Министерство науки и высшего образования Российской Федерации

Новосибирский Государственный технический университет

Кафедра автоматизированных систем управления



**Отчет по лабораторной работе №4**

**по дисциплине «Параллельное программирование»**

**«Модель с равноправными узлами на C#»**

Вариант - рпк2, рм

Выполнили

студенты группы АВТ-812:

Николин Владислав

Павлушин Юрий

Глинин Евгений

Преподаватель:

Ландовский Владимир Владимирович,

к.т.н., доцент кафедры АСУ

г. Новосибирск

2020 г.

Содержание

[1. Постановка задачи. 3](#_Toc54807100)

[2. Описание алгоритмов с учетом взаимодействия потоков и с описанием структур данных, использующихся в ходе взаимодействия. 4](#_Toc54807101)

[3. Примеры работы программы. 6](#_Toc54807102)

[4. Описание процесса работы программы с использованием диаграммы последовательности. 9](#_Toc54807103)

[5. Результаты работы программы. 10](#_Toc54807104)

[6. Выводы. 11](#_Toc54807105)

# **1. Постановка задачи.**

Программа решает множество независимых однотипных задач. В варианте задается модель распределения работы между потоками и тип решаемых задач. В каждом варианте для простоты предполагается, что условия задач (входные данные) расположены в файле, причем сложность задач может сильно отличаться друг от друга и заранее разделить их на равные по объему вычислений части невозможно. Результаты должны записываться в выходные файлы. Использовать POSIX Threads.

рпк2 - модель с равноправными узлами, представляющими собой пары потоков, первый читает входной файл, второй вычисляет результаты и записывает выходной файл. У каждой пары свой отдельный файл результатов.

рм - разложение на простые множители, элемент исходные данные - число, результат - последовательность чисел.

# **2. Описание алгоритмов с учетом взаимодействия потоков и с описанием структур данных, использующихся в ходе взаимодействия.**

Для каждой пары reader-writer существует общий **массив результатов** и соответствующий **мьютекс результатов** для осуществления корректной параллельной работы потоков.



Рисунок 1 – Блок-схема работы потока, который читает входной файл

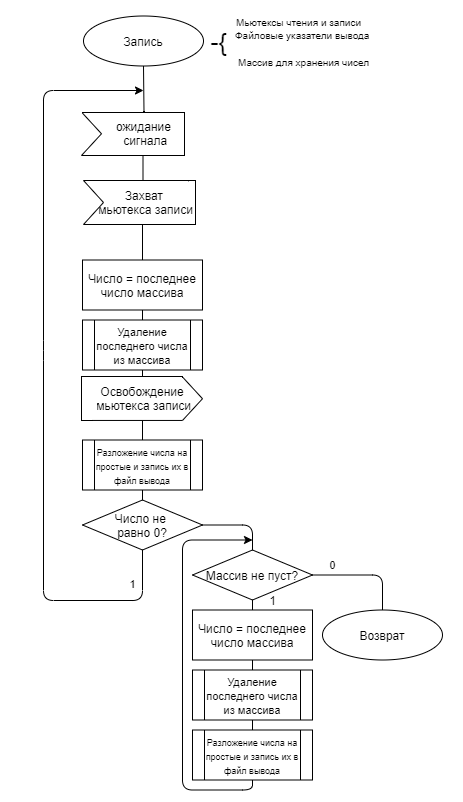


Рисунок 2 – Блок-схема работы потока, который записывает выходной файл и вычисляет результаты

# **3. Листинг программы.**

using System;

using System.Text;

using System.IO;

using System.Threading;

using System.Collections.Generic;

using System.Diagnostics;

namespace PP4

{

struct argument

{

public BinaryReader fileInput;

public StreamWriter fileOutput;

public List<int> arrayCells;

public Mutex readMutex;

public Mutex writeMutex;

public Semaphore waitSem;

/\*mutex\*/

}

class Program

{

static void Main(string[] args)

