ФЕДЕРАЛЬНОЕ АГЕНТСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ИРКУТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПУТЕЙ СООБЩЕНИЯ» (ФГБОУ ВО ИрГУПС)

Факультет: «Управления на транспорте и информационные технологии»

Кафедра: «Информационные системы и защита информации»

Лабораторная работа № 1

по дисциплине «Программирование параллельных процессов»

«Организация параллельно выполняемых потоков и межпроцессное взаимодействие»

ЛР.430500.09.04.04.186.ПЗ

Выполнил: Проверил:

студент гр. ПИм 1-22-1 преподаватель

Емельянова К.А Черкашин Е.А.

« » 2022 г. « » 2022г.

Иркутск 2022

**Задание**

**Группа А:** Выполнить вычисления в однопоточном и многопоточном режиме и сравнить времена выполнения. Количество потоков задаётся пользователем. Количество исходных данных не кратно в общем случае количеству потоков. Исходные данные для задания генерируются с помощью генератора псевдослучайных чисел, где 100000<n<1000000, 100<An<10000000. Результаты сравниваются по времени выполнения при разном числе процессов и объёме данных и оформляются в виде таблицы. В отчёте приводятся снимки экрана, программный код, таблицы тестов и замеров времени выполнения, формулируется вывод. Количество потоков по-умолчанию при необходимости получать из **Environment.ProcessorCount.**

1. Даны последовательности чисел А = {*а0…аn–1*} и С = {*с0…сn–1*}. Создать многопоточное приложение, определяющее, совпадают ли поэлементно строки А и С.

2. Дана последовательность чисел С = {*с0…сn–1*}. Дан набор из *N* пар кодирующих чисел (*ai,bi*), т.е. все *ai* заменяются на  *bi*. Создать многопоточное приложение, кодирующее последовательность Сследующим образом: массив разделяется на подмассивы и каждый поток осуществляет кодирование своего подмассива.

3. Дана последовательность чисел С = {*с0…сn–1*} и число *b*. Создать многопоточное приложение для определения количество вхождений числа *b* в массив C.

4. Дана последовательность натуральных чисел {*a0…an–1*}. Создать многопоточное приложение для поиска произведения чисел *a0\*а1\*…\*an–1*.

5. Дана последовательность натуральных чисел {a0…an–1}. Создать

многопоточное приложение для поиска максимального ai.

6. Дана последовательность натуральных чисел {a0…an–1}. Создать многопоточное приложение для поиска минимального ai.

7. Дана последовательность натуральных чисел {a0…an–1}. Создать многопоточное приложение для поиска всех ai, являющихся простыми числами.

8. Дана последовательность натуральных чисел {a0…an–1}. Создать многопоточное приложение для поиска всех ai, являющихся квадратами, любого натурального числа.

9. Дана последовательность натуральных чисел {a0…an–1}. Создать многопоточное приложение для вычисления выражения a0-а1+a2-а3+a4-а5+...

10. Дана последовательность натуральных чисел {a0…an–1}. Создать многопоточное приложение для поиска суммы ∑ai, где ai – четные числа.

**Содержание работы**

В ходе выполнения лабораторной работы на языке C# в среде Visual Studio по описанному заданию была написана программа. Во всех заданиях обеспечен контроль вводимой информации (включая отсутствие специальных символов-разделителей). При некорректном вводе – информация повторно запрашивается у пользователя. Код программы представлен в листингах в ПРИЛОЖЕНИИ А.

В результате выполнения работы были получены результаты, изображенные на рисунках 1-10.

