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 – рпк1, пп

Выполнили
студенты группы АВТ-813:
Кинчаров Данил
Пайхаев Алексей
Чернаков Кирилл
Преподаватель:
Ландовский Владимир Владимирович,
к.т.н., доцент кафедры АСУ

г. Новосибирск
2020 г.

Содержание

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.

рпк1 - модель с равноправными узлами, представляющими собой пары потоков, первый читает входной файл и вычисляет результаты, второй записывает выходной файл. Файл результатов общий, каждый результат записывается отдельно (каждое обращение к файлу записывает один результат).

пп - проверка на простоту, исходные данные - число, результат - да/нет.

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, соответствующий мьютекс для осуществления корректной параллельной работы потоков при чтении элементов массива и записи результатов в общий выходной файл, а также условие для пассивного ожидания результатов в массиве.

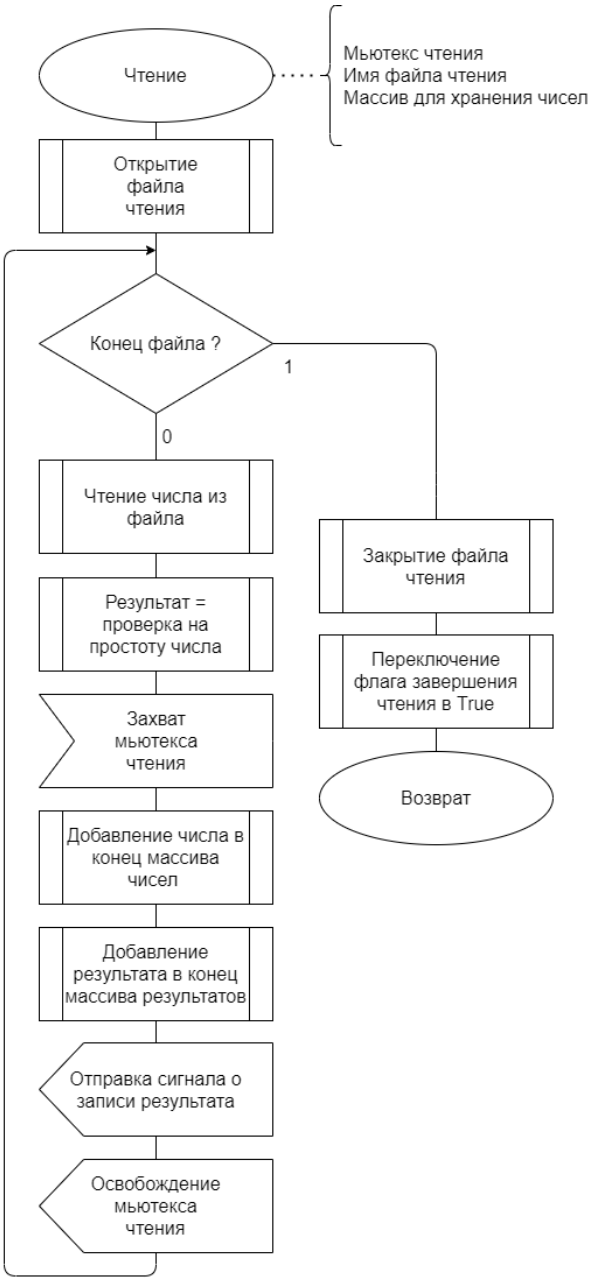
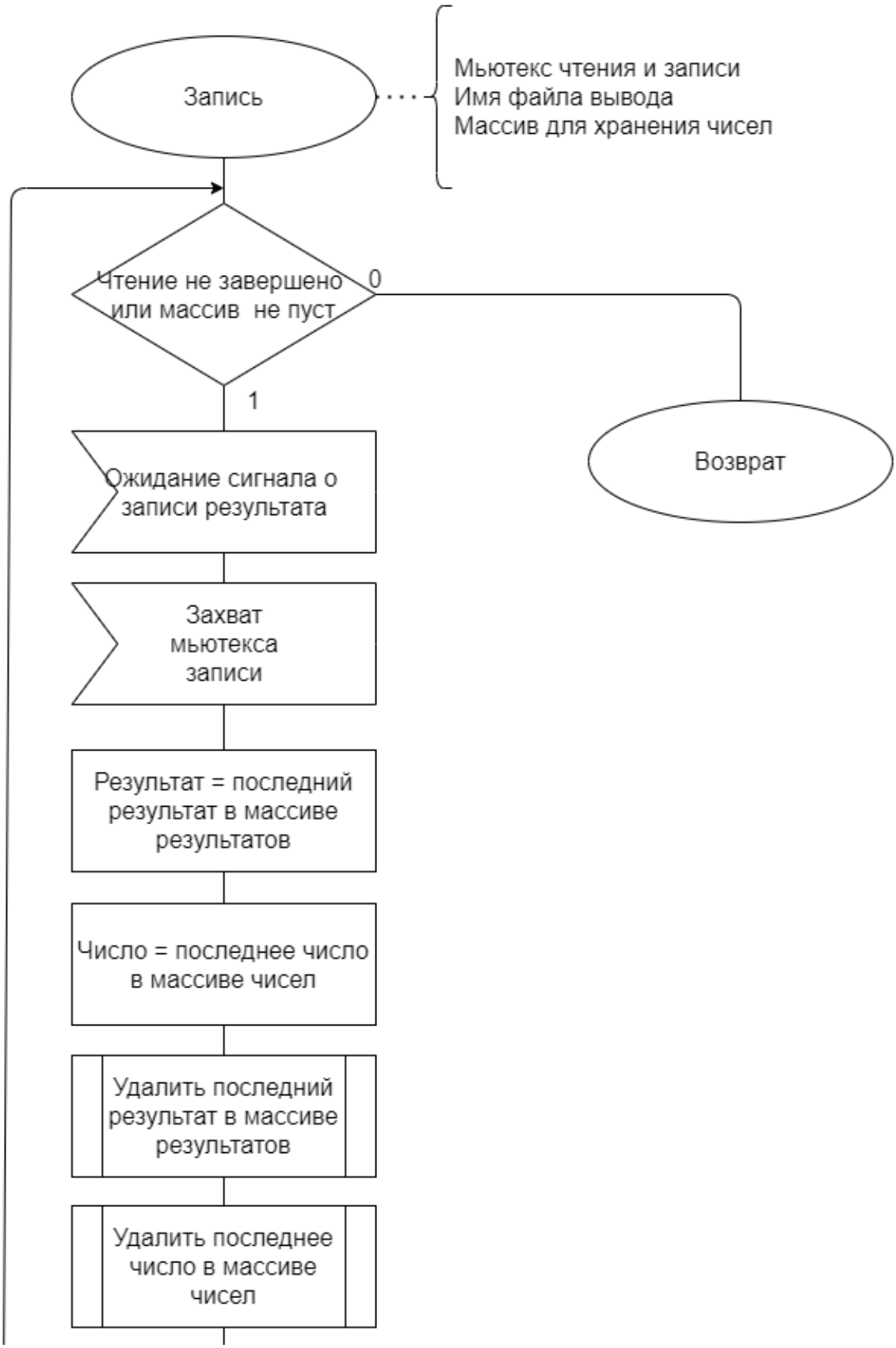


