**CS 590 – PARALLEL AND DISTRIBUTED COMPUTING**

**HOMEWORK 2**

**Name:** Ivan Sangines Escrig

**ID#:** 968606

**Instructor:** Dr. Mahmood

**Date:** September 17, 2018

**TABLE OF CONTENT**

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

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

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

[**CONCLUSION:** 1](#_Toc446970375)6

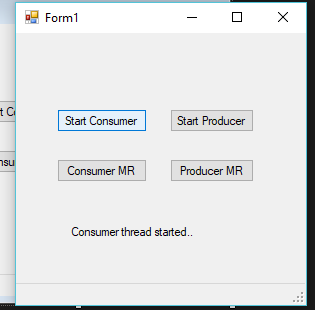
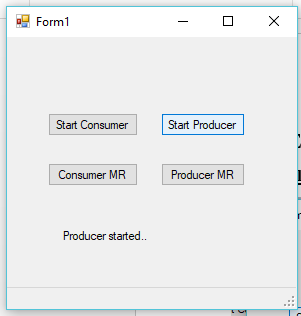
**INTRODUCTION**

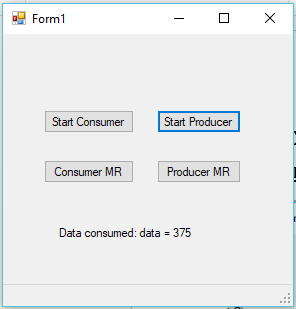
The porpuse of this assignment is to get Multithreading in C#. This will be achieved by doing the second half of the handout. After doing this examples I will have more knowledge about ManualResetEvent, ThreadPools, Semaphores and Mutex.

**SCREEN SHOTS:**

**Example 1:**

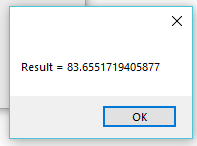
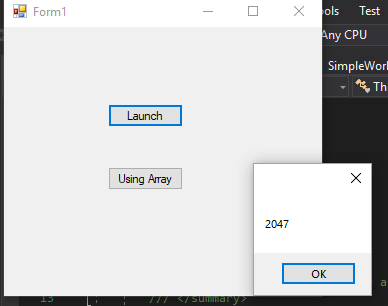
Here we can see all the stages of the application. Whenever a consumer starts, it gets on the state of waiting for a signal from the producer “saying that the data to consume is ready”. The consumers will not keep runing until producer, produces data and sends a pulse to the consumer to start runin and consuming the data.

** **

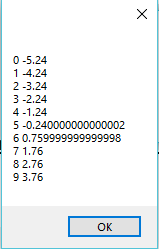
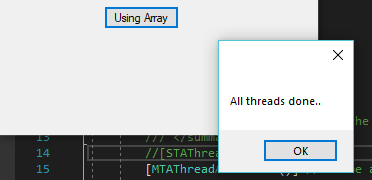
****

**Example 2:**

PART A:

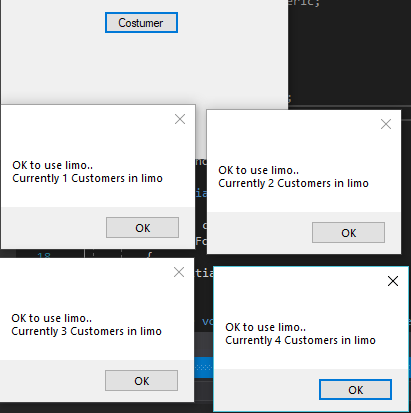
****

PART B:

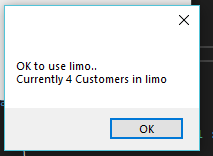


**Example 3:**

These are the outputs whenever 4 customer want to enter the limo. After, the forth one, there will be a delay before the 5th customer enters.