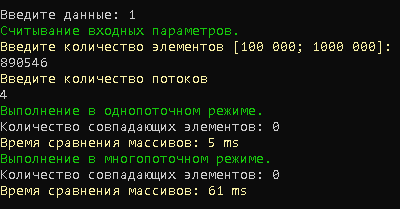


Рисунок 1 – Пример результата выполнения задачи 1

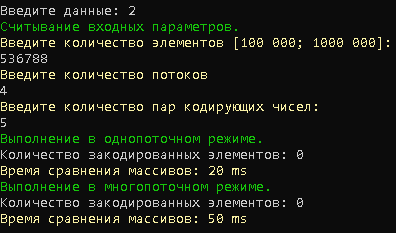


Рисунок 2 – Пример результата выполнения задачи 2

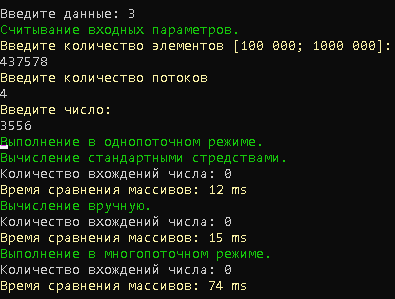


Рисунок3 – Пример результата выполнения задачи 3

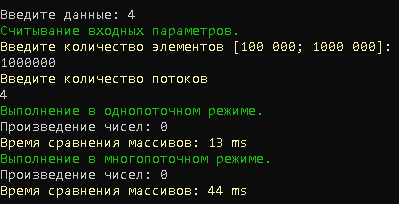


Рисунок 4 – Пример результата выполнения задачи 4

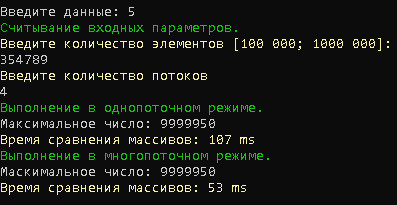


Рисунок 5 – Пример результата выполнения задачи 5

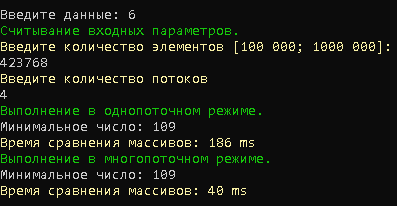


Рисунок 6 – Пример результата выполнения задачи 6

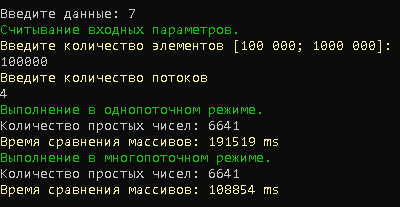


Рисунок 7 – Пример результата выполнения задачи 7

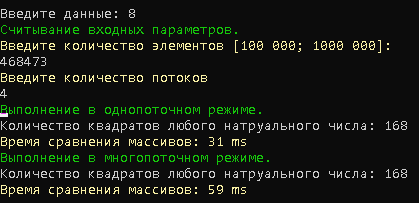


Рисунок 8 – Пример результата выполнения задачи 8

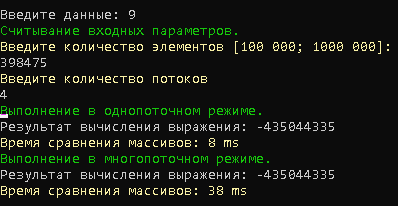


Рисунок 9 – Пример результата выполнения задачи 9

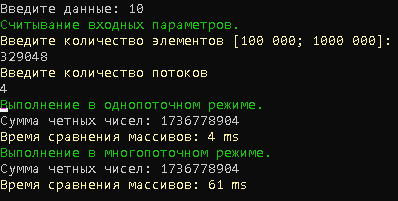


Рисунок 10 – Пример результата выполнения задачи 10

Далее выполним для всех задач вычисления в однопоточном и многопоточном режиме и сравним времена выполнения при разном числе процессов и объёме данных. Результаты оформим в виде таблицы 1.

Таблица 1 – Результаты времени выполнения

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Режим** | **Многопоточный (мс)** | | | | **Однопоточный (мс)** | |
| **Размер последовательности** | **100 000**  **элементов** | **100 000 элементов** | **1 000 000**  **элементов** | **1 000 000 элементов** | **100 000**  **элементов** | **1 000 000 элементов** |
| **Кол-во потоков** | **2** | **10** | **2** | **10** |  | |
| **Задача 1** | **43** | **127** | **30** | **143** | **1** | **13** |
| **Задача 2** | **135** | **166** | **88** | **163** | **3** | **37** |
| **Задача 3** | **27** | **144** | **23** | **146** | **1** | **25** |
| **Задача 4** | **66** | **119** | **29** | **128** | **13** | **22** |
| **Задача 5** | **47** | **393** | **155** | **162** | **1** | **39** |
| **Задача 6** | **23** | **163** | **25** | **215** | **1** | **13** |
| **Задача 7** | **228 908** | **125 433** | **1 118 933** | **1 102 713** | **228 932** | **1 953 841** |
| **Задача 8** | **24** | **147** | **64** | **227** | **7** | **66** |
| **Задача 9** | **19** | **91** | **22** | **150** | **4** | **18** |
| **Задача 10** | **22** | **95** | **30** | **124** | **1** | **15** |

**Результаты**

В результате выполнения лабораторной работы сделаем выводы, что в данной технологии в многопоточном режиме слишком большое количество потоков понижают производительность, кроме тех случаев, когда выполняется работа с громоздкими вычислениями больших чисел с множеством итераций и переборов, например, в задаче 7. Это доказывается тем фактом, что во всех задачах в многопоточном режиме с любым объемом данных вычисления с 2 потоками было производительнее, чем с 10, кроме задачи 7. На ее же примере можно сказать, что потенциал многопоточности в сравнении с однопоточным режимом, также раскрывается при большом количестве вычислений и итераций, так как это единственные разы, когда многопоточный режим превзошел в данных задачах однопоточный.