Рисунок 1 – Блок-схема работы потока, который читает входной файл


Запись
Мьютекс чтения и записи
Имя файла вывода
Массив для хранения чисел
Чтение не завершено или массив не пуст
0
1
Возврат
Ожидание сигнала о записи результата
Захват мьютекса записи
Результат = последний результат в массиве результатов
Число = последнее число в массиве чисел
Удалить последний результат в массиве результатов
Удалить последнее число в массиве чисел

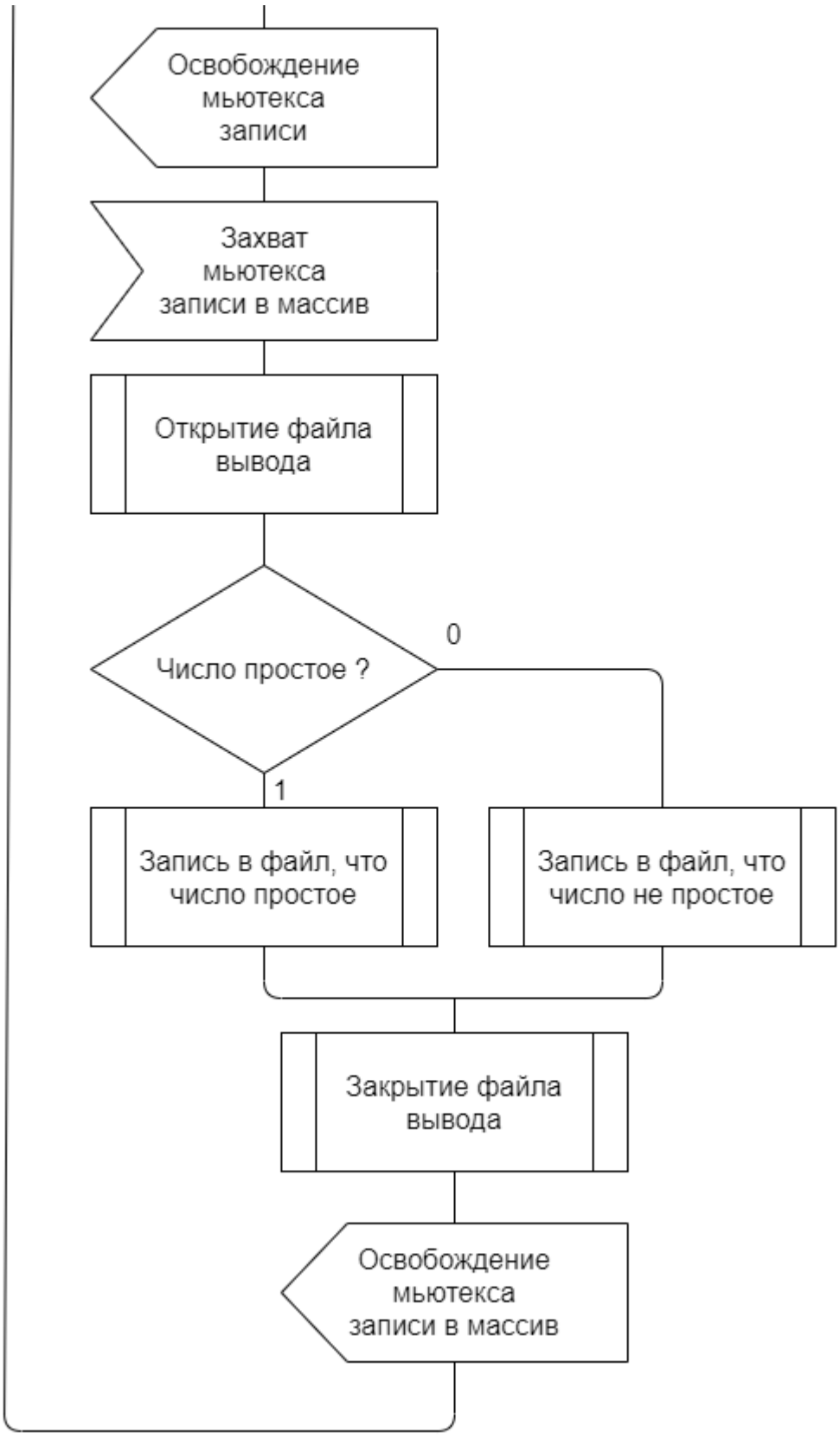


Рисунок 2 – Блок-схема работы потока, который записывает выходной файл и вычисляет результаты

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> numbers;
        public List<bool> results;
        public Mutex readMutex;
        public Mutex writeMutex;
        public Mutex writeFileMutex;
        public Semaphore waitSem;
        /*mutex*/
    }
    class Program
    {

        static void Main(string[] args)
        {
            Stopwatch timer = new Stopwatch();
            timer.Start();

            int numPair = 8;

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

            BinaryWriter inputFileW = new BinaryWriter(File.Open("input.bin", Fil
eMode.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", Fil
eMode.Open, FileAccess.Read), Encoding.ASCII);
            StreamWriter outputFileR = new StreamWriter(File.Open("output.bin", F
ileMode.OpenOrCreate, FileAccess.Write));

