCompute.Start();

thCompute.Join();

MessageBox.Show("Result = " + mc.Res.ToString());

\*/

//Task Code

MyCompute mc = new MyCompute();

mc.Data1 = 24.22f;

mc.Data2 = 11;

Action fCompute = new Action(mc.Compute3);

Task t1 = new Task(fCompute);

t1.Start();

t1.Wait();

MessageBox.Show("Result = " + mc.Res.ToString());

}

}

}

**CLASS**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace TPLTest

{

class MyCompute

{

object oLock = new object();

double res;

public double Res

{

get

{

double ress = 0;

lock (oLock) { ress = res; }

return ress;

}

}

float data1;

public float Data1

{

get { float d1 = 0; lock (oLock) { d1 = data1; } return d1; }

set { lock (oLock) { data1 = value; } }

}

int data2;

public int Data2

{

get { int d2 = 0; lock (oLock) { d2 = data2; } return d2; }

set { lock (oLock) { data2 = value; } }

}

public void Compute3()

{

double result = 0;

lock (oLock) { result = data1 \* data2 - 23.33; }

Thread.Sleep(1000);

lock (oLock) { res = result; }

}

}

}

**Example 9:**

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 Example9

{

public partial class Form1 : Form

{

bool bTerminate = false;

Thread thShowTime = null;

DelShowTime delShowTime = null;

delegate void DelShowTime(string msg);

public Form1()

{

InitializeComponent();

}

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

{

bTerminate = false;

if (thShowTime == null)

thShowTime = new Thread(new ThreadStart(this.ShowTime));

if (thShowTime.ThreadState == ThreadState.Stopped)

thShowTime = new Thread(new ThreadStart(this.ShowTime));

thShowTime.Start();

}

void ShowTime()

{

while (bTerminate == false)

{

if (statusStrip1.InvokeRequired)

{

if (bTerminate == false)

statusStrip1.Invoke(delShowTime, new string[] { DateTime.Now.ToLongTimeString() });

}

Thread.Sleep(1000);

}

}

void UpdateTime(string msg)

{

lblStatus.Text = msg;

}

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

{

delShowTime = new DelShowTime(this.UpdateTime);

}

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

{

bTerminate = true;

if (thShowTime != null)

{

if (thShowTime.ThreadState == ThreadState.Running)

thShowTime.Join();

}

lblStatus.Text = "stopped.. ";

}

}

}

**Example 10:**

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 Example10

{

public partial class Form1 : Form

{

bool bTerminate = false;

DelShowTime delShowTime = null;

delegate void DelShowTime(string msg);

Task tskShowTime = null;

public Form1()

{

InitializeComponent();

}

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

{

bTerminate = false;

if (tskShowTime == null)

tskShowTime = new Task(ShowTime);

if (tskShowTime.IsCompleted)

tskShowTime = new Task(ShowTime);

tskShowTime.Start();

}

void ShowTime()

{

while (bTerminate == false)

{

if (statusStrip1.InvokeRequired)

{

if (bTerminate == false)

statusStrip1.Invoke(delShowTime, new string[] { DateTime.Now.ToLongTimeString() });

}

Thread.Sleep(1000);

}

}

void UpdateTime(string msg)

{

lblStatus.Text = msg;

}

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

{

bTerminate = true;

if (tskShowTime != null)

{

if (tskShowTime.Status == TaskStatus.Running)

tskShowTime.Wait();

}

lblStatus.Text = "stopped.. ";

}

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

{

delShowTime = new DelShowTime(this.UpdateTime);

}

}

}

**Example 11:**

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 Example11

{

public partial class Form1 : Form

{

DelShowTime delShowTime = null;

delegate void DelShowTime(string msg);

Task tskShowTime = null;

CancellationTokenSource cancelTokenSource = null;

public Form1()

{

InitializeComponent();

}

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

{

cancelTokenSource = new CancellationTokenSource();