Также, если сравнить производительность многопоточного режима при идентичном количестве потоков, но разном объеме данных, видно, что потенциал многопоточности раскрывается при росте объёма данных. Чем больше элементов в последовательности, тем выше количество элементов, обрабатываемых за 1 м/с.

То есть многопоточный режим оптимально использовать при большом объеме данных и сложных вычислениях, тогда он раскрывает свой потенциал. Вероятно, это связано с тем, что в многопоточном режиме программа работает медленнее в ожидании общих ресурсов, а также у нее появляется необходимость тратить дополнительные ресурсы процессора для управления потоками. Таким образом, в простых вычислениях с малым объемом данных в данном случае выигрывает однопоточный режим, так как многопоточный тратит больше времени на реализацию многопоточности, нежели на сами вычисления. Однако он больше оправдывается при больших объемах данных и сложных вычислениях, когда затрата дополнительных ресурсов на реализацию технологии нивелируется более высокой скоростью обработки большого объема данных.

ПРИЛОЖЕНИЕ А

**Листинг 1 – Файл Program.cs**

using Lab1;

Console.ForegroundColor = ConsoleColor.DarkBlue;

Console.WriteLine("Лабораторная работа 1. Организация параллельно выполняемых потоков и межпроцессное взаимодействие.");

Console.WriteLine("Для запуска задачи введите её номер.");

Console.WriteLine("Для выхода нажмите q");

Console.ResetColor();

Console.WriteLine();

bool isExit = true;

do

{

var commands = new Commands();

Console.Write("\nВведите данные: ");

var command = Console.ReadLine();

isExit = commands.ParseCommand(command);

} while (isExit);

Console.WriteLine("Завершение программы.");

**Листинг 2 – Файл Commands.cs**

using Lab1.Task;

namespace Lab1;

public class Commands

{

private readonly Dictionary<string, ITask> \_dictionaryTask;

private readonly Dictionary<string, string> \_dictionary;

public Commands()

{

\_dictionaryTask = new Dictionary<string, ITask>

{

{ "1", new Task1() },

{ "2", new Task2() },

{ "3", new Task3() },

{ "4", new Task4() },

{ "5", new Task5() },

{ "6", new Task6() },

{ "7", new Task7() },

{ "8", new Task8() },

{ "9", new Task9() },

{ "10", new Task10() }

};

\_dictionary = new Dictionary<string, string>

{

{ "help", "Реализация отображение справки в разработке" },

{ "q", "Реализация выхода в разработке" }

};

}

public bool ParseCommand(string command)

{

if (\_dictionaryTask.ContainsKey(command))

{

\_dictionaryTask[command].Run();

}

else if (\_dictionary.ContainsKey(command))

{

Console.WriteLine(\_dictionary[command]);

}

else

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Неизвестная комманда, повторите ввод!");

Console.ResetColor();

}

return true;

}

}

**Листинг 3 – Файл Result.cs**

namespace Lab1;

/// <summary>

/// Результаты выполнения задачи.

/// </summary>

public struct TaskResult

{

/// <summary>

/// Номер задачи.

/// </summary>

public string Title;

/// <summary>

/// Количество элементов в последовательности.

/// </summary>

public int CountElements;

/// <summary>

/// Количество потоков.

/// </summary>

public int CountThreads;

/// <summary>

/// Время выполнения задачи.

/// </summary>

public string Time;

/// <summary>

/// Результат выполнения задачи.

/// </summary>

public string Results;

public override string ToString()

{

return string.Format($"{Title},{CountElements},{CountThreads},{Time},{Results}" +

$"{Environment.OSVersion}," +

$"{Environment.GetEnvironmentVariable("PROCESSOR\_ARCHITECTURE")}," +

$"{Environment.GetEnvironmentVariable("PROCESSOR\_IDENTIFIER")}," +

$"{Environment.ProcessorCount}");

}

}

**Листинг 4 – Файл AbstractTask.cs**

using System.Diagnostics;

using NLog;

namespace Lab1.Task;

public abstract class AbstractTask : ITask

{

/// <summary>

/// Название задачи.

/// </summary>

private readonly string \_title;

/// <summary>

/// Описание задачи.

/// </summary>

private readonly string \_description;

/// <summary>

///Данные о времени запуске программы.