{

Stopwatch timer = new Stopwatch();

timer.Start();

int numPair = 1;

List<string> output = new List<string>();

List<Thread> threads = new List<Thread>();

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

output.Add("output" + i + ".txt");

BinaryWriter inputFileW = new BinaryWriter(File.Open("input.bin", FileMode.Create, FileAccess.Write));

const int numberCount = 200000;

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

inputFileW.Write(i);

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

inputFileW.Write(0);

inputFileW.Close();

BinaryReader inputFileR = new BinaryReader(File.Open("input.bin", FileMode.Open, FileAccess.Read), Encoding.ASCII);

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

{

threads.Add(new Thread(read));

threads.Add(new Thread(writeOut));

}

List<argument> list = new List<argument>();

argument temp;

temp.fileInput = inputFileR;

temp.readMutex = new Mutex();

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

{

temp.fileOutput = new StreamWriter(File.Open(output[i], FileMode.OpenOrCreate, FileAccess.Write));

temp.arrayCells = new List<int>();

temp.writeMutex = new Mutex();

temp.waitSem = new Semaphore(0, numberCount / numPair);

list.Add(temp);

}

for (int i = 0; i < numPair \* 2; i += 2)

{

threads[i].Start(@list[i / 2]);

threads[i + 1].Start(@list[i / 2]);

}

for (int i = numPair; i < numPair \* 2; i++)

threads[i].Join();

temp.readMutex.Close();

inputFileR.Close();

timer.Stop();

Console.WriteLine(timer.ElapsedMilliseconds / (double)1000);

return;

}

public static void read(Object obj)

{

argument info = (argument)obj;

int number;

while (true)

{

info.readMutex.WaitOne();

number = info.fileInput.ReadInt32();

info.readMutex.ReleaseMutex();

if (number == 0) break;

info.writeMutex.WaitOne();

info.arrayCells.Insert(0, number);

info.writeMutex.ReleaseMutex();

info.waitSem.Release();

}

info.arrayCells.Insert(0, 0);

info.waitSem.Release();

return;

}

public static void writeOut(Object obj)

{

int number;

argument info = (argument)obj;

int division;

do

{

info.waitSem.WaitOne();

info.writeMutex.WaitOne();

number = info.arrayCells[0];

info.arrayCells.RemoveAt(0);

info.writeMutex.ReleaseMutex();

division = 2;

info.fileOutput.Write(number + " = ");

while (number > 1)

{

while (number % division == 0)

{

info.fileOutput.Write(division);

number = number / division;

if (number > 1)

info.fileOutput.Write(" \* ");

}

if (division == 2) division++;

else division += 2;

}

info.fileOutput.Write("\n");

} while (number != 0);

while (info.arrayCells.Count != 0)

{

number = info.arrayCells[0];

info.arrayCells.RemoveAt(0);

division = 2;

info.fileOutput.Write(number + " = ");

while (number > 1)

{

while (number % division == 0)

{

info.fileOutput.Write(division);

number = number / division;

if (number > 1)

info.fileOutput.Write(" \* ");

}

if (division == 2) division++;

else division += 2;

}

info.fileOutput.Write("\n");

}

info.writeMutex.Close();

info.fileOutput.Close();

return;

}

}

}

# **4. Описание процесса работы программы с использованием диаграммы последовательности.**

Рисунок 3 – Диаграмма последовательности типовой ситуации, в которой 2 пары потоков.

# **5. Результаты работы программы.**

199998 элементов, 2 пары потоков:

Файл input имеет расширение bin для увеличения скорости работы программы, в данном файле записаны последовательно числа от 2 до 199998.

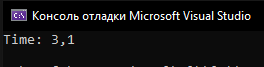


Рисунок 4 – время работы программы с 2 парами потоков на языке C#.



Рисунок 5 – время работы программы с 2 парами потоков на языке C++.

