|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего профессионального образования  **«Московский технологический университет»**  **МИРЭА** | | |

Институт Информационных Технологий

Кафедра Корпоративных Информационных Систем

**ОТЧЕТ**

по Лабораторной Работе №5

на тему

«Параллельная обработка данных»

по дисциплине

«ООП»

Выполнил студент группы ИСБО-11-16 Шайхуллин С.В.

Принял Cтарший преподаватель Мирзоян Д.И.

Выполнено «\_\_\_» \_\_\_\_\_\_\_\_\_2017 г.

Зачтено «\_\_\_» \_\_\_\_\_\_\_\_\_ 2017г.

Москва, 2017

**Теоретическая часть**

В эпоху многоядерных машин, которые позволяют параллельно выполнять сразу несколько процессов, стандартных средств работы с потоками в .NET уже оказалось недостаточно. Поэтому одним из новшеств платформы .NET 4.0 стала библиотека параллельных задач TPL (Task Parallel Library), основной функционал которой располагается в пространстве имен System.Threading.Tasks. Данная библиотека позволяет распараллелить задачи и выполнять их сразу на нескольких процессорах, если на целевом компьютере имеется несколько ядер. Кроме того, упрощается сама работа по созданию новых потоков. Поэтому начиная с .NET 4.0. рекомендуется использовать именно TPL и ее классы для создания многопоточных приложений, хотя стандартные средства и класс Thread по-прежнему находят широкое применение.

В основе библиотеки TPL лежит концепция задач, каждая из которых описывает отдельную продолжительную операцию. В библиотеке классов .NET задача представлена специальным классом - классом Task, который находится в пространстве имен System.Threading.Tasks. Данный класс описывает отдельную задачу, которая запускается асинхронно в одном из потоков из пула потоков. Хотя ее также можно запускать синхронно в текущем потоке.

Второй способ заключается в использовании статического метода Task.Factory.StartNew(). Этот метод также в качестве параметра принимает делегат Action, который указывает, какое действие будет выполняться. Свойства класса Task

Класс Task имеет ряд свойств, с помощью которых мы можем получить информацию об объекте. Некоторые из них:

AsyncState: возвращает объект состояния задачи

CurrentId: возвращает идентификатор текущей задачи

Exception: возвращает объект исключения, возникшего при выполнении задачи

Status: возвращает статус задачи

**Задание**

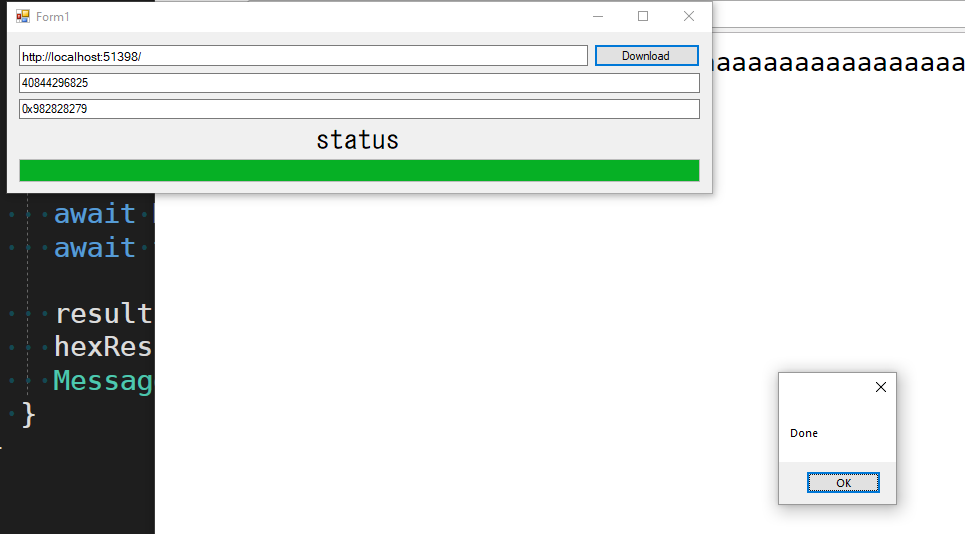
Создать приложение, реализующее обработку данных в параллельном и асинхронном режимах.

Приложению необходимо скачать указанный пользователем файл с сервера, подсчитать сумму, считая файл как массив int (дополнить нулями при необходимости). В процессе работы выводить текущий прогресс выполнения.

**Алгоритм решения задачи**

1. Скачать асинхронно файл
2. Параллельно посчитать сумму по четыре

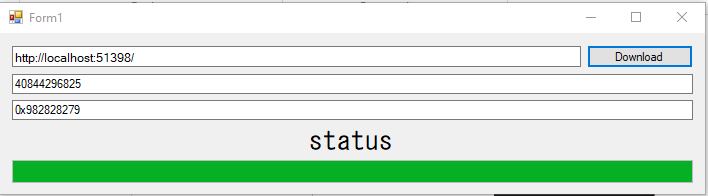
**Тестирование**



**Заключение**

В данной лабораторной работе мы познакомились с реализацией асинхронной и параллельной обработки данных

**Пример работы**



**Исходный код**

using System;

using System.Net;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Shaykhullin.Lab5.Client

{

public partial class MainView : Form

{

private object lockObject = new object();

private byte[] data;

private int position;

private long sumResult;

private int count;

private Task task;

public MainView()

{

InitializeComponent();

}

bool ProgressFinished

{

get

{

lock (lockObject)

{

return position >= data.Length;

}

}

}

bool HasMoreData(out int dataSize)

{

lock (lockObject)

{

if (count < data.Length)

dataSize = (count - position) & -4;

else

dataSize = count - position;

return dataSize > 0;

}

}

private void ProcessData()

{

while (!ProgressFinished)

{

if (HasMoreData(out int dataToProgress))

{

for (int i = position; i < position + (dataToProgress & -4); i += 4)

{

sumResult += data[i] + (data[i + 1] << 8) + (data[i + 2] << 16) + (data[i + 3] << 24);

}

switch (dataToProgress & 3)

{

case 1:

sumResult += data[position + (dataToProgress & -4)];

break;

case 2:

sumResult += data[position + (dataToProgress & -4)] + (data[position + (dataToProgress & -4) + 1] << 8);

break;

case 3:

sumResult += data[position + (dataToProgress & -4)] + (data[position + (dataToProgress & -4) + 1] << 8) + (data[position + (dataToProgress & -4) + 2] << 16);

break;

}

Thread.MemoryBarrier();

position += dataToProgress;

progressBar.BeginInvoke(new Action(() => progressBar.Value = (int)Math.Round(100.0 \* position / data.Length)));

}

else

{

Thread.Yield();

}

}

}

private async Task MakeAsyncRequest(string url)

{

var responce = await WebRequest.CreateHttp(url)

.GetResponseAsync();

using (var netStream = responce.GetResponseStream())

{

data = new byte[responce.ContentLength];

task = Task.Run(new Action(ProcessData));

while (count < responce.ContentLength)

{

int read = await netStream.ReadAsync(data, count, (int)responce.ContentLength - count);

Thread.MemoryBarrier();

lock (lockObject)

{

count += read;

}

}

}

}

private async void OnDownloadClick(object sender, EventArgs e)

{

data = null;

sumResult = position = count = 0;

progressBar.Value = 0;

task?.Dispose();

await MakeAsyncRequest(url.Text);

await task;

result.Text = sumResult.ToString();

hexResult.Text = $"0x{sumResult:X}";

MessageBox.Show("Done");

}

}

}