/// </summary>

protected Stopwatch TimeExecution;

/// <summary>

/// Количество элементов массивов.

/// </summary>

protected int CountElements;

/// <summary>

/// Минимальное количество элеметов массива.

/// </summary>

private const int MinCountElements = 100000;

/// <summary>

/// Максимальное количество элеметов массива.

/// </summary>

private const int MaxCountElements = 1000000;

/// <summary>

/// Количество потоков.

/// </summary>

protected int CountThreads = Environment.ProcessorCount;

/// <summary>

/// Последовательность натуральных чисел.

/// </summary>

protected int[] Array;

/// <summary>

/// Потоки.

/// </summary>

protected Thread[] Threads;

/// <summary>

/// Логгер приложения.

/// <include file="nlog.config" path="nlog/[@name="nlog"]/\*"/>

/// </summary>

protected Logger Logger { get; }

/// <summary>

/// Формат вывода целого числа.

/// </summary>

private const string IntFormat = "{0:## ##0}";

/// <summary>

/// Результат задачи.

/// </summary>

protected TaskResult TaskResult;

/// <summary>

/// Возвращаемые значения потоков.

/// </summary>

protected int[] ThreadReturns;

protected AbstractTask(string title, string description)

{

\_title = title;

\_description = description;

Logger = LogManager.GetCurrentClassLogger();

TimeExecution = new Stopwatch();

TaskResult = new TaskResult();

}

/// <summary>

/// Считывание входных данных.

/// </summary>

protected virtual void ReadInputData()

{

Logger.Debug("Считывание входных параметров.");

CountElements = ReadElementsFromConsole(

$"Введите количество элементов [{FormatInt(MinCountElements)}; {FormatInt(MaxCountElements)}]: ",

MinCountElements, MaxCountElements);

CountThreads = ReadThreadsFromConsole($"Введите количество потоков");

}

/// <summary>

/// Инициализация массива случайными числами.

/// </summary>

/// <returns> массива заполненный случайными числами.</returns>

protected int[] InitialArrayRandomData()

{

var array = new int[CountElements];

Random random;

for (var i = 0; i < CountElements; i++)

{

random = new Random();

array[i] = random.Next(100,10000000);

}

return array;

}

/// <summary>

/// Выполнение действий в многопоточном режиме.

/// </summary>

protected virtual void ExecutionWithThread()

{

Logger.Debug("Выполнение в многопоточном режиме.");

TaskResult.CountThreads = CountThreads;

}

/// <summary>

/// Выполнение действий в однопоточном потоке.

/// </summary>

protected virtual void ExecutionWithoutThread()

{

Logger.Debug("Выполнение в однопоточном режиме.");

TaskResult.CountThreads = 1;

}

/// <summary>

/// Считывание количества элементов с консоли.

/// </summary>

/// <param name="message">Сообщение перед вводом данных</param>

/// <param name="minValue">Минимальное значение (включительно).</param>

/// <param name="maxValue">Максимальное значение (включительно).</param>

/// <returns>Считанное целое число с консоли.</returns>

private int ReadElementsFromConsole(string message, int minValue, int maxValue)

{

bool error = true;

int resultRead = 0;

do

{

try

{

Logger.Info(message);

resultRead = Convert.ToInt32(Console.ReadLine());

error = (resultRead < minValue || resultRead > maxValue);

if (error)

{

Logger.Error($"Вводимое значение должно быть в промежутке [{minValue}; {maxValue}]");

}

}

catch (FormatException formatException)

{

WriteError("", formatException);

}

} while (error);

return resultRead;

}

/// <summary>

/// Считывание количества потоков с консоли.

/// </summary>

/// <param name="message">Сообщение перед вводом данных</param>

/// <returns>Считанное целое число с консоли.</returns>

private int ReadThreadsFromConsole(string message)

{

bool error = true;

int resultRead = 0;

do

{

try

{

Logger.Info(message);

resultRead = Convert.ToInt32(Console.ReadLine());

error = false;

}

catch (FormatException formatException)

{

WriteError("Некорректный ввод данных!", formatException);

error = true;

}

} while (error);

return resultRead;

}

/// <summary>

/// Считывание целового числа из консоли.

/// </summary>

/// <param name="message">Сообщение перед вводом данных</param>

/// <returns>Считанное целое число с консоли.</returns>

protected int ReadDigitFromConsole(string message)

{

bool error = true;

int resultRead = 0;

do

{

try

{

Logger.Info(message);

resultRead = Convert.ToInt32(Console.ReadLine());

error = false;

}

catch (FormatException formatException)

{

WriteError("Ошибка считывания числа", formatException);

}

} while (error);

return resultRead;

}

/// <summary>

/// Запись ошибки.