            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 = new argument();

            temp.fileInput = inputFileR;
            temp.readMutex = new Mutex();
            temp.writeFileMutex = new Mutex();
            temp.fileOutput = outputFileR;

            for (int i = 0; i < numPair; i++)
            {
                temp.numbers = new List<int>();
                temp.results = new List<bool>();
                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;
                bool result = isPrimeNumber(number);
                info.writeMutex.WaitOne();
                info.numbers.Insert(0, number);
                info.results.Insert(0, result);
                info.writeMutex.ReleaseMutex();
                info.waitSem.Release();
            }
            info.numbers.Insert(0, 0);
            info.waitSem.Release();

            return;
        }


        public static bool isPrimeNumber(int x)
        {
            for (int i = 2; i <= Math.Sqrt(x); i++)
                if (x % i == 0)
                    return false;
            return true;
        }

        public static void writeOut(Object obj)
        {
            int number;
            bool result;

            argument info = (argument)obj;

            do
            {
                info.waitSem.WaitOne();
                info.writeMutex.WaitOne();

                number = info.numbers[0];
                result = info.results[0];

                info.numbers.RemoveAt(0);
```

```
                info.results.RemoveAt(0);

                info.writeMutex.ReleaseMutex();

                info.writeFileMutex.WaitOne();
                if (result == true)
                {
                    info.fileOutput.Write(number);
                    info.fileOutput.Write(" - simple \n");
                }
                else
                {
                    info.fileOutput.Write(number);
                    info.fileOutput.Write(" - not simple \n");
                }

                info.writeFileMutex.ReleaseMutex();


            } while (number != 0);