CancellationToken cancelToken = cancelTokenSource.Token;

if (tskShowTime == null)

tskShowTime = new Task(() => ShowTime(cancelToken), cancelToken);

if (tskShowTime.IsCompleted)

tskShowTime = new Task(() => ShowTime(cancelToken), cancelToken);

tskShowTime.Start();

}

void ShowTime(CancellationToken canToken)

{

//while (bTerminate == false)

while (true)

{

if (statusStrip1.InvokeRequired)

{

// if (bTerminate == false)

statusStrip1.Invoke(delShowTime, new string[] { DateTime.Now.ToLongTimeString() });

}

/\*Thread.Sleep(1000);

if (canToken.IsCancellationRequested == true)

throw new OperationCanceledException(canToken); \*/

canToken.WaitHandle.WaitOne(15000);

canToken.ThrowIfCancellationRequested();

}

}

void UpdateTime(string msg)

{

lblStatus.Text = msg;

}

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

{

delShowTime = new DelShowTime(this.UpdateTime);

}

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

{

try

{

cancelTokenSource.Cancel(); // cancel tasks

if (tskShowTime != null)

{

if (tskShowTime.Status == TaskStatus.Running)

tskShowTime.Wait();

}

}

catch (AggregateException ex) // exception in parallel framework

{

lblStatus.Text = tskShowTime.Status.ToString();

foreach (var ee in ex.InnerExceptions)

{ // important if many tasks are being cancelled

MessageBox.Show(ee.Message);

}

}

}

}

}

**Example 12:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Net;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Example11

{

public partial class Form1 : Form

{

DelShow delShowTime = null;

DelShow delShowStockPrice = null;

delegate void DelShow(string msg);

Task tskShowTime = null;

Task tskTicks = null;

CancellationTokenSource cancelTokenSource = null;

public Form1()

{

InitializeComponent();

}

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

{

cancelTokenSource = new CancellationTokenSource();

CancellationToken cancelToken = cancelTokenSource.Token;

if (tskShowTime == null)

tskShowTime = new Task(() => ShowTime(cancelToken), cancelToken);

if (tskShowTime.IsCompleted)

tskShowTime = new Task(() => ShowTime(cancelToken), cancelToken);

tskShowTime.Start();

}

void ShowTime(CancellationToken canToken)

{

while (true)

{

try

{

if (statusStrip1.InvokeRequired)

{

statusStrip1.Invoke(delShowTime, new string[] { DateTime.Now.ToLongTimeString() });

}

canToken.WaitHandle.WaitOne(5000);

canToken.ThrowIfCancellationRequested();

}

catch (Exception)

{

throw;

}

}

}

void ShowTicks(CancellationToken canToken)

{

while (true)

{

try

{

if (statusStrip1.InvokeRequired)

{

statusStrip1.Invoke(delShowStockPrice, new string[] { DateTime.Now.Ticks.ToString() });

}

canToken.WaitHandle.WaitOne(5000);

canToken.ThrowIfCancellationRequested();

//THIS WILL PRODUCE A LOCKDEAD

//canToken.ThrowIfCancellationRequested();

//canToken.WaitHandle.WaitOne(15000);

}

catch (Exception)

{

throw;

}

}

}

void UpdateTime(string msg)

{

lblStatus.Text = msg;

}

void UpdateTicks(string msg)

{

lblStat2.Text = msg;

}

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

{

delShowTime = new DelShow(this.UpdateTime);

delShowStockPrice = new DelShow(this.UpdateTicks);

}

private void btnStart2Tasks\_Click(object sender, EventArgs e)

{

// ShowTime task

if (tskShowTime == null)

{

cancelTokenSource = new CancellationTokenSource();

CancellationToken cancelToken1 = cancelTokenSource.Token;

tskShowTime = new Task(() => ShowTime(cancelToken1), cancelToken1);

}

if (tskShowTime.IsCompleted)

{

cancelTokenSource = new CancellationTokenSource();

CancellationToken cancelToken1 = cancelTokenSource.Token;

tskShowTime = new Task(() => ShowTime(cancelToken1), cancelToken1);

}

if (tskShowTime.Status != TaskStatus.Running)

tskShowTime.Start();