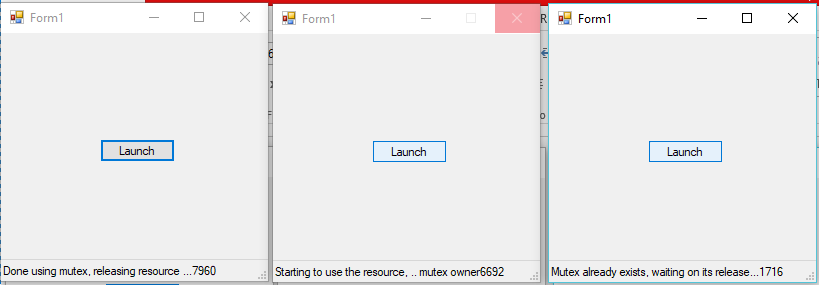


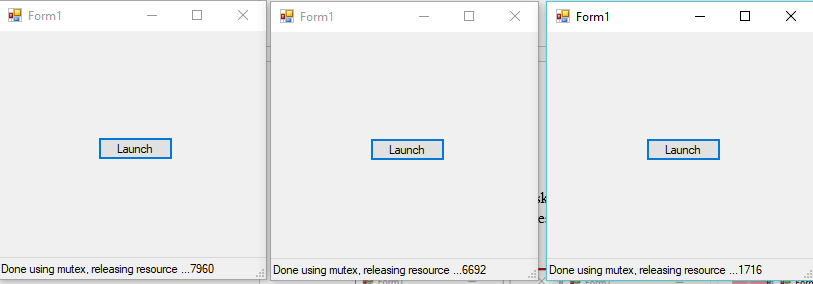
Once the “limo” is full with 4 Customers, it will have a delay (need to wait for one custumer to leave) in order to add a new Custumer.



**Example 4:**

Here we can see the three tasks and how the status bar is changing over the time; realizing, starting and waiting for the resources.





**SOURCE CODE:**

**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.Threading;

using System.Windows.Forms;

namespace ProducerConsumer

{

delegate void DisplayDel(string msg);

public partial class Form1 : Form

{

Thread thConsumer;

Thread thProducer;

Thread thConsumerMR;

Thread thProducerMR;

object olock = new object();

bool dataReady = false;

int data = 0;

DisplayDel ddel = null;

ManualResetEvent mr = new ManualResetEvent(false);

public Form1()

{

InitializeComponent();

ddel = new DisplayDel(DisplayStatus);

}

void DisplayStatus(string msg)

{

lblStatus.Text = msg;

}

void Consumer()

{

for (int i = 0; i < 2; i++)

{

statusStrip1.Invoke(ddel, new string[] { "Consumer thread started.." });

if (dataReady == false)

{

lock (olock)

{

Monitor.Wait(olock);

}

// Monitor.Wait and Monitor.Pulse

// are required to be inside the same

// lock they are testing

}

Thread.Sleep(1000);

statusStrip1.Invoke(ddel, new string[] { "starting to consume data.." });

lock (olock)

{

data = data - 2000;

dataReady = false;

}

Thread.Sleep(4000); // pretend it takes 4 secs to

// consume data

statusStrip1.Invoke(ddel, new string[]{"Data consumed: data = " + data.ToString()});

}

}

void Producer()

{

statusStrip1.Invoke(ddel, new string[]{"Producer started.."});

Thread.Sleep(9000); // 9 secs to produce data

lock (olock)

{

data = 2375;

dataReady = true;

}

lock (olock)

{

Monitor.Pulse(olock); // signal the waiting thread

}

}

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

{

thConsumer = new Thread(new ThreadStart(Consumer));

thConsumer.Start();

}

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

{

thProducer = new Thread(new ThreadStart(Producer));

thProducer.Start();

}

void ConsumerMR() // uses ManualResetEvent for

{ // synchronization

for (int i = 0; i < 2; i++)

{

statusStrip1.Invoke(ddel, new string[]{"Consumer thread started.."});

if (dataReady == false)

{

mr.WaitOne(); // wait for producer

mr.Reset(); // close the gate

}

Thread.Sleep(1000);

statusStrip1.Invoke(ddel, new string[]{"starting to consume data.."});

lock (olock)

{

data = data - 2000;

dataReady = false;

}

Thread.Sleep(4000); // pretend it takes 4 secs to

// consume data

statusStrip1.Invoke(ddel, new string[]{ "Data consumed: data = " + data.ToString() });

}

}

void ProducerMR() // based on ManualResetEvent

{

statusStrip1.Invoke(ddel, new string[]{"Producer started.."});

Thread.Sleep(9000); // 9 secs to produce data

lock (olock)

{

data = 2375;

dataReady = true;

}

mr.Set(); // send signal to waiting thread

}

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

{

thConsumerMR = new Thread(new ThreadStart(ConsumerMR));

thConsumerMR.Start();

}

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

{

thProducerMR = new Thread(new ThreadStart(ProducerMR));

thProducerMR.Start();

}

}

}

**Example 2:**

Class Worker:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Threading;

namespace ThreadPooling

{

class MyWorker

{

ManualResetEvent mrev;

object olock = new object();

int workerNum = -1;

public int WorkerNum

{

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

}

int inputData = -1;

double result = 0;

public double Result

{

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

}

public MyWorker(ManualResetEvent ev, int inpData)

{

mrev = ev;

inputData = inpData;

}

// main thread function

public void ThreadPoolCallback(object context)

{

lock (olock) { workerNum = (int)context; }

double res = DoWork(inputData);

lock (olock) { result = res; }

mrev.Set();

}

public double DoWork(int data)

{

Thread.Sleep(2000);

return 5.2 \* 3.8 - 25 + data;

}

}

}

Class SimpleWorker:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Threading;

namespace ThreadPooling

{

class SimpleWorker

{

public SimpleWorker(int data, ManualResetEvent ev)

{

inputData = data;

mRE = ev;

}

object olock = new object();

double result;

public double Result

{

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

}

ManualResetEvent mRE = null;

int inputData = 0;

public void MainCompute(object context)

{

// Writing to a Console does not

// cause a cross threading error.