            return;
        }
    }
}
```

4. Описание процесса работы программы с использованием диаграммы последовательности.

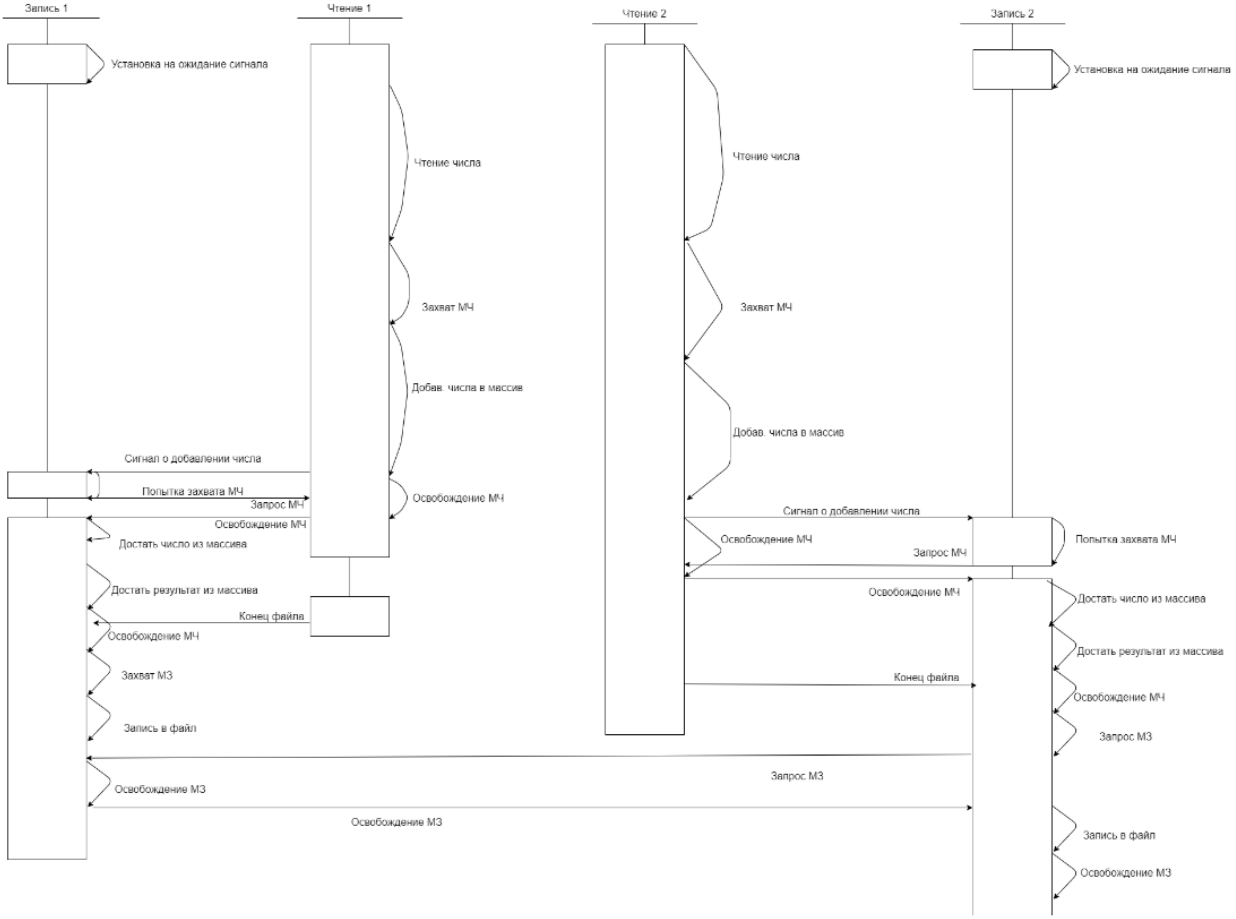


Рисунок 3 – Диаграмма последовательности типовой ситуации, в которой 2 пары потоков.

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.

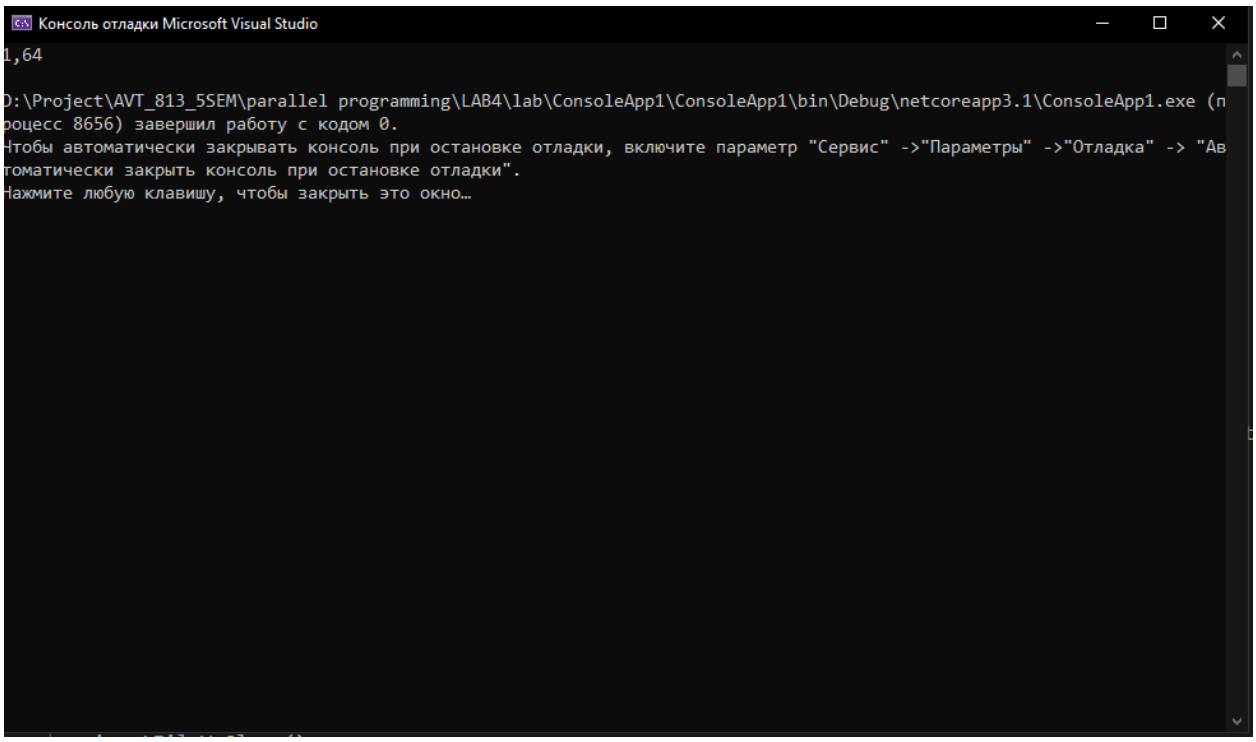


Рисунок 4 – время работы программы с 2 парами потоков на языке C#.

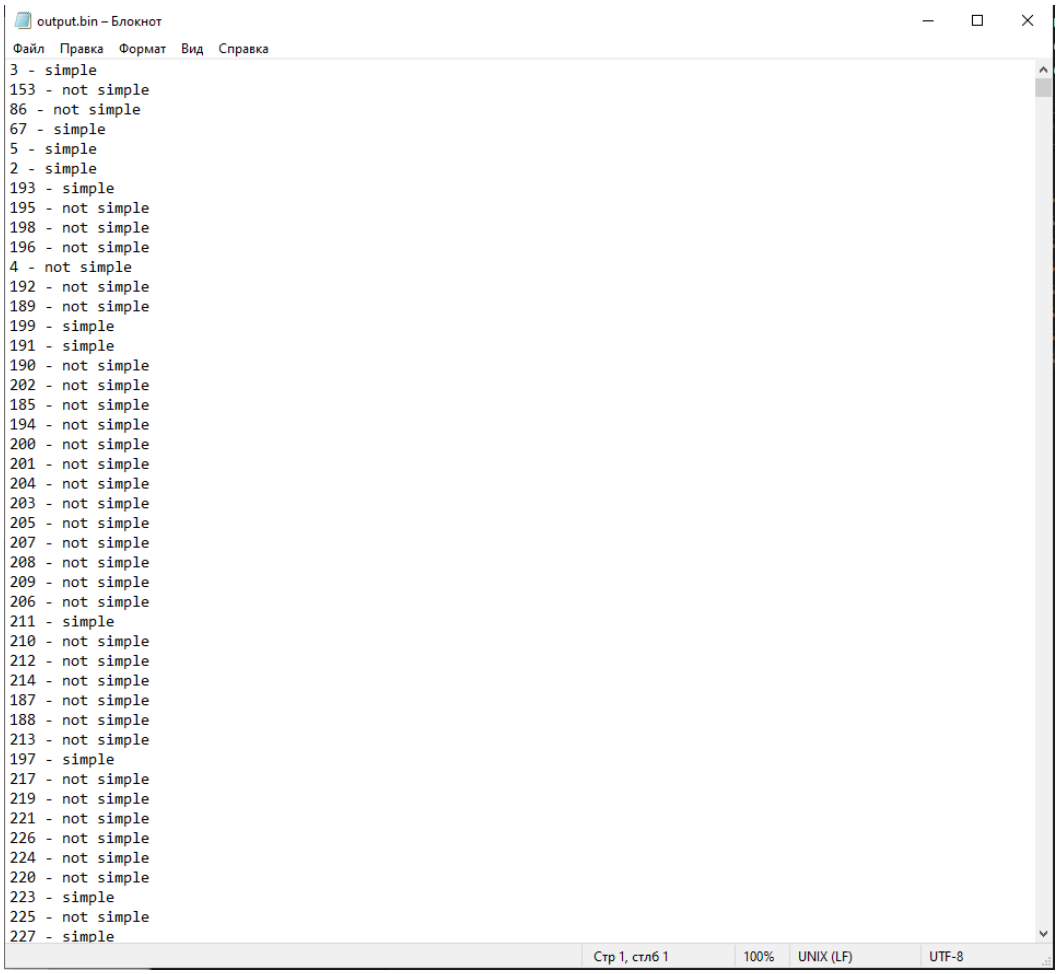


Рисунок 5 – Файл с результатами работы программы.

6. Выводы.

В ходе работы была написана программа, с помощью которой осуществляется проверка чисел на простоту, заданных в файлах чтения. 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 вычисляет результат проверки затем передает в поток writer;

2) Writer записывает результаты в выходной файл.

Построена диаграмма последовательности, а также продемонстрированы примеры работы программы с различным количеством пар потоков. При увеличении количества файлов, то есть пар потоков, время выполнения программы не увеличивается прямо пропорционально, так как за чтение данных из входных файлов отвечают параллельно работающие потоки.

В целом время выполнения программы увеличивается за счёт общего файла вывода. Можно заметить, что 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 C++ компилируется непосредственно в машинный код и работает с минимально возможным количеством хелперов и прослоек.