// Ticks task

if (tskTicks == null)

{

CancellationToken cancelToken2 = cancelTokenSource.Token;

tskTicks = new Task(() => ShowTicks(cancelToken2), cancelToken2);

}

if (tskTicks.IsCompleted)

{

CancellationToken cancelToken2 = cancelTokenSource.Token;

tskTicks = new Task(() => ShowTicks(cancelToken2), cancelToken2);

}

if (tskTicks.Status != TaskStatus.Running)

tskTicks.Start();

}

/\* I THINK THERE IS SOME KIND OF PROBLEM WITH THE WEBSITE, USING SHOWTICKS() INSTEAD OF SHOWSTOCKPRICE()

void ShowStockPrice(CancellationToken canToken)

{

while (true)

{

try

{

WebClient wbc = new WebClient();

byte[] bdata = wbc.DownloadData("http://www.nasdaq.com/aspx/infoquotes.aspx?symbol=IBM&selected=IBM");

string pageText = new UTF8Encoding().GetString(bdata);

int pos1 = pageText.IndexOf("LastSale1'>");

int pos2 = pageText.IndexOf("</", pos1 + 1);

string price = pageText.Substring(pos1 + 18, pos2 - pos1 - 18);

if (statusStrip1.InvokeRequired)

{

statusStrip1.Invoke(delShowStockPrice, new string[] { price });

}

canToken.WaitHandle.WaitOne(1000);

// this has to be before

// ThrowIfCancellationRequested, otherwise causes deadlock

// if cancellation is not done in a separate task

canToken.ThrowIfCancellationRequested();

}

catch (Exception)

{

throw;

}

}

}

\*/

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

{

try

{

cancelTokenSource.Cancel(); // cancel tasks

if (tskShowTime != null)

{

if (tskShowTime.Status == TaskStatus.Running)

tskShowTime.Wait();

}

}

catch (AggregateException ex) // exception in parallel framework

{

lblStatus.Text = tskShowTime.Status.ToString();

foreach (var ee in ex.InnerExceptions)

{ // important if many tasks are being cancelled

MessageBox.Show(ee.Message);

}

}

}

private void btnCancel2Tasks\_Click(object sender, EventArgs e)

{

Task tCancel = new Task(CancelAllTasks);

tCancel.Start();

/\*

try

{

cancelTokenSource.Cancel(); // cancel tasks

Task.WaitAll(tskShowTime, tskTicks);

// takes variable number of task objects or a task array

}

catch (AggregateException ex)

{

lblStatus.Text = tskShowTime.Status.ToString();

lblStat2.Text = tskTicks.Status.ToString();

foreach (var ee in ex.InnerExceptions)

{

MessageBox.Show(ee.GetType() + ":" + ee.Message);

}

}

\*/

}

void CancelAllTasks()

{

try

{

cancelTokenSource.Cancel(); // cancel tasks

Task.WaitAll(tskShowTime, tskTicks);

}

catch (AggregateException ex) // parallel framework excep.

{

statusStrip1.Invoke(delShowTime, new string[] { tskShowTime.Status.ToString() });

statusStrip1.Invoke(delShowStockPrice, new string[] { tskTicks.Status.ToString() });

foreach (var ee in ex.InnerExceptions)

{

MessageBox.Show(ee.GetType() + ":" + ee.Message);

}

}

}

}

}

**Example 13:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Net;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Example11

{

public partial class Form1 : Form

{

DelShow delShowTime = null;

DelShow delShowTicks = null;

delegate void DelShow(string msg);

Task tskShowTime = null;

Task tskTicks = null;

CancellationTokenSource cancelTokenSource = null;

CancellationTokenSource cancelTokenSourceShowTime = null;

CancellationTokenSource cancelTokenSourceShowStockPrice = null;

public Form1()

{

InitializeComponent();

}

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

{

cancelTokenSource = new CancellationTokenSource();

CancellationToken cancelToken = cancelTokenSource.Token;

if (tskShowTime == null)

tskShowTime = new Task(() => ShowTime(cancelToken), cancelToken);

if (tskShowTime.IsCompleted)

tskShowTime = new Task(() => ShowTime(cancelToken), cancelToken);

tskShowTime.Start();