/// </summary>

/// <param name="errorMessage">Ошибка.</param>

/// <param name="exception"></param>

private void WriteError(string errorMessage, Exception exception)

{

Logger.Error(errorMessage);

Logger.Trace(exception);

}

/// <summary>

/// Запуск выполения задачи.

/// </summary>

public void Run()

{

ReadInputData();

TaskResult.Title = \_title;

TaskResult.CountElements = CountElements;

Array = new int[CountElements];

Array = InitialArrayRandomData();

Threads = new Thread[CountThreads];

Threads.Initialize();

ExecutionWithoutThread();

TimeExecution.Reset();

ExecutionWithThread();

}

/// <summary>

/// Вывод результатов работы таймера.

/// </summary>

protected void WriteTimeResult()

{

Logger.Info($"Время сравнения массивов: {TimeExecution.ElapsedMilliseconds} ms");

TaskResult.Time = TimeExecution.ElapsedMilliseconds.ToString();

using var writer = new StreamWriter("result.csv", true);

writer.WriteLine(TaskResult.ToString());

writer.Flush();

}

/// <summary>

/// Форматирование числа.

/// </summary>

/// <param name="number">Число</param>

/// <returns>Отформатированное число.</returns>

private string FormatInt(int number)

{

return string.Format(IntFormat, number);

}

/// <summary>

/// Запуск выполнения потоков.

/// </summary>

protected void StartExecutionThread()

{

ThreadReturns = new int[CountThreads];

for (var i = 0; i < CountThreads; i++)

{

int temp = i;

ThreadReturns[temp] = 0;

Threads[i] = new Thread(() => { ThreadReturns[temp] = ThreadFunction(temp); });

Threads[i].Start();

}

}

/// <summary>

/// Функция разделения последовательности на подпоследовательности для вычисления функции потоком.

/// </summary>

/// <param name="numThread">Номер потока</param>

/// <returns>Результат вычисления функции потоком.</returns>

protected int ThreadFunction(int numThread)

{

int begin, parts, end;

parts = CountElements / CountThreads;

begin = parts \* numThread;

end = begin + parts;

if (numThread == CountThreads - 1)

end = CountElements;

return CalculateThreadFunction(begin, end);

}

/// <summary>

/// Вычисление функции потоком.

/// </summary>

/// <param name="begin">Начало подпоследовательности</param>

/// <param name="end">Конец подпоследовательности</param>

/// <returns>Результат вычисления функции потоком.</returns>

protected abstract int CalculateThreadFunction(int begin, int end);

}

**Листинг 5 – Файл ITask.cs**

namespace Lab1.Task;

public interface ITask

{

/// <summary>

/// Запуск задачи.

/// </summary>

/// <returns></returns>

public void Run();

}

**Листинг 6 – Файл Task1.cs**

using System.Diagnostics;

namespace Lab1.Task;

public class Task1 : AbstractTask

{

/// <summary>

/// Последовательность чисел С.

/// </summary>

private int[] \_arrayC;

public Task1()

: base("Task1",

"Совпадают ли поэлементно массивы А и С")

{

}

protected override void ReadInputData()

{

base.ReadInputData();

\_arrayC = new int[CountElements];

\_arrayC = InitialArrayRandomData();

}

protected override void ExecutionWithThread()

{

base.ExecutionWithThread();

TimeExecution.Start();

StartExecutionThread();

int result = 0;

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

{

Threads[i].Join();

result += ThreadReturns[i];

}

TimeExecution.Stop();

Console.WriteLine("Количество совпадающих элементов: {0}",

TaskResult.Results = result.ToString());

WriteTimeResult();

}

protected override int CalculateThreadFunction(int begin, int end)

{

int countEqualElements = 0;

for (int i = begin; i < end; i++)

{

if (Array[i] == \_arrayC[i])

{

countEqualElements++;

}

}

return countEqualElements;

}

protected override void ExecutionWithoutThread()

{

base.ExecutionWithoutThread();

int countEqualElements = 0;

TimeExecution.Start();

for (var i = 0; i < CountElements; i++)

{

if (Array[i] == \_arrayC[i])

{

countEqualElements++;

}

}

TimeExecution.Stop();

Console.WriteLine("Количество совпадающих элементов: {0}",

TaskResult.Results = countEqualElements.ToString());

WriteTimeResult();

}

}

**Листинг 7 – Файл Task2.cs**

namespace Lab1.Task;

public class Task2 : AbstractTask

{

/// <summary>

/// Количество пар кодирующих чисел.

/// </summary>

private int \_N;

/// <summary>

/// Массив кодирующих чисел a.

