**Занятие № 13**

**Дата выполнения работы:** 04.05.2023

# **Тема работы:** «Многопоточные приложения»

**Ход работы**

**Задание 1**

1. Определите и выведите на консоль/в файл все запущенные процессы: id,

имя, приоритет, время запуска, текущее состояние, сколько всего времени

использовал процессор и т.д.

2. Исследуйте текущий домен вашего приложения: имя, детали

конфигурации, все сборки, загруженные в домен. Создайте новый домен.

Загрузите туда сборку. Выгрузите домен.

3. Создайте в отдельном потоке следующую задачу расчета (можно сделать

sleep для задержки) и записи в файл и на консоль простых чисел от 1 до n

(задает пользователь). Вызовите методы управления потоком (запуск,

приостановка, возобновление и тд.)

Вовремя выполнения выведите информацию о статусе потока, имени,

приоритете, числовой идентификатор и т.д.

4. Создайте два потока. Первый выводит четные числа, второй нечетные до n

и записывают их в общий файл и на консоль. Скорость расчета чисел у

потоков – разная.

a. Поменяйте приоритет одного из потоков.

b. Используя средства синхронизации организуйте работу потоков, таким

образом, чтобы

выводились сначала четные, потом нечетные числа

последовательно выводились одно четное, другое нечетное.

5. Придумайте и реализуйте повторяющуюся задачу на основе класса Timer

**Листинг программы:**

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 pr13

{

public partial class Form1 : Form

{

List<Book> books = new List<Book>();

List<School> schools = new List<School>();

Random random = new Random();

public Form1()

{

InitializeComponent();

}

void RefreshList<T>(ListBox listBox, List<T> ts)

{

listBox.Items.Clear();

foreach (T t in ts)

{

string text = "";

if (t is School school)

{

text = $"{school.name} ({school.index})";

}

else if (t is Book book)

{

text = $"{book.name} ({book.index})";

}

listBox.Items.Add(text);

}

}

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

{

if (textBox1.Text == "")

{

MessageBox.Show("Вы не ввели текст");

return;

}

books.Add(new Book(textBox1.Text, random.Next(100000)));

RefreshList(listBox2, books);

}

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

{

if (textBox2.Text == "")

{

MessageBox.Show("Вы не ввели текст");

return;

}

schools.Add(new School(textBox2.Text, random.Next(100000)));

RefreshList(listBox1, schools);

}

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

{

var vibor = from school in schools from book in books select new { namesc = school.name, nameb = book.name };

listBox3.Items.Clear();

foreach (var v in vibor)

{

listBox3.Items.Add($"school {v.namesc,-30} book {v.nameb}");

}

}

private void button4\_Click(object sender, EventArgs e)

{

if (listBox1.SelectedIndex == -1)

{

MessageBox.Show("Вы не выбрали элемент");

return;

}

listBox1.Items.RemoveAt(listBox1.SelectedIndex);

schools.RemoveAt(listBox1.SelectedIndex);

}

private void button5\_Click(object sender, EventArgs e)

{

if (listBox2.SelectedIndex == -1)

{

MessageBox.Show("Вы не выбрали элемент");

return;

}

listBox1.Items.RemoveAt(listBox1.SelectedIndex);

books.RemoveAt(listBox1.SelectedIndex);

}

}

class School

{

public string name;

public int index;

public School(string name,int ind)

{

this.name = name;

this.index = ind;

}

}

class Book

{

public string name;

public int index;

public Book(string name, int ind)

{

this.name = name;

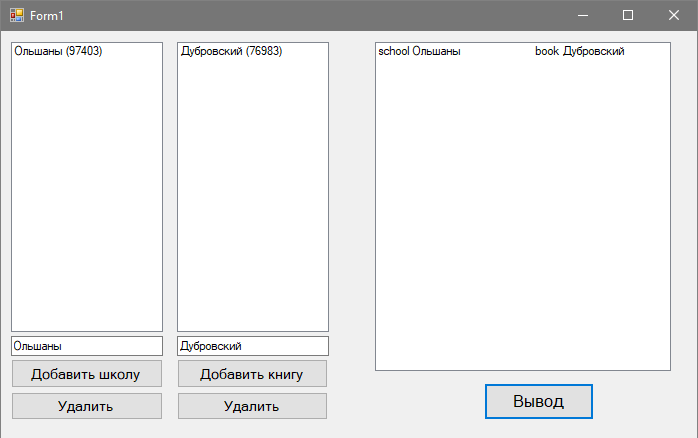
this.index = ind;

}

}

}

**Результат:**

****

**Вывод:** Многопоточные приложения в C# позволяют разделять выполнение программы на несколько параллельных потоков, что позволяет увеличить производительность и снизить время отклика приложения. Однако, многопоточное программирование требует более тщательного проектирования и обработки ошибок, поскольку возможны проблемы, такие как состояние гонки и взаимная блокировка. В C# существует несколько способов создания многопоточных приложений, таких как использование класса Thread, класса Task, класса ThreadPool, а также асинхронного программирования с использованием ключевых слов async/await. Каждый из этих подходов имеет свои преимущества и недостатки, и их выбор зависит от требований к приложению и особенностей задачи, которую нужно решить. Кроме того, важно учитывать возможности и ограничения железа, на котором будет запускаться приложение, так как многопоточность может потребовать больших ресурсов и не всегда эффективна на слабых машинах.