}

void ShowTime(CancellationToken canToken)

{

while (true)

{

try

{

if (statusStrip1.InvokeRequired)

{

statusStrip1.Invoke(delShowTime, new string[] { DateTime.Now.ToLongTimeString() });

}

canToken.WaitHandle.WaitOne(5000);

canToken.ThrowIfCancellationRequested();

}

catch (Exception)

{

throw;

}

}

}

void ShowTicks(CancellationToken canToken)

{

while (true)

{

try

{

if (statusStrip1.InvokeRequired)

{

statusStrip1.Invoke(delShowTicks, new string[] { DateTime.Now.Ticks.ToString() });

}

canToken.WaitHandle.WaitOne(5000);

canToken.ThrowIfCancellationRequested();

//THIS WILL PRODUCE A LOCKDEAD

//canToken.ThrowIfCancellationRequested();

//canToken.WaitHandle.WaitOne(15000);

}

catch (Exception)

{

throw;

}

}

}

void UpdateTime(string msg)

{

lblStatus.Text = msg;

}

void UpdateTicks(string msg)

{

lblStat2.Text = msg;

}

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

{

delShowTime = new DelShow(this.UpdateTime);

delShowTicks = new DelShow(this.UpdateTicks);

}

private void btnStart2Tasks\_Click(object sender, EventArgs e)

{

// ShowTime task

if (tskShowTime == null)

{

cancelTokenSource = new CancellationTokenSource();

CancellationToken cancelToken1 = cancelTokenSource.Token;

tskShowTime = new Task(() => ShowTime(cancelToken1), cancelToken1);

}

if (tskShowTime.IsCompleted)

{

cancelTokenSource = new CancellationTokenSource();

CancellationToken cancelToken1 = cancelTokenSource.Token;

tskShowTime = new Task(() => ShowTime(cancelToken1), cancelToken1);

}

if (tskShowTime.Status != TaskStatus.Running)

tskShowTime.Start();

// Ticks task

if (tskTicks == null)

{

CancellationToken cancelToken2 = cancelTokenSource.Token;

tskTicks = new Task(() => ShowTicks(cancelToken2), cancelToken2);

}

if (tskTicks.IsCompleted)

{

CancellationToken cancelToken2 = cancelTokenSource.Token;

tskTicks = new Task(() => ShowTicks(cancelToken2), cancelToken2);

}

if (tskTicks.Status != TaskStatus.Running)

tskTicks.Start();

}

/\* I THINK THERE IS SOME KIND OF PROBLEM WITH THE WEBSITE, USING SHOWTICKS() INSTEAD OF SHOWSTOCKPRICE()

void ShowStockPrice(CancellationToken canToken)

{

while (true)

{

try

{

WebClient wbc = new WebClient();

byte[] bdata = wbc.DownloadData("http://www.nasdaq.com/aspx/infoquotes.aspx?symbol=IBM&selected=IBM");

string pageText = new UTF8Encoding().GetString(bdata);

int pos1 = pageText.IndexOf("LastSale1'>");

int pos2 = pageText.IndexOf("</", pos1 + 1);

string price = pageText.Substring(pos1 + 18, pos2 - pos1 - 18);

if (statusStrip1.InvokeRequired)

{

statusStrip1.Invoke(delShowStockPrice, new string[] { price });

}

canToken.WaitHandle.WaitOne(1000);

// this has to be before

// ThrowIfCancellationRequested, otherwise causes deadlock

// if cancellation is not done in a separate task

canToken.ThrowIfCancellationRequested();

}

catch (Exception)

{

throw;

}

}

}

\*/

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

{

try

{

cancelTokenSource.Cancel(); // cancel tasks

if (tskShowTime != null)

{

if (tskShowTime.Status == TaskStatus.Running)

tskShowTime.Wait();

}

}

catch (AggregateException ex) // exception in parallel framework

{

lblStatus.Text = tskShowTime.Status.ToString();

foreach (var ee in ex.InnerExceptions)