/// </summary>

private int[] \_arrayA;

/// <summary>

/// Массив кодирующих чисел b.

/// </summary>

private int[] \_arrayB;

/// <summary>

/// Массив для хранения последовательности чисел С.

/// </summary>

private int[] \_arrayС;

public Task2() : base("Task2","Кодировка последовательности C")

{

}

protected override void ReadInputData()

{

base.ReadInputData();

\_N = ReadDigitFromConsole("Введите количество пар кодирующих чисел: ");

\_arrayA = InitialEncodingNumbersRandomData();

\_arrayB = InitialEncodingNumbersRandomData();

}

/// <summary>

/// Инициализация массива случайными неповторяющимися числами.

/// </summary>

/// <returns> Массив, заполненный случайными числами.</returns>

protected int[] InitialEncodingNumbersRandomData()

{

int[] arrayTemp = new int[\_N];

int temp;

Random random;

for (int i = 0; i < \_N; i++)

{

random = new Random();

temp = random.Next();

while (arrayTemp.Contains(temp)) temp = random.Next();

arrayTemp[i] = temp;

}

return arrayTemp;

}

protected override void ExecutionWithoutThread()

{

base.ExecutionWithoutThread();

\_arrayС = Array.ToArray();

int countEncodedElements = 0;

TimeExecution.Start();

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

{

for (int j = 0; j < \_N; j++)

{

if (Array[i] == \_arrayA[j])

{

Array[i] = \_arrayB[j];

countEncodedElements++;

}

}

}

TimeExecution.Stop();

Console.WriteLine("Количество закодированных элементов: {0}",

TaskResult.Results = countEncodedElements.ToString());

WriteTimeResult();

}

protected override void ExecutionWithThread()

{

base.ExecutionWithThread();

Array = \_arrayС.ToArray();

TimeExecution.Start();

StartExecutionThread();

int result = 0;

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

{

Threads[i].Join();

result += ThreadReturns[i];

}

TimeExecution.Stop();

Console.WriteLine("Количество закодированных элементов: {0}",

TaskResult.Results = result.ToString());

WriteTimeResult();

}

protected override int CalculateThreadFunction(int begin, int end)

{

int countEncodedElements = 0;

for (int i = begin; i < end; i++)

{

for (int j = 0; j < \_N; j++)

{

if (Array[i] == \_arrayA[j])

{

Array[i] = \_arrayB[j];

countEncodedElements++;

}

}

}

return countEncodedElements;

}

}

**Листинг 8 – Файл Task3.cs**

namespace Lab1.Task;

public class Task3 : AbstractTask

{

/// <summary>

/// Число b.

/// </summary>

private int \_number;

public Task3() : base("Task3",

"Определения количества вхождений числа в массив")

{

}

protected override void ReadInputData()

{

base.ReadInputData();

\_number = ReadDigitFromConsole("Введите число: ");

}

protected override void ExecutionWithoutThread()

{

base.ExecutionWithoutThread();

ExecutionStandard();

ExecutionCustom();

}

/// <summary>

/// Вычисление стандартными средствами количества вхождений числа b.

/// </summary>

private void ExecutionStandard()

{

Logger.Debug("Вычисление стандартными стредствами.");

TimeExecution.Start();

Console.WriteLine("Количество вхождений числа: {0}",

TaskResult.Results = Array.Count(element => element == \_number).ToString());

TimeExecution.Stop();

WriteTimeResult();

}

/// <summary>

/// Вычисление вручную количества вхождений числа b.

/// </summary>

private void ExecutionCustom()

{

Logger.Debug("Вычисление вручную.");

int count = 0;

TimeExecution.Start();

for (var i = 0; i < CountElements; i++)

{

if (Array[i] == \_number)

{

count++;

}

}

TimeExecution.Stop();

Console.WriteLine("Количество вхождений числа: {0}",

TaskResult.Results = count.ToString());

WriteTimeResult();

}

protected override void ExecutionWithThread()

{

base.ExecutionWithThread();

TimeExecution.Start();

StartExecutionThread();

int result = 0;

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

{

Threads[i].Join();

result += ThreadReturns[i];

}

TimeExecution.Stop();

Console.WriteLine("Количество вхождений числа: {0}",

TaskResult.Results = result.ToString());

WriteTimeResult();

}

protected override int CalculateThreadFunction(int begin, int end)

{

int count = 0;

for (int i = begin; i < end; i++)

{

if (Array[i] == \_number)

{

count++;

}

}

return count;

}

}

**Листинг 9 – Файл Task4.cs**

namespace Lab1.Task;

public class Task4 : AbstractTask

{

public Task4() : base(

"Task4",

"Вычисление произведения последовательности чисел")