// However, you can only see the Console

// output if you choose Debug->Start debugging

// Use Console.WriteLine for debugging

// purposes

if (context is DateTime)

{

Console.WriteLine(((DateTime)context).ToString());

}

double res = InternalCompute();

Thread.Sleep(5000); // pretend it takes 5 secs to compute

lock (olock) { result = res; }

mRE.Set(); // signal the caller that thread is done

}

double InternalCompute()

{

return 45.7575 - inputData / 3.097 + 45.97;

}

}

}

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.Threading;

using System.Windows.Forms;

namespace ThreadPooling

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

ManualResetEvent mr = new ManualResetEvent(false);

// gate is closed

SimpleWorker sw = new SimpleWorker(25, mr);

WaitCallback wcb = new WaitCallback(sw.MainCompute);

// wcb is pointing to MainCompute

// create a launch ThreadPool thread

int maxWorkerThreads = 0;

int maxCompletionPorts = 0;

ThreadPool.GetMaxThreads(out maxWorkerThreads, out maxCompletionPorts);

MessageBox.Show(maxWorkerThreads.ToString());

ThreadPool.QueueUserWorkItem(wcb, DateTime.Now);

// no start method in threadpool, above line

// starts the thread

mr.WaitOne(); // wait for thread to finish

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

}

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

{

MyWorker[] workers = new MyWorker[10];

ManualResetEvent[] ev = new ManualResetEvent[10];

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

{

ev[i] = new ManualResetEvent(false);

workers[i] = new MyWorker(ev[i], i);

ThreadPool.QueueUserWorkItem(workers[i].ThreadPoolCallback, i);

}

WaitHandle.WaitAll(ev); //WaitHandle.WaitAll(ev);

MessageBox.Show("All threads done..");

string out1 = "";

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

{

out1 += workers[i].WorkerNum.ToString() + " "

+ workers[i].Result.ToString() + "\n";

}

MessageBox.Show(out1);

}

}

}

**Example 3:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Threading;

using System.Windows.Forms;

namespace SemaphoreTest

{

class LimoCar

{

public LimoCar(int num, int time)

{

limoNumber = num;

travelTime = time; // specified in seconds

semLim = new Semaphore(4, 4);

// second parameter indicates max users

// that can access the resource

// first parameter, if it is zero, denies

// access to the resource

// each access reduces the value in the

// first parameter

}

object olock = new object();

Semaphore semLim = null;

int limoNumber;

public int LimoNumber

{

get { return limoNumber; }

set { limoNumber = value; }

}

int travelTime;

public int TravelTime

{

get { return travelTime; }

set { travelTime = value; }

}

int customersInLimo = 0;

public int CustomersInLimo

{

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

}

public void UseLimo()

{

semLim.WaitOne(); // check if gate is open

lock (olock) { customersInLimo++; }

MessageBox.Show("OK to use limo..\n" + "Currently " + customersInLimo.ToString() + " Customers in limo");

Thread.Sleep(1000 \* travelTime);

semLim.Release(); // increment gate count

lock (olock) { customersInLimo--; }

}

}

}

**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;

using System.Threading;

namespace MutexTest

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

bool reqCreation = true; // to request creation of a mutex

bool alreadyExists; // if a mutex already exists

int thId = System.AppDomain.GetCurrentThreadId();

Mutex m1 = new Mutex(reqCreation, "MyMutex", out alreadyExists);

// check to see if the mutex is owned by this thread

if (!(reqCreation && alreadyExists))

{

sb1.Text = "Mutex already exists, waiting on its release..." +

thId.ToString();

m1.WaitOne(); // this thread wil block here

}

sb1.Text = "Starting to use the resource, .. mutex owner" +

thId.ToString();

// if this is the owner thread, you can use the shared resource

Thread.Sleep(10000);

// done using the resource

sb1.Text = "Done using mutex, releasing resource ..." + thId.ToString();

m1.ReleaseMutex();

}

}

}

**Conclusion:**

Since I was not very familiar with MultiThreading, this examples helped me to understand it much better. Also, I am now able to use and apply ManualResetEvent, ThreadPools, Semaphores and Mutex.