{ // important if many tasks are being cancelled

MessageBox.Show(ee.Message);

}

}

}

private void btnCancel2Tasks\_Click(object sender, EventArgs e)

{

Task tCancel = new Task(CancelAllTasks);

tCancel.Start();

/\*

try

{

cancelTokenSource.Cancel(); // cancel tasks

Task.WaitAll(tskShowTime, tskTicks);

// takes variable number of task objects or a task array

}

catch (AggregateException ex)

{

lblStatus.Text = tskShowTime.Status.ToString();

lblStat2.Text = tskTicks.Status.ToString();

foreach (var ee in ex.InnerExceptions)

{

MessageBox.Show(ee.GetType() + ":" + ee.Message);

}

}

\*/

}

void CancelAllTasks()

{

try

{

cancelTokenSource.Cancel(); // cancel tasks

Task.WaitAll(tskShowTime, tskTicks);

}

catch (AggregateException ex) // parallel framework excep.

{

statusStrip1.Invoke(delShowTime, new string[] { tskShowTime.Status.ToString() });

statusStrip1.Invoke(delShowTicks, new string[] { tskTicks.Status.ToString() });

foreach (var ee in ex.InnerExceptions)

{

MessageBox.Show(ee.GetType() + ":" + ee.Message);

}

}

}

private void btnstart2TasksWSepTokenSource\_Click(object sender, EventArgs e)

{

// ShowTime task

if (tskShowTime == null)

{

cancelTokenSourceShowTime = new CancellationTokenSource();

CancellationToken cancelToken1 = cancelTokenSourceShowTime.Token;

tskShowTime = new Task(() => ShowTime(cancelToken1), cancelToken1);

}

if (tskShowTime.IsCompleted)

{

cancelTokenSourceShowTime = new CancellationTokenSource();

CancellationToken cancelToken1 = cancelTokenSourceShowTime.Token;

tskShowTime = new Task(() => ShowTime(cancelToken1), cancelToken1);

}

if (tskShowTime.Status != TaskStatus.Running)

tskShowTime.Start();

// Ticks task

if (tskTicks == null)

{

cancelTokenSourceShowStockPrice = new CancellationTokenSource();

CancellationToken cancelToken2 = cancelTokenSourceShowStockPrice.Token;

//cancelToken2.Register(ShowTicksCancelled);

Action<object> cancelCallback = new Action<object>(ShowTicksCancelled2);

cancelToken2.Register(cancelCallback, DateTime.Now);

tskTicks = new Task(() => ShowTicks(cancelToken2), cancelToken2);

}

if (tskTicks.IsCompleted)

{

cancelTokenSourceShowStockPrice = new CancellationTokenSource();

CancellationToken cancelToken2 = cancelTokenSourceShowStockPrice.Token;

tskTicks = new Task(() => ShowTicks(cancelToken2), cancelToken2);

}

if (tskTicks.Status != TaskStatus.Running)

tskTicks.Start();

}

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

{

Task tcancel = new Task(() =>

{

try

{

cancelTokenSourceShowStockPrice.Cancel();

tskTicks.Wait();

}

catch (AggregateException ae)

{

statusStrip1.Invoke(delShowTicks, new string[] { tskTicks.Status.ToString() });

}

});

tcancel.Start();

}

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

{

Task tcancel = new Task(() =>

{

try

{

cancelTokenSourceShowTime.Cancel();

tskShowTime.Wait();

}

catch (AggregateException ae)

{

statusStrip1.Invoke(delShowTime, new string[] { tskShowTime.Status.ToString() });

}

});

tcancel.Start();

}

void ShowTicksCancelled()

{

MessageBox.Show("Show Price Task has been cancelled..");

}

void ShowTicksCancelled2(object state)

{

MessageBox.Show("Show Ticks Task has been cancelled..\n" +

"Task was started at: " + state.ToString());

}

}

}

**Conclusion:**

After doing this assignment I got to understand better Actions and Functions as long as lambda expressions by redoing the assignments.