{

}

protected override void ExecutionWithoutThread()

{

base.ExecutionWithoutThread();

TimeExecution.Start();

Console.WriteLine("Произведение чисел: {0}",

TaskResult.Results = Array.Aggregate((x, y) => x \* y).ToString());

TimeExecution.Stop();

WriteTimeResult();

}

protected override void ExecutionWithThread()

{

base.ExecutionWithThread();

TimeExecution.Start();

StartExecutionThread();

int result = 1;

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

{

Threads[i].Join();

result \*= ThreadReturns[i];

}

TimeExecution.Stop();

Console.WriteLine("Произведение чисел: {0}",

TaskResult.Results = result.ToString());

WriteTimeResult();

}

protected override int CalculateThreadFunction(int begin, int end)

{

int multiplication = 1;

for (int i = begin; i < end; i++)

{

multiplication \*= Array[i];

}

return multiplication;

}

}

**Листинг 10 – Файл Task5.cs**

namespace Lab1.Task;

public class Task5 : AbstractTask

{

public Task5() : base("Task5",

"Поиск максимального значения последовательности")

{

}

protected override void ExecutionWithoutThread()

{

base.ExecutionWithoutThread();

TimeExecution.Start();

Console.WriteLine("Максимальное число: {0}",

TaskResult.Results = Array.Max().ToString());

TimeExecution.Stop();

WriteTimeResult();

}

protected override void ExecutionWithThread()

{

base.ExecutionWithThread();

TimeExecution.Start();

StartExecutionThread();

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

{

Threads[i].Join();

}

int result = ThreadReturns.Max();

TimeExecution.Stop();

Console.WriteLine("Маскимальное число: {0}",

TaskResult.Results = result.ToString());

WriteTimeResult();

}

protected override int CalculateThreadFunction(int begin, int end)

{

int max = Array[begin];

for (int i = begin; i < end; i++)

{

if (Array[i] > max)

{

max = Array[i];

}

}

return max;

}

}

**Листинг 11 – Файл Task6.cs**

namespace Lab1.Task;

public class Task6 : AbstractTask

{

public Task6() : base("Task6",

"Поиск минимального значения последовательности")

{

}

protected override void ExecutionWithoutThread()

{

base.ExecutionWithoutThread();

TimeExecution.Start();

Console.WriteLine("Минимальное число: {0}",

TaskResult.Results = Array.Min().ToString());

TimeExecution.Stop();

WriteTimeResult();

}

protected override void ExecutionWithThread()

{

base.ExecutionWithThread();

TimeExecution.Start();

StartExecutionThread();

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

{

Threads[i].Join();

}

int result = ThreadReturns.Min();

TimeExecution.Stop();

Console.WriteLine("Минимальное число: {0}",

TaskResult.Results = result.ToString());

WriteTimeResult();

}

protected override int CalculateThreadFunction(int begin, int end)

{

int min = Array[begin];

for (int i = begin; i < end; i++)

{

if (Array[i] < min)

{

min = Array[i];

}

}

return min;

}

}

**Листинг 12 – Файл Task7.cs**

namespace Lab1.Task;

public class Task7 : AbstractTask

{

/// <summary>

/// Список простых чисел.

/// </summary>

private List<int> \_primeNumbers;

public Task7() : base("Task7",

"Нахождение в последовательности всех простых чисел")

{

\_primeNumbers = new List<int>();

}

protected override void ExecutionWithoutThread()

{

base.ExecutionWithoutThread();

TimeExecution.Start();

foreach (var element in Array)

{

if (IsPrimeNumber(element))

{

\_primeNumbers.Add(element);

}

}

TimeExecution.Stop();

Console.WriteLine("Количество простых чисел: {0}",

TaskResult.Results = \_primeNumbers.Count.ToString());

WriteTimeResult();

}

/// <summary>

/// Выявление, является ли число простым.

/// </summary>

private bool IsPrimeNumber(int number)

{

for (var i = 2; i < number; i++) {

if (number % i == 0)

{

return false;

}

}

return true;

}

protected override void ExecutionWithThread()

{

base.ExecutionWithThread();

TimeExecution.Start();

StartExecutionThread();

\_primeNumbers = new List<int>();

int result = 0;

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

{

Threads[i].Join();

result += ThreadReturns[i];

}

TimeExecution.Stop();

Console.WriteLine("Количество простых чисел: {0}",

TaskResult.Results = result.ToString());

WriteTimeResult();

}

protected override int CalculateThreadFunction(int begin, int end)

{

int countPrimeNumbers = 0;

for (int i = begin; i < end; i++)

{

if (IsPrimeNumber(Array[i]))

{

countPrimeNumbers++;

\_primeNumbers.Add(Array[i]);

}

}

return countPrimeNumbers;

}

}

**Листинг 13 – Файл Task8.cs**

namespace Lab1.Task;

public class Task8 : AbstractTask

{

/// <summary>

/// Список квадратов натуральных чисел.