Небольшие фрагменты файлов output.txt:

Output0.txt:

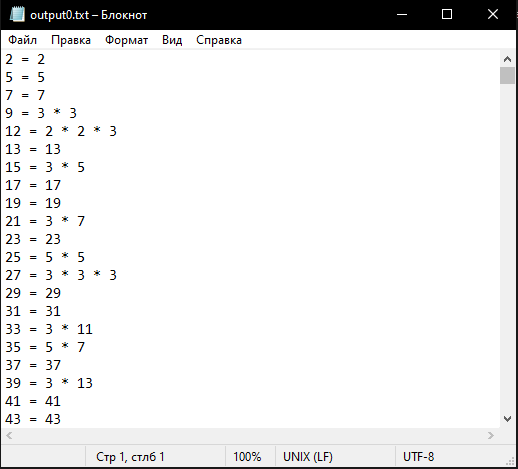


Рисунок 6 – 1 файл с результатами работы программы с 2 парами потоков.

Output1.txt:

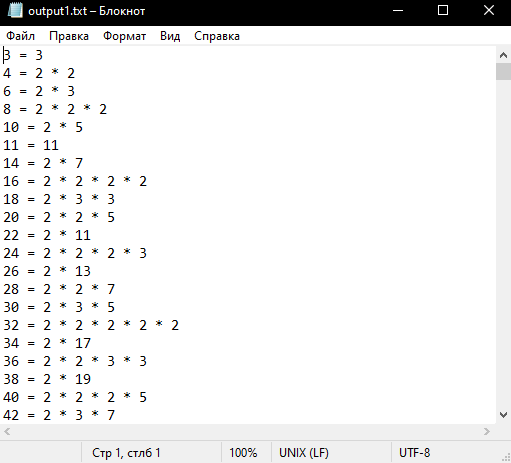


Рисунок 7 – 2 файл с результатами работы программы с 2 парами потоков.

199998 элементов, 4 пары потоков:

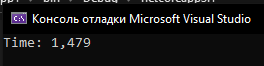


Рисунок 8 – время работы программы с 4 парами потоков на языке C#.



Рисунок 9 – время работы программы с 4 парами потоков на языке C++.

199998 элементов, 8 пар потоков:

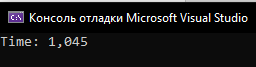


Рисунок 10 – время работы программы с 8 парами потоков на языке C#.



Рисунок 11 – время работы программы с 8 парами потоков на языке C++.

199998 элементов, 16 пар потоков:

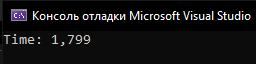


Рисунок 12– время работы программы с 16 парами потоков на языке C#.



Рисунок 13– время работы программы с 16 парами потоков на языке C++.

# **6. Выводы.**

В ходе работы была написана программа, с помощью которой осуществляется разложение на простые множители заданных в файле чисел. Разложение осуществляется с помощью пар потоков. Для передачи данных внутри пары потоков используется массив.

Есть 2 типа потоков:

1) Reader считывает информацию из файла и передаёт её в поток writer;

2) Writer считывает и преобразует данные затем записывает результаты в выходной файл.

Построена диаграмма последовательности, а также продемонстрированы примеры работы программы с различным количеством пар потоков. В результате работы программы было выяснено, что при увеличении числа потоков уменьшается время работы программы, а после превышения числа потоков процессора идёт увеличение времени работы программы, так как потоки ожидают освобождения вычислительной мощности процессора.

Исходя из пар рисунков 4,5’ 8,9’ 10,11’ 12,13 программа на языке C# работает медленнее, так как он более высокого уровня в сравнении с C++. Код C# может работать быстрее за счет оптимизации под платформу вовремя JIT компиляции. Или, например с тем, что ядро .Net Framework само по себе очень хорошо оптимизировано. С другой стороны, весомым аргументом является то, что С++ компилируется непосредственно в машинный код и работает с минимально возможным количеством хелперов и прослоек.