/// </summary>

private List<int> \_squaresNaturNumbers;

public Task8() : base("Task8" ,

"Нахождение в последовательности всех элементов, являющихся квадратами, любого натурального числа")

{

\_squaresNaturNumbers = new List<int>();

}

protected override void ExecutionWithoutThread()

{

base.ExecutionWithoutThread();

TimeExecution.Start();

foreach (var element in Array)

{

if (IsSquareNumber(element))

{

\_squaresNaturNumbers.Add(element);

}

}

TimeExecution.Stop();

Console.WriteLine("Количество квадратов любого натруального числа: {0}",

TaskResult.Results = \_squaresNaturNumbers.Count.ToString());

WriteTimeResult();

}

/// <summary>

/// Выявление, является ли число квадратом любого натурального числа.

/// </summary>

private bool IsSquareNumber(int number)

{

double sqrtNumber = Math.Sqrt(number);

if (sqrtNumber == Math.Truncate(sqrtNumber))

{

return true;

}

return false;

}

protected override void ExecutionWithThread()

{

base.ExecutionWithThread();

TimeExecution.Start();

StartExecutionThread();

\_squaresNaturNumbers = new List<int>();

int result = 0;

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

{

Threads[i].Join();

result += ThreadReturns[i];

}

TimeExecution.Stop();

Console.WriteLine("Количество квадратов любого натруального числа: {0}",

TaskResult.Results = result.ToString());

WriteTimeResult();

}

protected override int CalculateThreadFunction(int begin, int end)

{

int countSquaresNumbers = 0;

for (int i = begin; i < end; i++)

{

if (IsSquareNumber(Array[i]))

{

countSquaresNumbers++;

\_squaresNaturNumbers.Add(Array[i]);

}

}

return countSquaresNumbers;

}

}

**Листинг 14 – Файл Task9.cs**

namespace Lab1.Task;

public class Task9 : AbstractTask

{

public Task9() : base("Task9",

"Вычисдление суммы последовательности с чередованием знака.")

{

}

protected override void ExecutionWithoutThread()

{

base.ExecutionWithoutThread();

int znak = 1;

TimeExecution.Start();

Console.WriteLine("Результат вычисления выражения: {0}",

TaskResult.Results = Array.Aggregate((x, y) => x + y \* (znak \*= -1)).ToString());

TimeExecution.Stop();

WriteTimeResult();

}

protected override void ExecutionWithThread()

{

base.ExecutionWithThread();

TimeExecution.Start();

StartExecutionThread();

int result = 0;

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

{

Threads[i].Join();

result += ThreadReturns[i];

}

TimeExecution.Stop();

Console.WriteLine("Результат вычисления выражения: {0}",

TaskResult.Results = result.ToString());

WriteTimeResult();

}

protected override int CalculateThreadFunction(int begin, int end)

{

int calculate = 0;

int znak = -1;

if (begin % 2 == 0)

{

znak = 1;

}

for (int i = begin; i < end; i++)

{

calculate += Array[i] \* znak;

znak \*= -1;

}

return calculate;

}

}

**Листинг 15 – Файл Task10.cs**

namespace Lab1.Task;

public class Task10 : AbstractTask

{

public Task10() : base("Task10",

"Вычисление суммы чётных элементов последовательности")

{

}

protected override void ExecutionWithoutThread()

{

base.ExecutionWithoutThread();

int sum = 0;

TimeExecution.Start();

foreach (var element in Array)

{

if (element % 2 == 0)

{

sum += element;

}

}

TimeExecution.Stop();

Console.WriteLine("Сумма четных чисел: {0}",

TaskResult.Results = sum.ToString());

WriteTimeResult();

}

protected override void ExecutionWithThread()

{

base.ExecutionWithThread();

TimeExecution.Start();

StartExecutionThread();

int result = 0;

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

{

Threads[i].Join();

result += ThreadReturns[i];

}

TimeExecution.Stop();

Console.WriteLine("Сумма четных чисел: {0}",

TaskResult.Results = result.ToString());

WriteTimeResult();

}

protected override int CalculateThreadFunction(int begin, int end)

{

int sum = 0;

for (int i = begin; i < end; i++)

{

if (Array[i] % 2 == 0)

{

sum += Array[i];

}

}

return sum;

}

}